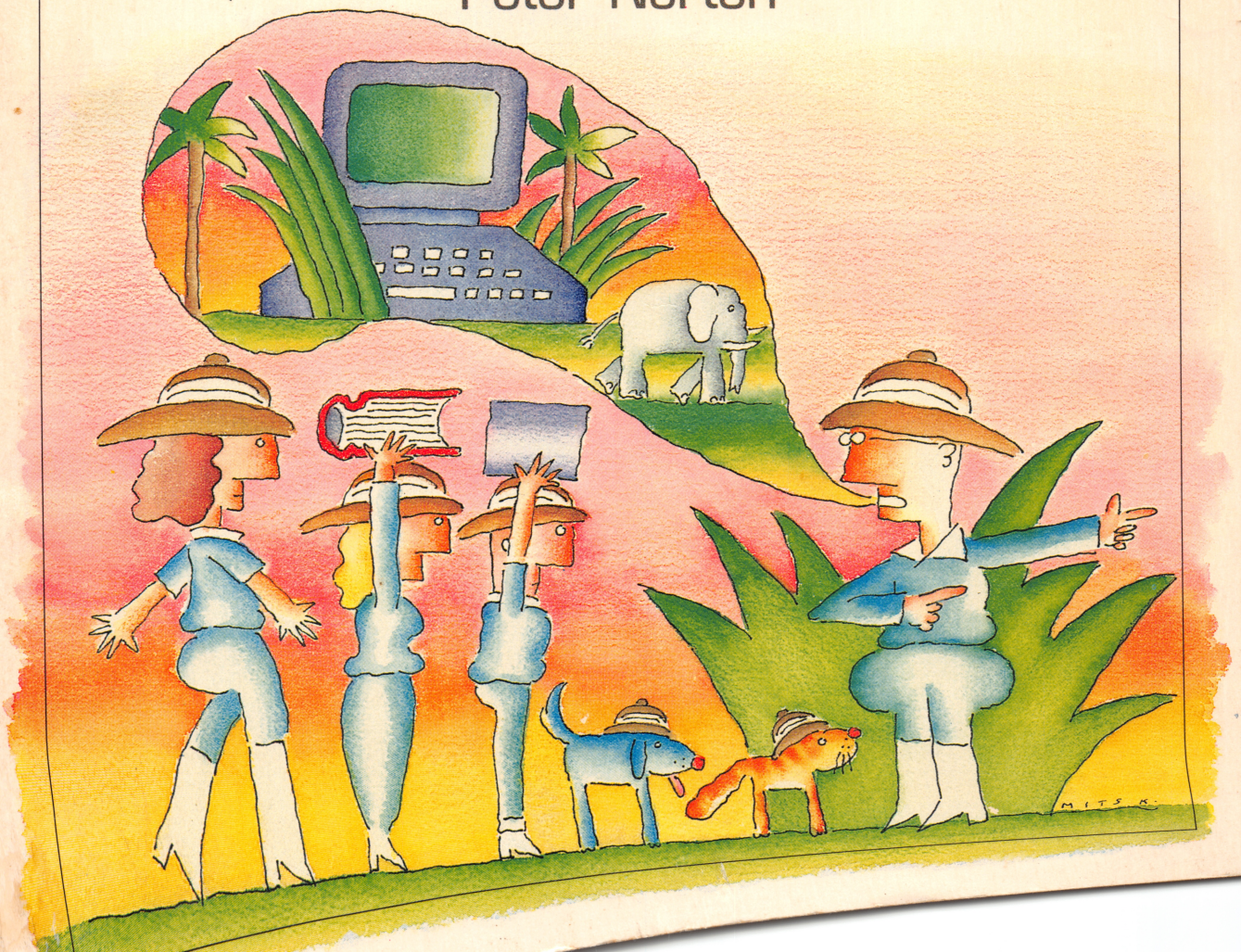
MICROSOFT® PRESS

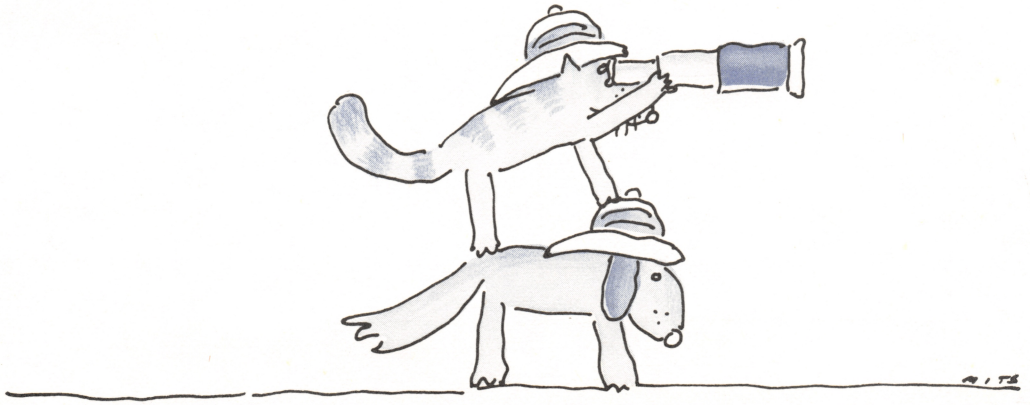
DISCOVERING The IBM® PC/Jr.™ HOME COMPUTER

Peter Norton



DISCOVERING

The IBM PC/Jr. HOME COMPUTER



DISCOVERING
The IBM PC/Jr. HOME COMPUTER

Peter Norton

Illustrated by Mits Katayama

MICROSOFT PRESS

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
10700 Northup Way, Bellevue, Washington 98004

Library of Congress Cataloging in Publication Data
Norton, Peter, 1943-
Discovering the IBM PCjr home computer.
Includes index.
1. IBM PCjr (Computer) I. Title.
II. Title: Discovering the I.B.M. P.C.jr. home computer.
QA76.8.I2593N66 1984 001.64 84-4530
ISBN 0-914845-01-2

Printed and bound in the United States of America

1 2 3 4 5 6 7 8 9 DODO 8 9 0 9 8 7 6 5 4

Distributed to the book trade in the United States and Canada by Simon and Schuster, Inc.

To Tom and T.J., John, Don and Jeanette, and cousin Bill.
It never would have happened without you.

Introduction ix

1
Computer Basics and Computer Talk 1

2
A Tour Around the PCJR 17

3
Starting Up the Computer 33

4
Games: Fun and More 43

5
Education: Learning in All Sorts of Ways 53

6
Serious Business: Putting Junior to Work 59

7
Communications: Junior on the Phone 73

8
Programming: Making Your Own Magic 85

9
Getting a Taste of BASIC 103

10
Working with DOS 119

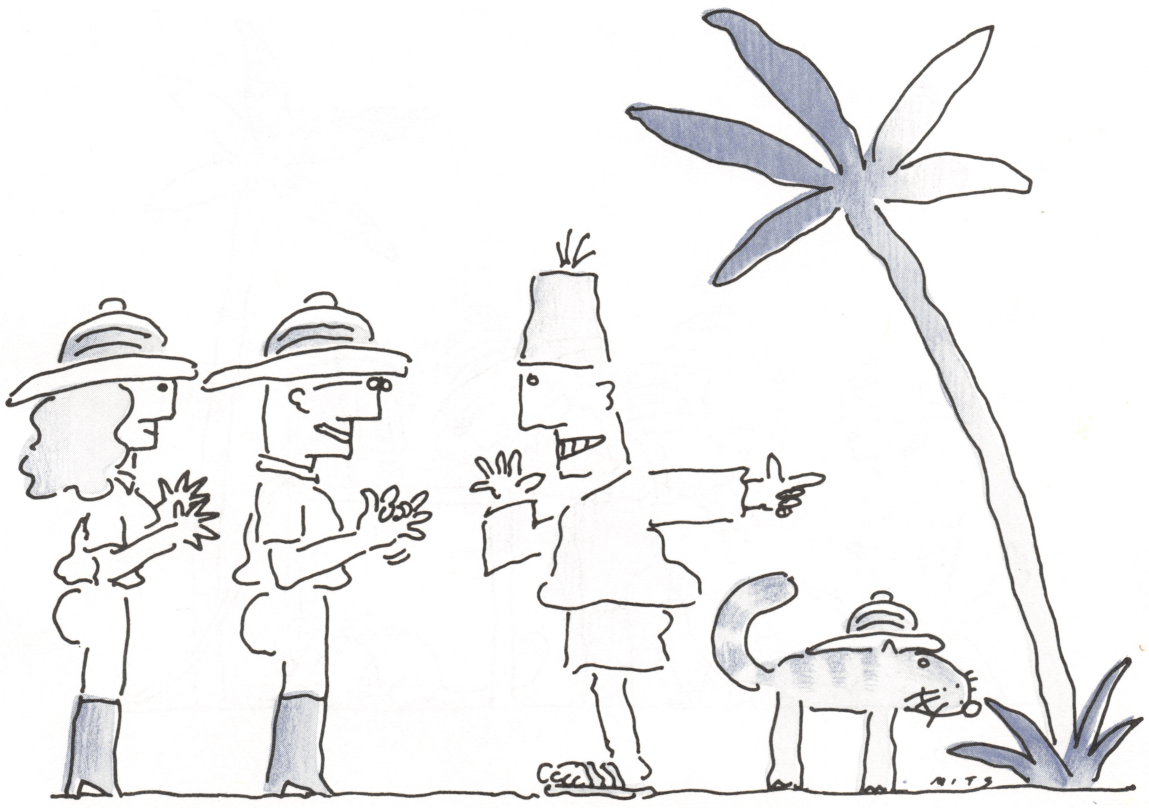
11
Understanding the Workings 139

12
Our Junior and the Rest of the Family 149

Index 165

A VOYAGE OF DISCOVERY

We're going to begin
a wonderful adventure, a safari
to discover the marvels of
the IBM PCjr.



BEGINNING THE SAFARI

You'll learn the basics of what computers are and how they work, and then you'll discover the many ways you can use the IBM® PCjr™ at home and in your work: for fun, for education, for school and business homework, and much more.

This book will be your guide, your map, to help you find your way. It will take you on a tour of all the parts of the PCjr, explaining how they fit together and what each part does. It will describe the optional equipment for the PCjr, and from there you'll go on to explore computer games, practical applications, and programming. You'll also see how your Junior's ability to communicate over the telephone opens up enormous possibilities in your use of computers. And at the end of your journey, we will look back on your adventure and use what you have learned to see how the PCjr fits into the whole range of IBM personal computers.

For myself, I've been working with all kinds of computers for over a dozen years. When the IBM® Personal Computer—the big brother of our PCjr—first appeared, I bought one as fast as I could. I fell in love with it, and I've loved explaining it to people. Now, though, I've got a new love, the PCjr, because it's such a completely wonderful little computer. My mission now, and my great joy, is to take you on a safari of discovery, guiding you into the mysteries of our PCjr.

"My turn."

THE GUIDE TELLS A SECRET

Would you like to know a secret? This is one that makes me chuckle. Plenty of families today are debating whether it's really worthwhile to get a home, or personal, computer at all. These families are asking themselves, "Is it truly worth the one or two thousand dollars that a complete computer costs?" The secret that makes me laugh is that, when a family has more than one school-age child, or when a couple both have professional jobs, a single computer isn't justified—several computers are.

That thought might seem astonishing at first. After all, computers aren't like toothbrushes. They aren't so inexpensive that you don't have to give a second thought to everyone having their own. Yet, once you begin to realize the potential of having a home computer like the PCjr, you'll find that its uses grow in many ways. If several people in your household will end up really using the computer, then you may end up needing more than one.

Let's put it another way. How many people in your household have individual desks—for schoolwork, work brought home, home financial management work, I'm-starting-my-own-business work? The chances are that most of those desks would be more effective with computers sitting on them.

"Everyone needs one."

A STARTING POINT

To help out, as a guide for those of you who may be uncertain about just what is what with these electronic marvels, Chapter 1 will take you on a quick walk through the thickets of computer terminology. Then, for those of you who are deciding what computer gear you might need, Chapter 2 will discuss the kinds of equipment—computer hardware—and the kinds of programs—computer software—that you'll need for the various things that you may want to do with your computer. As part of that discussion, you'll see what you need when you have just one computer, and what you can share when you have more than one. I'll try to help you avoid wasting money by buying too much, and wasting time by buying too little. And once we've covered that, we can head off on our journey into personal computing itself.

Chapter 3 will show you how to set up and start the PCjr and Chapters 4 through 7 will give you a look at games, education, home and business applications, and communications. In Chapters 8 and 9 you'll explore programming; in Chapters 10 and 11 you'll move on to DOS and the inner world of the PCjr itself; and, as mentioned earlier, in Chapter 12 you'll see how the PCjr fits into the realm of IBM personal computing—past, present, and future.

In the early days of home computers, the computers truly were toys. Nothing wrong with that—the computer is just about the best toy ever. But when people tried to use toy computers for more serious things, it just didn't work out. As a consequence, plenty of those computers ended up sitting on closet shelves when the amusement of their games wore off. This experience has led lots of people to doubt the value of buying a home computer. But this doubt is based on experience with computers that had little real capability.

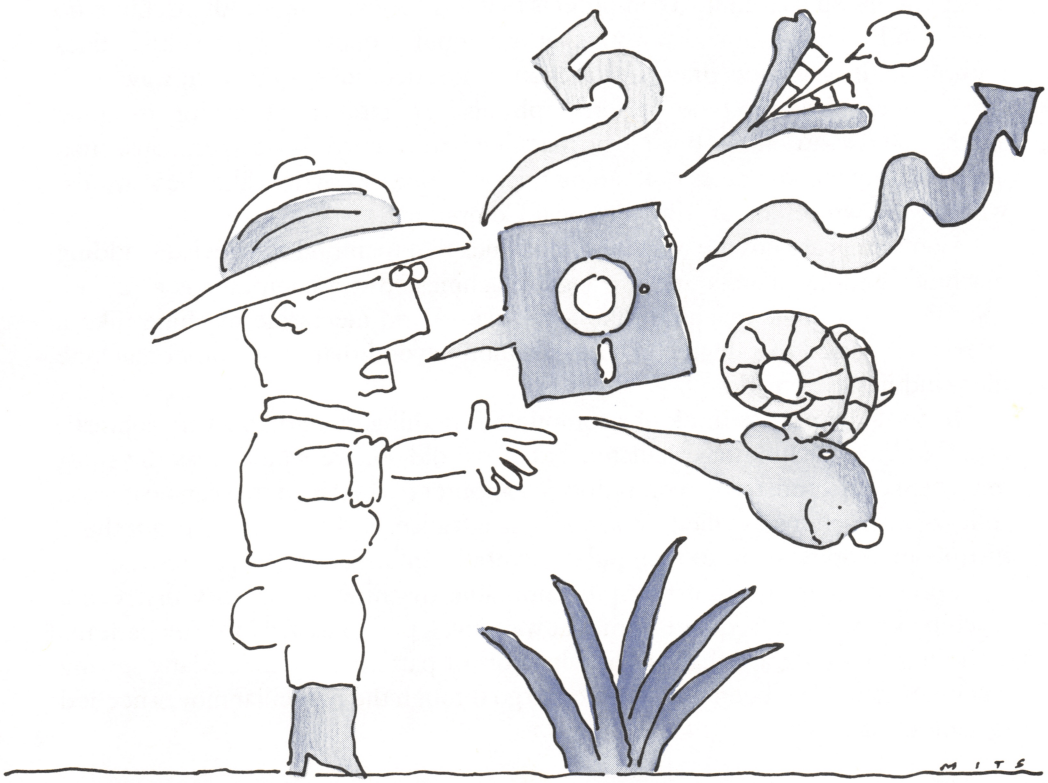
The situation is very different with the IBM PCjr. While the PCjr is a marvelous game player and a wonderful toy, it is also a very capable, serious-minded computer, able to perform a majority of the tasks that more expensive professional computers, like the IBM PC and XT™ do. We'll be showing you plenty you can do with a PCjr to get things done.

So, let's begin our safari. Let's plunge into the jungle of computing and discover the IBM PCjr.

1

COMPUTER BASICS AND COMPUTER TALK

Anthropologists sometimes call us "man, the tool maker" to emphasize that one of our most important distinguishing characteristics as a species is our ability to create tools to enhance our lives.



THE IDEA OF A COMPUTER

Before we can begin our voyage of discovery, in quest of the IBM PCjr, we must know what we are stalking. Just as a safari needs some basic equipment before it sets out, we need some basic understanding of the ideas and terminology used with personal computers, like our PCjr, before we're really ready to pursue it further. So in this chapter, we'll cover some computer fundamentals.

If you already understand what personal computers are all about, you can skip to the next chapter, where we'll begin discussing the details of the PCjr. If you're just starting to work with computers, read this chapter closely. And if you feel you're in between, skim through what we have to say here, to be sure you understand it all.

What a Computer Is

Of all the tools that we have created, from screwdrivers to sewing machines, the most flexible and powerful is the computer. A computer is a machine, but it's not like an ordinary mechanical machine. First, a computer is mostly electronic—just as a telephone or a television is electronic; it is not mechanical. But that isn't the most important thing that sets a computer apart from other kinds of machines. What is really special about a computer is that it is a *logic* machine. All machines do some kind of work, and the work that a computer does is logic work: It does arithmetic, it constructs drawings, it even makes decisions, in its own way.

A mechanical machine performs physical operations—a sewing machine stitches, and a saw cuts—but a logic machine performs symbolic operations, such as calculating income taxes or keeping track of things we write (like these words, which were written on an IBM Personal Computer).

Computers are not the only logic machines that mankind has devised; adding machines and calculators are also logic machines. But a computer is a unique combination of logic machine (like a calculator) and electronic machine (like a television), raised to a degree of sophistication beyond that of any other machine mankind has ever made.

It would be easy to think of a computer as nothing more than a very sophisticated adding machine or calculator, but if we did so, we would miss the most important characteristic of a computer: A computer is a machine that can be given a series of instructions—called a *program*—in advance, and can then carry out those instructions, on its own, to manipulate symbolic information.

Computers are not the first programmable machines. The very first was a machine known as a Jacquard loom that was developed to weave intricate patterns according to a program that told it what sort of pattern to weave. Many sewing machines, too, have been programmed to go through the particular moves needed for fancy stitches.

What is special and unique about a computer is that it is a *programmable logic* machine that combines the ability to work with symbolic information—both numbers and letters—and the ability to be programmed to perform a great variety of tasks.

A Handy Analogy

A good way of learning about the general idea of a computer is to compare it to a person at work, particularly to a handyman or a jack-of-all-trades. So to help you better understand what a computer is, let's draw an analogy between a computer and a handyman. Our handyman will serve as a model to help explain what a computer is, what its parts are, what it can do for us, and how it goes about doing it.

What makes this analogy between a computer, like our PCjr, and a handyman so apt is that the things a computer can do are much closer to human skills than are the things that other kinds of machines can do. A computer can read and write, in its own special way, just as we can read and write. A computer can manipulate numbers and words. A computer can even talk on the telephone, as we'll discuss later.

Like a handyman, a computer can do many things within the limits of its basic abilities. Also, like a handyman, a computer starts out with some skills and some knowledge, and it can "learn" new skills and new knowledge when necessary. Of course, a computer isn't exactly like a person, and we have to be careful not to stretch this analogy too far. But with a little care, we can get a very good idea of what a computer is and what it can do, by drawing a picture of the computer as a handyman.

KEY PARTS

Let's begin by looking at the main parts of a computer. There is computer *hardware*, and there is computer *software*.

Computer hardware is all the physical parts of the computer: its keyboard, where we type things in; its display screen, which can be a special computer display or can be a TV set; its printer, if the computer has one; and its innermost parts which, for the PCjr, are contained in the mysterious box called the system unit. Computer hardware is the equivalent of a handyman's body—eyes, ears, hands, and more—plus any tools that the handyman might have. A handyman might, for example, have a typewriter to type things for us, just as our computer might have a printer connected to it.

Computer software is all the programs that we might use on the computer. Software consists of instructions to the computer, telling it what to do and how to do it. Computer software is the equivalent of two kinds of skills our handyman has.

First, there is *system software,* the basic software that helps give the computer its fundamental skills; this software is similar to what our handyman learned as he grew up—how to tie his shoes, and how to read and write. Second, there is *application software,* which gives the computer specific useful skills; this software is similar to the training that our handyman can get to learn to do a particular job. For example, if we need to keep track of supplies, our handyman can learn to do it; similarly, we can get inventory software for our computer.

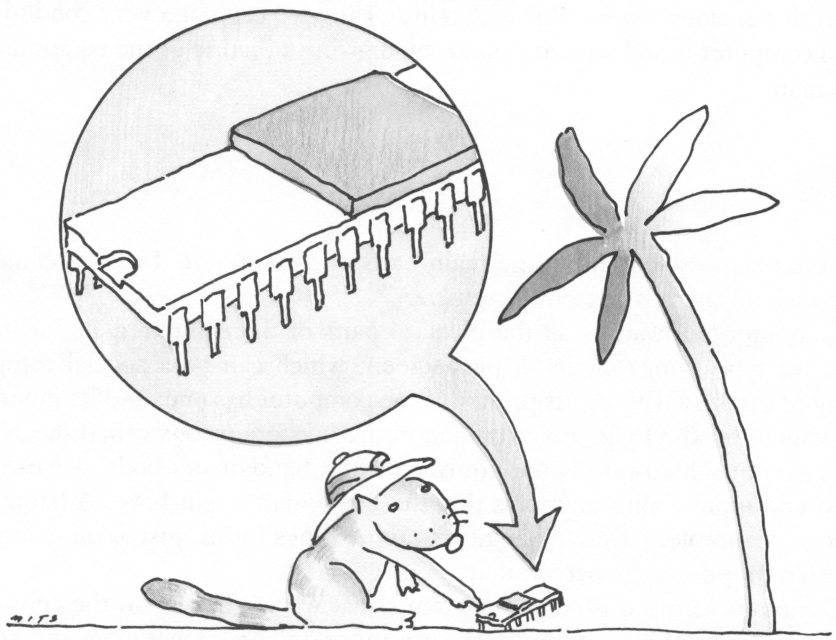
A Brain

No handyman could carry out our instructions without a brain. It's with our brain that we know how to do the thousands of things that people can do.

The brain of a computer is called a *processor* or, for a small computer like our PCjr, a *microprocessor.* A microprocessor is a tiny *silicon chip,* full of miniaturized transistors and other electronic parts, that has the ability to manipulate symbolic information.

A Place to Work

In order to get anything done, a person needs a place to work. A handyman needs a workbench, an explorer needs a clearing in the jungle to set up camp. The computer's workplace is called its *memory.*



"So this is a microprocessor."

The word memory means one thing when we're talking about people and something very different when we're talking about computers. Don't think of the computer's memory as you think of your own. The computer's memory is its workplace; it's the handyman's workbench, it's the explorer's jungle clearing.

When our computer is working for us, it works in memory, as a handyman works on the workbench. What does the computer put in that work space? The computer's memory is used to hold programs (which are like instruction manuals that the handyman would use), and it is also used to hold information, or *data* (like the lumber and other raw materials that a handyman would use) and work-in-progress (the half-built parts of the work that is being accomplished).

There is also another way that the computer's memory is like a handyman's workbench. Our handyman doesn't keep the workbench permanently cluttered: The bench is cleared off for each job. The same thing happens with computer memory. The computer's memory, for the most part, is used only temporarily for the task at hand. If today we play a game on the computer, and tomorrow we use the computer to write a report, the same memory—the same work space—is used each time. Used, and then cleared for other work. Some parts of the memory are set aside, permanently, for some special purposes—much as a workbench might have part of its space permanently taken up by a vise or a work lamp, but most memory is the erasable, workbench type.

How much work a computer can do for us depends on two main things: its speed and its memory size. A computer's speed is just like the speed at which a handyman can work. A computer like our PCjr is quick enough to do most of the things that we might want it to do in a reasonable amount of time, but there are some computer jobs that involve just too much work for the speed of a home computer and would take too long to get done.

An even more important factor in what we can ask our computer to do for us is the size of the memory. Since the computer's memory is its workplace, the computer must have enough memory to hold all the things that the job requires. If the instruction book and the lumber are too big to be managed on the handyman's workbench, the job can't be done; if the program and the data are too much to be managed in memory, the job can't be done.

So, it's mostly speed and memory capacity, especially memory capacity, that determine whether or not we have enough computer power to do what we want to do. Fortunately for us, most of what we are likely to want to do with a home computer is well within the capacity of computers like our IBM PCjr. But don't expect miracles.

Getting Down to Business

The next thing we'll look at is how the computer does work for us. To get our computer to do something, we have to tell it what to do, one way or another.

One way to tell a home computer like the PCjr what to do is to plug in a *software cartridge*. A software cartridge is a prerecorded program that connects directly to the computer's memory. When we plug a software cartridge into the PCjr, it's something like putting an instruction book on our handyman's bench. We've interrupted whatever the handyman might have been doing, and the handyman knows enough to follow the instructions in the book without another word. When we plug a software cartridge into our computer, the computer immediately starts carrying out the program on the cartridge. Plugging in a cartridge actually places the cartridge's program right into the computer's memory—right on the workbench, ready to swing into action.

But cartridges aren't the only way that a computer can get its work orders. When our computer uses diskettes, as the enhanced model of PCjr does, then it has access to a library of programs it can carry out. A cartridge generally has only one program on it, and when we plug in a cartridge, we tell the computer to use that program. A diskette, which we'll talk about shortly, can have several programs stored on it.

When we slip a diskette into our computer's disk drive, we've done something very similar to placing a reference library next to our handyman's workbench. The handyman hasn't been told to do anything, but some instruction books are now within reach; the computer hasn't been told to do anything, but its instructions are also within reach.

Once a diskette with some programs on it is placed into the computer's disk drive, we can give the computer a *command*. A command is simply the name of a program that we want the computer to carry out, or *execute*, for us. If the command is very ordinary—the computer equivalent of "tie your shoelaces"—the computer already knows how to carry it out. Some commands are always on tap in the computer, but most commands aren't. When we give our computer a command, it usually has to go looking for the program corresponding to the name of the command, and the usual place to look is among the programs that are recorded on a diskette.

By analogy, we tell our handyman to do something; on our computer, we type a command on the keyboard. The handyman looks in the reference library for the right instruction book; our computer searches the diskette in its disk drive for a program with the name of the command. The handyman puts the instruction book onto the workbench; our computer copies the program from diskette into memory. Once the program is in memory, the computer can start using it to do whatever kind of work the program instructs it to do.

Either way, with a cartridge or a diskette, we have to give our computer a program to carry out. Either we plug in a cartridge, which the computer gets to work on immediately, or we slip a diskette into the disk drive and then give a command to the computer, telling it to find a program on the diskette.

That, briefly, is how we get our computer down to business.

MORE ABOUT MEMORY

Now we have a general idea of what a computer is and some idea of how it works, but we need to start learning many more details. Partly, these are things we really need to know, and partly they are things we don't need to know, but that we ought to be familiar with, just so we don't get confused by some of those technical computer terms. There's probably no part of the computer that is associated with more confusing jargon than the computer's memory.

Bits

Computer memory is used to store information in the particular way that a computer has to have that information for its electronic needs. The smallest piece of computer memory, the computer's equivalent of the atom, is the *binary digit*, or *bit*. The decimal digits that we use every day are made up of the numbers 0, 1, 2, and so on, through 9. Binary means "having two parts" and bits are represented by just two numbers, 0 or 1. When we work with computers, we rarely talk about, or work directly with, bits, but bits are still the fundamental building blocks of all computer information. Incidentally, binary digits are particularly useful in computer work, because they can easily and naturally be used to represent the computer's two possible "states of being": ON (which is represented by 1) and OFF (which is represented by 0).

Bytes

What we do commonly talk about with computers is something called a *byte*. A byte is simply eight bits, gathered together into a single unit. A byte can be used to hold a single *character* of the alphabet, such as the letter A, so bytes are also sometimes called characters. Whenever you hear that some part of a computer can hold so many bytes, or so many characters, you can figure that the two terms mean essentially the same thing. Each byte in the computer's memory can take on any of 256 different values. If we do a little math, we can figure out why. A byte could have its eight bits set into any pattern of the two bit numbers, 0 and 1. We calculate the possible different combinations of the bits in a byte by raising two to the eighth power, which gives us 256.

The bytes that make up the computer's memory can be used in many different ways. A byte can be a number with 256 possible values. Or, we could reinterpret the same byte, and consider it a character. As an example, one particular byte value is 65 when our programs look at it as a number; when our programs look at it as a character, it's the letter A.

When we use bytes to store alphabetic characters, or *text*, as it is also called, we use a special coding scheme in which various byte values represent the various

letters of the alphabet, punctuation symbols, and so forth. Most computers, like our PCjr, use the coding scheme called ASCII, or American Standard Code for Information Interchange.

Just as we can use bytes to hold numbers or letters of the alphabet, we can also use them to hold our programs. Programs, numbers, and text are all stored in the same kind of computer memory. Depending upon what we are using any particular part of the computer's memory for, the computer interprets the information stored in the bytes as either programs or data, and interprets the data as either numbers or alphabetic text.

While bytes of memory can be handled separately by the computer, they are usually treated in groups of bytes—as many as are needed for any given purpose. For example, the word "For" at the beginning of this sentence is stored in three adjacent bytes of memory, one with the ASCII code for F, the next with the code for o, and then the final one with the code for r. However many characters are in the text we are working with, that is the number of bytes used to store the same text in the computer's memory.

When the computer works with numbers, it can also gather bytes together, to handle large numbers. When the computer uses two bytes together, they are called a *word*; but don't confuse this word with what "word" usually means in English. In PCjr-talk, a word is two bytes, treated as a single unit.

The Mysterious K

As shorthand, we usually talk of bytes in terms of K (short for *kilobyte*). One K is exactly 1,024, or about a thousand bytes. The number 1,024 is used for a simple technical reason: It's a round number in the computer's binary arithmetic system, and nearly a round number in our decimal arithmetic.

Computer memories have lots of bytes in them; for example, an entry-level PCjr computer has 64K of memory; that's 64 times 1,024, or 65,536 bytes. An enhanced PCjr has twice as much, 128K, so an enhanced PCjr has a workbench that is twice the size of the memory workbench in an entry-level PCjr.

Bits and bytes and K don't refer just to the computer's memory. They are used whenever we talk about computer information storage. So the same terms will be used when we talk about storing information on diskettes as well.

Two Kinds of Memory

Computers actually have two main kinds of memory: *random access memory*, or RAM, and *read only memory*, or ROM. The computer's working memory, our handyman's workbench, is random access memory, or RAM. This memory can be used for any purpose; it is used, then erased and reused whenever needed. When we say that an enhanced PCjr has 128K, we're talking about ordinary working

memory, or RAM. But, as you'll see in the next section, a computer needs some kinds of permanently recorded information, as well as erasable working storage; this is what read only memory, or ROM, is used for.

These two terms, RAM and ROM, are easy to confuse, since they differ by just one letter. To avoid confusion, I usually don't use the term RAM; I just say memory when I mean the computer's ordinary working RAM. When we're talking about the permanently recorded, read only memory, most people say ROM, and that's the term I'll use.

Read only memory, ROM, is used in two different ways in computers like our PCjr. One way is inside software cartridges; the programs in each cartridge are permanently recorded in ROM. The other way is inside the computer itself. This ROM is used to hold whatever programs are permanently installed in the computer as part of the computer's fundamental education. That's what we'll talk about next.

THE COMPUTER's EDUCATION

All the programs that a computer uses are like the education that a human handyman would have. Just as our handyman has gone through different levels of education for different purposes, so our computer has different kinds of programs that take care of the different things it needs to know. Since an understanding of these kinds of programs is very important to our understanding of computers, we'll take a look at what kind of programming education a computer has.

BIOS: An Elementary Education

You might think that the physical parts of a computer—its keyboard, its display screen, its printer—would be completely capable of doing their assigned tasks. Interestingly enough, that isn't the case. A certain amount of what it takes to get any part of the computer to work is done with the physical hardware, but the rest is done under the control of some special programming software.

This special control software is called the *BIOS*, short for Basic Input/Output System. It's called that because, for the most part, this software deals with the parts of the computer that take in information (*input*)—for example, from the keyboard—or put out information (*output*)—for example, to the display screen. In the PCjr, the BIOS is permanently stored in ROM, so it's sometimes also called the ROM-BIOS.

The BIOS is one part of what we earlier called system software, the software necessary to make the computer work in general (so that the computer can then get down to some specific business we want it to do).

Parts of the BIOS help the computer get its work done. Other parts test the computer when it is first turned on, and also when it is running, to make sure that

nothing is going wrong. These parts of the BIOS are called *diagnostics*. Our PCjr has a lot of diagnostics programs built into it; some of them run automatically, and some of them work just when we tell them to. (The *Guide to Operations* that comes with the PCjr tells us how to run these diagnostics and what to do when we find something wrong.)

The BIOS is the most fundamental part of the computer's education, and it is also the most rudimentary. When we grow up, we learn how to dress ourselves, how to use electric lights, and how to avoid getting run over in traffic. The BIOS provides our computer with an equivalent level of skills. Nothing sophisticated, but all the basics.

DOS: A College Education

If the BIOS is the computer's elementary education, then the computer's operating system is its college education. Like the BIOS, an operating system is part of the system software—part of the programming that makes the computer work in general. The operating system for our PCjr computer is called the *Disk Operating System*, or DOS for short. Like most operating systems (and as the name *disk* operating system implies), the DOS that our PCjr uses comes on diskette and is based on the use of diskettes; so we can only use DOS when we have the enhanced model of PCjr, with its diskette drive and extra memory.

An operating system, such as DOS, provides an enhanced level of skills for our computer, and many—not all, but many—of the programs that we'll want to use with our computer count on the computer having those skills on tap. From games to financial applications, these programs all rely on DOS.

Programs that do need DOS usually are clearly labeled "requires DOS" on the package, so we can generally tell which ones these are. Programs that don't use DOS aren't usually marked in any special way; they just don't say they need DOS. I like to call this latter type of program *stand-alone*, because it stands by itself, without DOS; there really isn't any standard terminology for this kind of program, though.

Most cartridge programs for the PCjr are stand-alone programs that don't use DOS, but that's not a universal rule. There are cartridge programs that work with DOS and when we come to the chapter on DOS, we'll see why. Some programs, particularly game programs, that come to us on diskette are stand-alone. On the whole, though, most programs that come to us on diskette are programs that need DOS in order to be used.

What does DOS do?

Why do some programs need DOS, anyway? And what is the difference between a stand-alone program and one that needs DOS?

Let's make an analogy with the kind of worker we might hire to help do certain things. If we needed some simple manual work done, we wouldn't need a worker who knew how to type and write shorthand. On the other hand, if we needed all kinds of office work done, we'd want to hire someone who had the special skills that secretaries and other office workers have.

When we use DOS with a PCjr, we get a computer that has lots of extra skills, particularly skills that have to do with keeping information stored on diskettes. DOS provides these standard skills, in a common and consistent format that can be used by a great many programs. DOS relieves our other programs of the burden of knowing how to do lots of things, such as how to work with diskettes. When a program is based on DOS, it's similar to our hiring a college-educated worker: We can count on the combination of the computer and DOS to know how to do things that the computer alone doesn't know.

Stand-alone programs can have all the skills that DOS has and, in fact, some of them do. But the people who wrote those programs had to duplicate a lot of the effort that went into creating DOS. Programs that rely on DOS don't have to duplicate that effort—they can let DOS take care of the routine stuff, while the programs get down to their own special tasks.

You'll find that many of the most important programs that you want to use with your computer count on having the support and skills of DOS.

Now let's take a peek at what computer programming languages are all about.

Learning Languages

Deep inside each computer, our instructions—the computer programs—are being carried out in a very cryptic and mysterious way. The actual form that computer programs take isn't designed for us to understand, it's designed for the speed and efficiency of the computer itself.

Yet computer programs are written by ordinary humans, and the programmers who create them have to be able to work with them in a practical way. To help with this work, computers have *programming languages*.

A programming language is simply a scheme that lets us express what we want the computer to do; after we've written a program, it is translated, one way or another, into the form that the computer needs.

The form that programs take when the computer carries them out is called *machine language;* as I've mentioned, it's very cryptic. When programmers want to write programs more or less in machine language, they use something very close to it called *assembly language*. Assembly language is almost exactly the same as machine language, except that when programmers want to add two numbers, they get to write ADD, instead of 83C120. A program called an *assembler,* or a fancier translator called a *macro assembler,* then translates the instruction into machine language.

Different computers can have completely different machine languages and, therefore, different assembly languages, unless they happen to use the same processor. For example, all IBM personal computers use the same microprocessor, so they all have the same machine language. The different models of Apple® computer use several different microprocessors, so they have several different forms of machine language.

High and low: More about languages

Assembly language is called a *low-level language,* because the programmer has to write a separate assembler instruction for each machine instruction. Since each separate instruction does just a little, the procedure can become very laborious.

High-level languages make things somewhat easier. With a high-level language, we can use one instruction to ask for a relatively complicated operation to be done, and that instruction will be translated into dozens or even hundreds of machine-language instructions. The savings in effort can be enormous.

The most commonly used high-level programming languages for personal computers are BASIC and Pascal. Other popular languages are Logo, FORTH, and C. Traditionally, large computers have used the languages COBOL and FORTRAN a lot, and these languages are also available for personal computers.

Each brand of computer tends to have its own versions of the high-level languages, which may not be exactly the same as the versions for another computer. The different versions, however, will have a lot in common. This similarity makes it easier to move programs from one computer to another when the programs are written in a high-level language than when they are written in assembly language.

In addition, several different versions of one programming language may be available for one particular model of computer, like our PCjr. These different versions may not all work exactly the same. For example, our PCjr, like all the IBM personal computers, comes with a simple version of BASIC built right into its read only memory. An extended version of the same BASIC is available on a cartridge.

Lots of other languages are also available for the IBM personal computers; some of them can be used on our PCjr, but others, such as IBM's COBOL, require a bigger memory workbench than our Junior has.

In the end, any programming language has to be translated into machine language. With high-level languages, this translating can be done by a program called an *interpreter* or by one called a *compiler.* Later, when we cover programming in more detail, we'll see what the difference is; interpreters and compilers work very differently, and so it turns out to be both interesting and worthwhile to understand them.

We have mentioned that the BIOS and the operating system, DOS, are system programs, because they are part of what makes the computer run, rather than

application programs, which do some particular job. The distinction doesn't have any significance; it just helps us understand the differences between types of programs. The programs that we call assemblers, interpreters, and compilers, which translate programming languages into machine language, fall somewhere in between system programs and application programs. Some people consider them system programs and some consider them applications: It's a matter of opinion or viewpoint. Let's consider them a little of both.

Job Skills

Application programs are the ones that we use to do some particular thing. These programs are the finishing touches to our computer's education; they are the specific job skills that our computer can acquire.

There are lots and lots of application programs that we can use on our computer. In fact, there are so many, and they are so interesting, that we'll be devoting all of Chapters 4 through 7 to discussing them, covering them subject by subject.

Playing games and learning with educational programs are among the many applications we use our computers for. There are lots of business and professional applications, as well: word-processing programs to help us write letters, reports, and term papers; accounting programs to help us with bookkeeping; home-finance programs to help us budget and plan; tax programs to help us with filing our income taxes; database programs to help us keep all kinds of records straight; and much more.

In addition, there are communications programs that make it possible for our computer to use the telephone to call other computers and gain access to a world of interesting and useful information. And then there are programming applications, which help us create our own programs.

In short, there is no end to the applications that we can put our computer to work on.

DISKETTE FUNDAMENTALS

We've been mentioning the *diskettes* that our computer can use, particularly in connection with the use of an operating system, such as DOS. We ought to take a look at just what diskettes are and how they work.

A diskette is a place where we store all sorts of computer information. Our computer can read information from the diskette, and can also write new information on it. Old information can be changed or simply erased, so that it is no longer there. Since it's relatively easy to erase information from diskettes, we should make

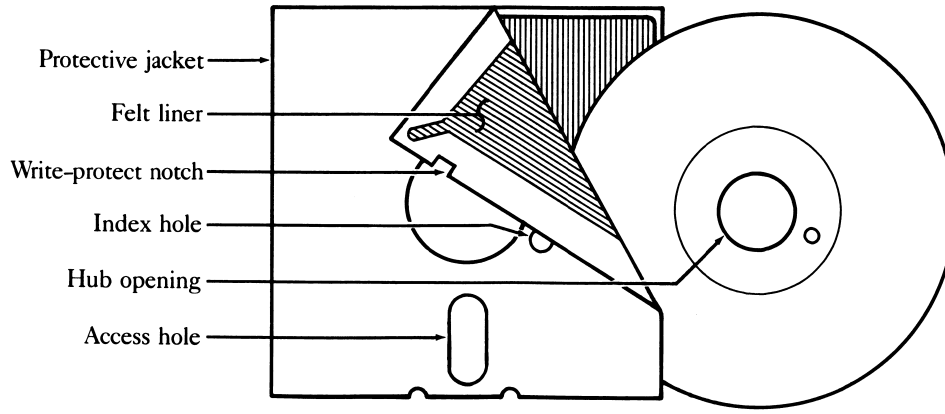


Figure 1-1. A diskette

backups, or copies of our diskette data, so we don't accidentally lose or damage our data or programs.

Physically, a diskette is a thin, flexible circle of plastic, coated just like recording tape, with magnetic material. You can see a drawing of one in Figure 1-1. For protection, a diskette is enclosed in a square jacket, which has a slot in it so information can be read from or written onto the surface by the computer.

A diskette fits into a *diskette drive*, shown in Figure 1-2, which is both a recorder and a playback mechanism for what is recorded on the diskette. The enhanced model of PCjr has a single diskette drive. Electronically, a diskette drive works like a tape recorder, writing (or recording) information on the diskette and reading (or playing back) information that was stored there. Physically, a diskette drive works more like a phonograph player, spinning the circular diskette inside its protective jacket, just as a phonograph spins a record. A magnetic *read/write head* passes over the surface of the diskette something like the tone arm on a phonograph, reading or writing information to or from the spinning diskette.

Diskette Technical Terms

There are lots and lots of technical terms that are used when diskettes are discussed. You don't need to understand them to use your computer. You only need to learn about them if you're interested in the technical side of computers. We won't be digging into these details here; when you're ready for them, you can find them in my more advanced book, *Exploring the IBM PCjr Home Computer* (Microsoft Press, 1984). What we do need to do here, though, is simply mention the most common terms, so you won't be confused when you run across them.

Diskettes come in several varieties. For example, our PCjr uses the kind that is 5¼ inches in diameter, a size sometimes called a *mini-diskette*. Information can be

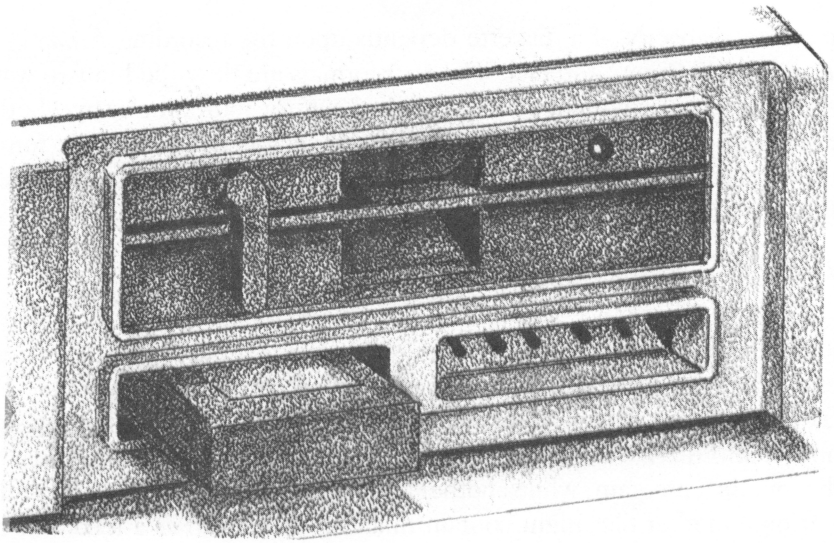


Figure 1-2. The PCjr's diskette drive

recorded in different densities; on our diskettes, information is recorded in *double density,* so our diskettes can hold more data than *single-density* diskettes can. Diskettes can be recorded on one or both sides; ours are double sided, and those are the ones we should buy, but we can also use *single-sided* diskettes if we happen to get them for one reason or another.

The actual information that is kept on the diskette is recorded in *sectors,* and the sectors are arranged in concentric circles called *tracks.* You can see a diagram of these details in Figure 1-3. Our PCjr uses diskettes with 40 tracks on them; each track has eight or nine sectors; each sector holds 512 bytes of data.

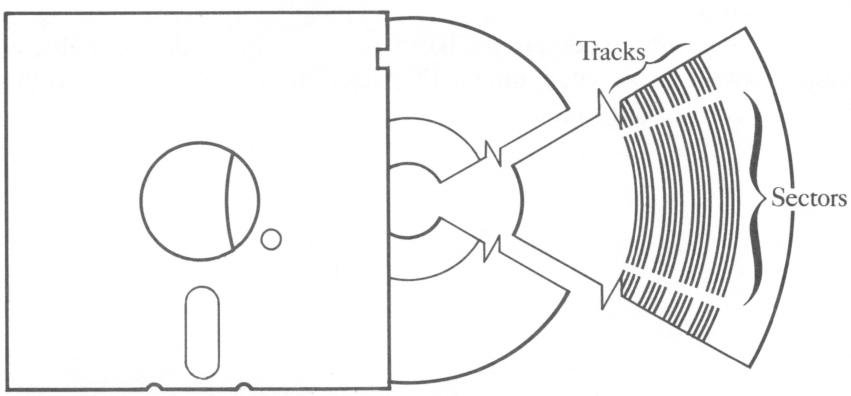


Figure 1-3. Sectors and tracks on a diskette

The storage capacity of a diskette depends upon the recording *format.* Our PCjr can use four different formats. The four formats are described in two ways: either by their size (which might be 160K, 180K, 320K, or 360K) or else by whether they are single or double sided, with eight or nine sectors per track. The format used most often with our PCjr is double sided, nine sector, which can hold a total of 360K of information for us.

Diskette Files

Much more important to us than those technical details is how we use diskettes. We organize the information that we keep on diskettes into *files.* The files on a diskette are analogous to the files we keep in a filing cabinet. We can store any computer information that we want to keep on a diskette in a separate file. One file might contain a program, while another file might hold the text of a report that we've written. Another file might contain financial data, such as a record of the entries in our checkbook.

Each file is identified by a *file name.* File names have a particular form, which we'll learn more about when we get to know how DOS works with diskettes. For now, it's enough to say that file names are short and sometimes consist of two parts separated by a period. To give you an idea of what they are like, here are some sample file names:

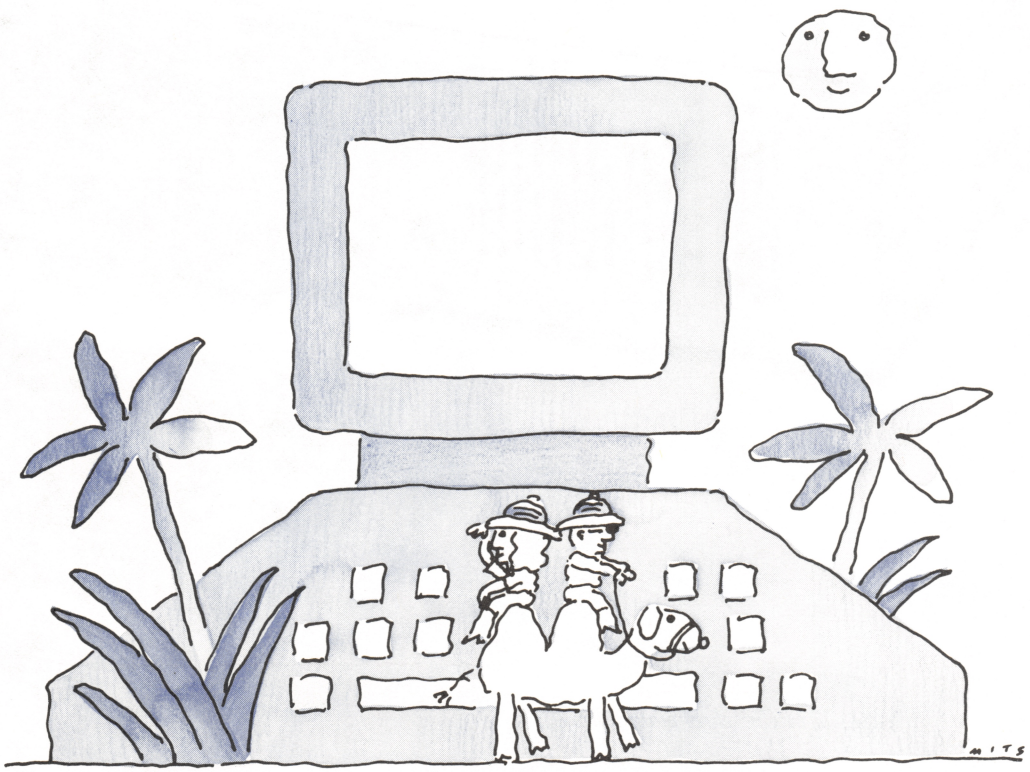
INFO
COMMAND.COM
SAMPLES.BAS
MINE
YOURS.TOO

Whenever we need to get to a diskette file, we must refer to the file by name. One of DOS's jobs is to keep track of diskette files, and to make sure that they don't get mixed up.

With this short introduction to the basics of computers under our belts, we're now ready to set off on our safari into the PCjr itself. In the next chapter, we'll look at the various parts of our Junior.

2
A TOUR AROUND THE PCJR

Now it's time for us to take a look at the parts of the PCjr and all the optional equipment we can connect to it, so we know what we have with this wonderful little computer.



THE PARTS OF THE PCJR

As you can see in Figure 2-1, there are three main parts to the PCjr itself. First, there's the heart of the computer, the box that contains most of what makes our computer a computer; it's called the system unit. We'll be talking a lot about what's in the system unit, and how all the parts in it work together.

The second part of our PCjr is the *keyboard*. Our Junior has a nifty little keyboard, with a feature that no computer has had before: a remote control that talks to the system unit by infrared light in the same way that a remote control works on a TV set. We'll cover more about the keyboard a little later.

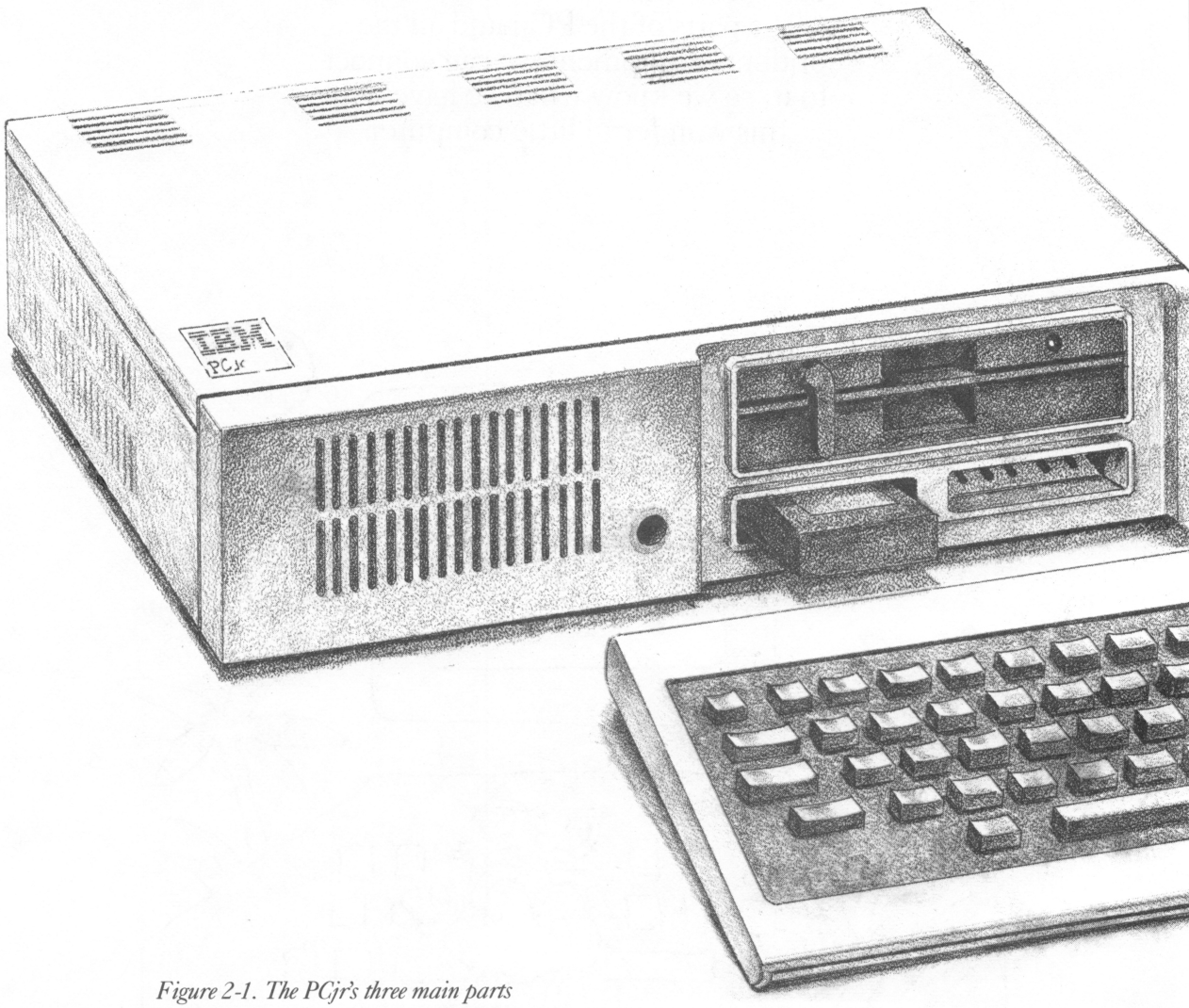
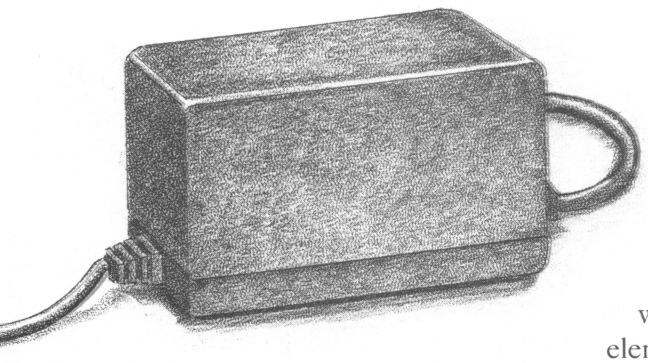


Figure 2-1. The PCjr's three main parts

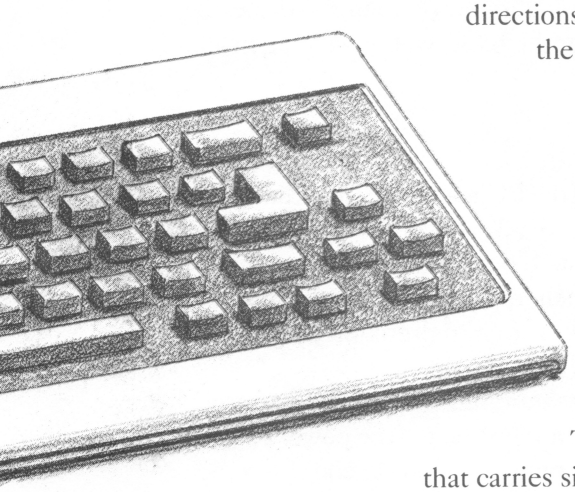
The third part of our PCjr is the *transformer.* Most pieces of electronic equipment, including stereo systems, have transformers that convert standard electric power to a lower voltage. Our PCjr has its transformer outside the system unit, instead of inside, and this location has a lot of advantages. With the transformer outside, there's no high voltage inside the system unit. So when we open the system unit, as we might do sometime, the procedure is much safer than if the transformer were inside. We are also safer if we ever spill anything on the computer. In addition, having the transformer outside makes the system unit lighter and easier to move around, and it makes our computer more reliable—heat from the transformer isn't cooking the delicate electronics inside the system unit.

The System Unit



The *system unit* is the heart of the PCjr. It contains the circuitry that makes our computer go. Included in the system unit are most of the parts we mentioned in the last chapter when we drew an analogy between the computer and a handyman: the microprocessor (the handyman's brain), the memory (the workbench), and the built-in programs (the elementary education).

If you want to look inside the system unit, you can pry its top off with a screwdriver. It's simple and safe to do; you'll find official directions on how to do it in IBM's *Guide to Operations* for the PCjr. I'll leave that for you to do when you feel curious and adventuresome. What we need to learn about now is what we can see on the outside of our Junior's system unit.



Across the front

When we look at the front of the PCjr's system unit, we see the five openings shown in Figure 2-2. On the left are ventilation slots, to help keep our Junior cool. In the middle is a little round circle that is the PCjr's "eye." This is how the computer sees the infrared light that carries signals from the keyboard. We have to make sure that this electric eye isn't covered by anything when we use the keyboard; if it is, the computer won't be able to see what we're typing.

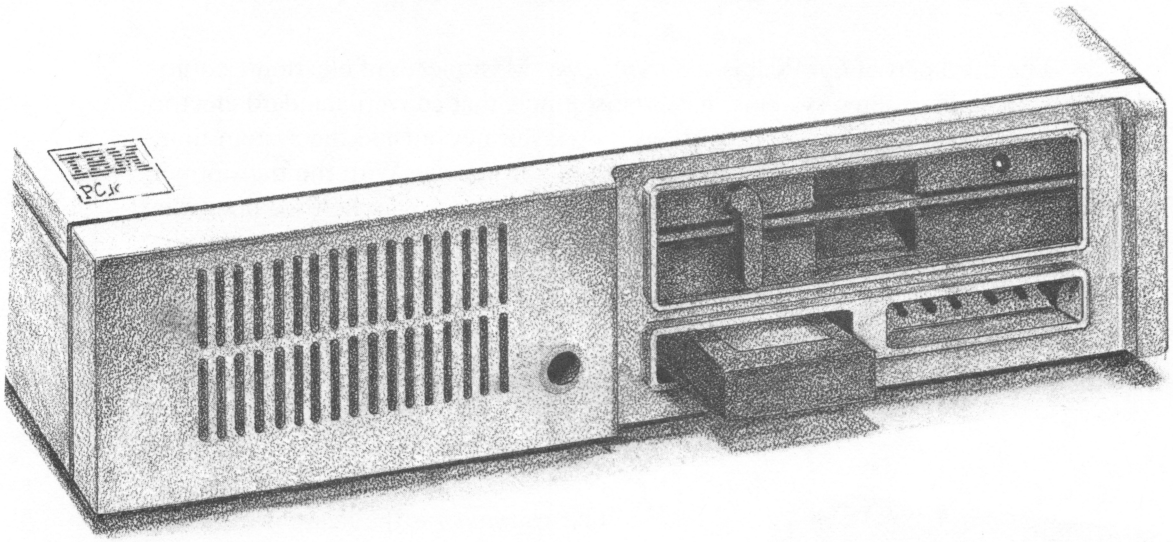


Figure 2-2. The front of our Junior's system unit

On the right-hand side of the front are three openings—one across the top, and two below. The top opening is for the computer's diskette drive. That's where we insert any diskettes that we're using.

Just below the diskette drive are the two openings where we plug in software cartridges. There are two cartridge slots, for the times we use two cartridges that are designed to work together. Both slots work in the same way, and it doesn't matter which slot we use to plug in any cartridge.

That's the front of the computer, where we plug in our programs—diskette programs and cartridge programs. Now let's look at the back, where we plug in all sorts of equipment.

Across the back

There are all sorts of sockets across the back of the PCjr, and they make all sorts of connections. You can see them labeled in Figure 2-3, and just listing them all will give us quite a good idea of what we can do with our Junior. Fortunately, the sockets are labeled with small, molded letters to help make it easier for us to keep all the connections straight. We'll go over the sockets from right to left as we look from the back.

At the top right is the power switch for the system unit; it's labeled with the international symbols for on and off, 1 (on) and 0 (off). Just below the power switch is the socket labeled P, for power; this is where we plug in the power cord from the external transformer.

Just to the right of the power socket is the audio connector, labeled A. Our PCjr can generate all sorts of sounds, and this audio connector lets us feed the sound

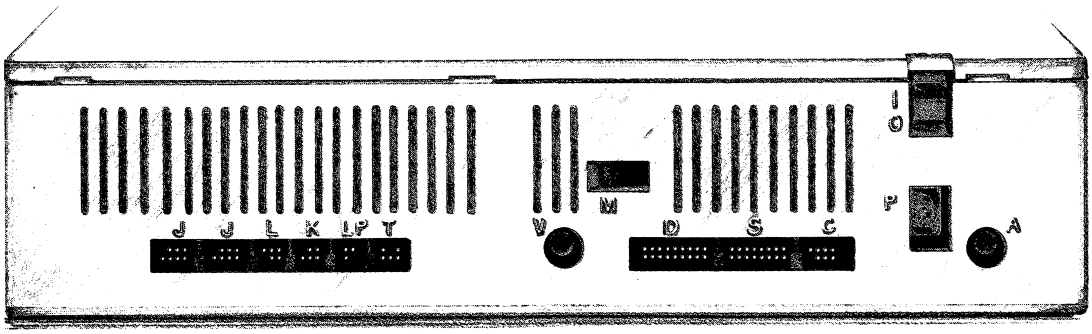


Figure 2-3. The back of the PCjr's system unit

signals into a hi-fi system if we want to. This is a jack-type socket, the kind used on most stereos.

To the left of the power socket is a row of three sockets labeled C, S, and D. The C socket is for connecting the PCjr to a cassette tape recorder; we can use cassette tapes to record programs or information. Cassette tapes aren't used much with the IBM personal computers, but it's nice to know that we can use them if we want to.

The next of this group of three is the S socket. S stands for *serial port,* which is a standard way of connecting a computer to all kinds of optional equipment, such as a printer or a modem. The serial port is also frequently called an *RS-232* connection; if you see that term used, now you'll know where the connection is on your computer.

The last socket in this row of three is labeled D, for *direct-drive display.* As we'll learn shortly, there are several sorts of display screens that we can use with our PCjr. One kind is known as an RGB (Red-Green-Blue), or direct-drive display. When we use that kind, it plugs in here.

The next socket to the left is a second jack-type socket labeled V, for video. This is for another kind of display screen, known as a composite video display. If you have a composite display, you plug it in here.

Above the V socket is a square socket labeled M. This is the modem connection, into which we can plug a telephone line. One of the options that fits right inside the PCjr's system unit is a modem, which does the job of translating our computer's communication signals into telephone signals and vice versa. The S-serial socket can be connected to an external modem and then to a telephone, but if we have the Junior's internal modem option, we can connect a telephone line directly to this M socket.

Next along to the left is a row of six sockets, similar to the three we saw before. The one labeled T is the last of the three display connections; we use this one when we use a TV set as a display screen.

Next is the LP socket, for a *light pen*. Few home computers have light pens, but if we get one, it plugs in here. We can touch the point of the light pen to our computer's display screen, and a program can tell where on the screen the pen is touching. Light pens can be used with such things as drawing programs and can be lots of fun for kids—and grownups, too.

The K, for keyboard, socket is next. If we don't want to use the keyboard's infrared light connection, we can connect the keyboard to the system unit with a cable, which plugs in here.

Next is the L socket, which is reserved for some later use.

Finally, there are two J sockets, for joysticks. Joysticks are used with computer games and educational programs.

That's a summary of all the things that can plug into the different sockets on the back of our computer.

And at the side

On the right side of the computer, there is one more connection, hidden behind a panel. This is called the *I/O channel connector*, a technical mouthful. This connector can be used for additional equipment that we might want to add to our computer. IBM makes a printer adapter that plugs in here; other options, from other manufacturers, can connect here as well.

THE ENTRY-LEVEL MODEL

IBM sells the PCjr in two versions. The entry-level model is less expensive, and with it, we get all the basic parts: the transformer, the keyboard, and the system unit.

Inside the system unit is 64K of memory. Although 64K is the minimum for a PCjr, our programs can do a lot with this amount of memory. And while 64K is the least that we can have on a PCjr, it's the most that can be put on many other home computers; some can't even accommodate that much.

An entry-level PCjr doesn't have a diskette drive, so the opening for the drive, on the front of the system unit, has a white plastic cover over it.

The entry-level PCjr lacks the two things that are really needed to make it a full-powered computer: a diskette drive and more memory. The main thing we can do with an entry-level model is use software cartridges, mostly games, but we can't do much more with it. That's why there are relatively few entry-level models for sale. But, with its $670 price tag, the entry-level PCjr is an inexpensive way to get started with IBM computing; and the most important thing about an entry-level model is that we can upgrade it to an enhanced model whenever we want.

THE ENHANCED MODEL

The enhanced PCjr has the two most important things that the entry-level model doesn't: a diskette drive and extra memory. Adding these options brings the price up to $1270, almost twice as much as the entry-level model, but it vastly increases the ways we can use our computer.

The enhanced model doesn't include everything that we could put on our computer. There's lots more, as we'll see shortly; but the enhanced model has these two most important options. Since they are so important, let's discuss them in more detail.

THE TWO KEY OPTIONS

The diskette drive and extra memory are key options because, as we have seen, they change our Junior from a whiz with software cartridges into a powerful little computer that can use DOS and most of the programs its big brothers, the PC and XT, can use. To use a diskette drive and DOS, we must have the memory expansion as well; this is the reason these two options usually come together, and why these two separate the entry-level model from the enhanced model. We can buy the diskette drive and the memory expansion separately, but these two options together are much more effective than either one alone.

The Memory Option

The memory-expansion option increases the PCjr's memory from 64K to twice as much, 128K. As we mentioned in Chapter 1, memory acts as the computer's work space. With more memory, the computer can take on bigger jobs.

As it turns out, the difference between 64K and 128K is just the amount of memory that is needed for most of the extra things we'd like our computer to do. If we had even more memory—which we can't have on the PCjr but can on the other IBM personal computers—it would add to the things we could do, but it wouldn't make as dramatic a difference as this second 64K of memory does.

There are two main things the memory expansion does for us: It opens up room for fancier uses of our computer's display screen, and it gives us just enough room to be able to use DOS, the disk operating system. Without 128K, we wouldn't be able to use DOS and the hundreds of programs, such as word processors and business applications, that work with DOS.

If we have an entry-level PCjr, we can add the memory expansion if we want to. It can be bought separately from the diskette option.

The Diskette Option

The diskette drive is the other key option for our PCjr computer. With a diskette drive, we can use all sorts of diskette programs—programs that work with DOS, and stand-alone programs that work by themselves.

The diskette option opens up vast realms for us and our PCjr computer, because it enables us to use the many IBM personal computer programs that are available only on diskette. Having a diskette drive also makes it possible for us to save our work. Whether it's a report we've written, or financial information we've developed, the diskette option makes it possible for us to do the work and then to save it to diskette.

The Other Options

There are lots of other options available for our Junior. More and more of them will appear as many different manufacturers make them available for us. What we'll look at here are the main options IBM provides for us.

Printers

One of the most important options we are likely to want for our computer is a *printer,* such as the IBM/Epson printer, so that we can get a paper record of our computer work.

Computer printers can be a real source of confusion, so we should take the time here to cover the most important things about them.

There are two quite different ways that a printer can be connected to a computer. When we get a printer for our computer, we need to be sure that both the computer and the printer expect the same connection.

There are two kinds of connections: serial and parallel. A serial connection allows the computer to pass information to the printer one *bit* at a time. A serial printer can be plugged right into the S-serial socket on the back of our Junior.

A *parallel connection* allows the computer to pass information to the printer one *byte* at a time. A parallel printer needs to be connected to a parallel printer adapter which, in turn, is connected to the I/O channel connector on the right-hand side of our computer's system unit.

If your computer has a parallel printer adapter on the side, then it is ready to use either kind of printer, serial or parallel. Without the adapter, you can use only a serial printer.

IBM sells both kinds of printers. There are also lots of other brands that can be used with our PCjr. IBM's inexpensive Compact Printer is intended to be used with our Junior, and it plugs into the serial connection through a special adapter

Figure 2-4. The IBM Graphics Printer and the IBM Compact Printer

cable; the printer and cable together cost only $215. IBM's Graphics Printer, on the other hand, is designed to connect to the parallel printer adapter and costs much more. The extra cost of the graphics printer, or others like it, is because it can print on any paper and make carbon copies as well; the IBM Compact Printer uses special, heat-sensitive, thermal paper that reduces its flexibility, and print quality too. Both of these printers are illustrated in Figure 2-4.

THIS IS AN EXAMPLE OF DOT MATRIX PRINTING

this is an example of dot matrix printing

Figure 2-5. Dot matrix printing

When we are choosing a printer for our computer, we need to decide what kind of printer we want, in addition to what kind of printer connection (serial or parallel) we need. You'll have to visit a store to learn about all the differences in printers, but a short summary here won't hurt.

The biggest difference in printers is in the type and quality of printing. *Dot-matrix printers* print characters made up of a series of dots as shown in Figure 2-5; the IBM printers are dot-matrix printers.

Most dot-matrix printers print rather crudely, but some can make very well-drawn characters.

Daisy-wheel printers, on the other hand, work more like a typewriter. They are named for the daisy-like shape of the print wheel, which has the individual letters, numbers, and so forth on the ends of lots of spokes (the daisy "petals") connected to a central "stem." Daisy-wheel printers usually print so well that they are referred to as *letter-quality printers.* Figure 2-6 shows an example of the print from a daisy-wheel printer.

Besides the print quality, there is also the printer speed to consider, and whether or not the printer can print graphics images, such as graphs and charts. Many dot-matrix printers can print graphics, but daisy-wheel printers cannot.

There are other factors to consider besides the ones we've covered here. Some printers have what's called a *"buffer,"* a feature that lets a printer accept and temporily hold information faster than it can be printed. Also, some printers, like IBM's Compact Printer, use only special, thermal printing paper. Another feature to consider is whether the printer is friction-feed, using individual sheets of paper,

THESE ARE EXAMPLES OF DAISY WHEEL or letter quality printing. these

STYLES CAN BE SET IN CAPITALS or lower case and have 3

SPACING VARIATIONS: 15, 12 and 10 pitch.

Figure 2-6. Daisy-wheel, or letter-quality printing

or tractor-feed, using pin holes on the sides of long sheets of perforated "computer" paper. Look at both to see which kind is best for you.

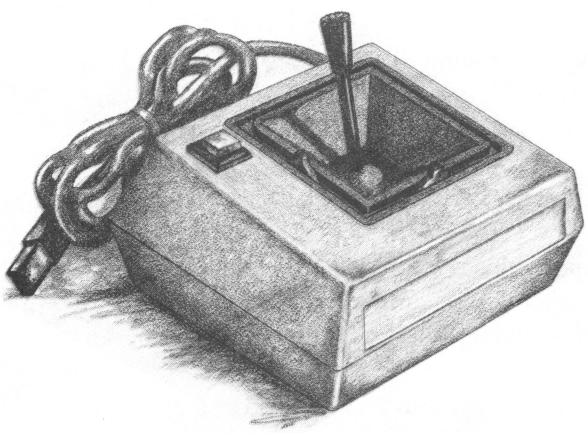
When you buy a printer, you should get advice to help you choose from among all these possibilities.

The Smart Modem

Most of the options for our computer plug into the outside, but there are three options that fit inside our PCjr's system unit. We know about two of them: the memory expansion and the diskette drive. The third internal option is the PCjr's smart modem.

When we want to connect our computer to a telephone line, so that it can work with other computers and remote computer services, we need a modem to translate between computer signals and telephone signals. We can plug an external modem into the S-serial socket, or we can buy the internal modem option that was designed especially for our PCjr.

There are several advantages to having our Junior's internal modem. One is that it is inside the computer, out of the way. Another is that it leaves our serial connection free for other uses (such as a printer). But the biggest advantage, in addition to these features, is that the PCjr's internal modem is also "smart." A smart modem is one that can carry out commands for us, such as dialing numbers or automatically answering the telephone. With a smart modem like our Junior's, it's easier to use our computer to communicate over the telephone.



Joysticks

To make playing games better, we can buy *joysticks* for our computer. A joystick, like the one in Figure 2-7, has a lever that we can move around to tell a game program how to move our player. There are also two buttons on each joystick, which we use as firing buttons or to reset a game. Our PCjr is designed to use two joysticks, so two players can enjoy a game at the same time.

Joysticks can be used with programs other than games. They can also be used with educational programs, drawing programs, and many others, as well.

CHOOSING A DISPLAY SCREEN

Our computer needs a *display screen* to show us what it is doing. The PCjr has three different sockets to connect it to a display screen, as we saw when we looked at the system unit. There are actually four main types of display screen that we can use with our PCjr, and even some more choices within each type. So here, we'll outline what the various display screens are like.

TV Sets

The simplest and cheapest option for a display is a TV set. You can connect either a color or a black-and-white TV to the PCjr's TV signal outlet. As one might expect, this option is the simplest and cheapest, but it also gives us the poorest results. A TV set's resolution—its ability to create a detailed picture—is relatively poor for computer use. For example, when a computer displays written text characters on a screen, it normally shows 80 characters across the width of the screen. Eighty characters will not show clearly on an ordinary TV set, so the PCjr, like the other IBM personal computers, is able to switch to showing only 40 characters on each line.

When we work with BASIC programs, we can set the display width with either of these statements:

```
WIDTH 40
WIDTH 80
```

Besides having a practical limitation of 40 characters across the screen, a TV set does not show most graphics games very well. Having a TV set connected to your PCjr is like having a bad antenna connected to your TV set—the picture quality will be poor; usable, but poor.

On the plus side, though, color TV sets do show all the colors that the PCjr is able to produce, and even black-and-white TV sets do a reasonable job of showing grey tones in place of different colors. I would say, though, that using your PCjr with a TV set as the display screen is very similar to not having the diskette and memory options: You can use the computer this way, but you won't get to realize its full potential.

When we use a TV set with a PCjr, we have to buy a special adapter called a *radio-frequency,* or *RF modulator.* The RF modulator converts the computer's signals into TV-style signals. The RF modulator plugs into the T-television outlet in the back of the system unit.

Some of the newest, component-style TV sets can also be used as a different kind of computer display. If you have this kind of TV, be sure to read the section on RGB monitors.

Composite Video Monitors

The next main kind of display screen for our computer is a *composite video monitor* (monitor is just another word for display). This kind of display plugs into the V-video jack in the back of the PCjr.

Composite monitors are the next step up from using a TV for a display screen; generally, composite monitors provide a much clearer picture than TV sets can.

There are two kinds of composite monitors: ones that show color and ones that don't. As it turns out, and as you'll see, this difference can be very important when we get down to the business of using our computer.

Color composites

Like a color TV, color composite monitors can show all the colors our PCjr can generate. The main difference is that a color composite monitor gives us a crisper, clearer picture, as it is designed for the special needs of computer pictures.

Monochrome composites

While it's very pleasant to see all the colors that our PCjr can produce, it turns out that color monitors aren't always the best to work with. The reason for this is very simple. Much of the work that people do with computers is with text—words and numbers. Unfortunately, when we're working with text, color displays for computers can be very hard on the eyes after a while. Very few color monitors show text clearly enough for us to avoid eyestrain.

Luckily, though, there is another kind of display screen that does show the words and numbers of text information very clearly. This is a monochrome, or black-and-white, composite monitor.

A monochrome composite display screen plugs into the same V-video socket that a color composite display does. But a monochrome display doesn't show any colors—everything comes out as shades of one color.

The advantage of a monochrome display is that the text it shows is usually much sharper than it would be on a color display. If you plan to use your computer mostly for writing, or for working with numbers, then this clarity may be a big advantage for you.

There are different colors of monochrome display. Many use green characters on a black background. Some glow with an amber color, which many people find easier on the eyes. A few others are straight black and white. If you want a monochrome display, you need to consider which of these colors you might like best.

RGB monitors

The highest-quality and most expensive kind of display screen for our computer is a Red-Green-Blue, or *RGB*, monitor. RGB monitors plug into the D-direct

drive socket on the back of the computer. The computer generates a separate signal for each of the three primary colors, red, green, and blue. This separation makes it possible for an RGB monitor to show a better picture than either a TV set or a color composite display, in which the color signals are mixed together. A good RGB monitor gives a much better picture than either a TV or a composite monitor. But, for working with written text, an RGB monitor is still not as clear as a monochrome monitor.

The new, component-style TV sets are usually set up so that they will take an RGB signal. So, if you have this type of TV set, you can connect it to your PCjr either as an ordinary TV (through the T socket), or as an RGB monitor (through the D socket). Either way works, but the RGB connection should work better.

SOME STRATEGIES

To close off this chapter, let's consider some reasonable strategies for buying and equipping an IBM PCjr. You'll have to rely on your own judgment and needs to determine which model of PCjr would suit you best, and which options you are most likely to use, but this section might give you a little insight and help you get just the right computer setup.

Compared to the other IBM personal computers, even a PCjr with every possible option installed is inexpensive. From that perspective, you might decide that the best PCjr for you is a PCjr with every option available. Taking this approach might waste a little money, but it will ensure that you have everything you want, right from the beginning. Countless computer owners have wasted many hours wishing they had bought some option that they later found they needed. One way to avoid that problem is to get everything. With the modest cost of the IBM PCjr, this approach isn't as foolish as it might seem.

If you don't intend to equip your Junior with every possible option, let's consider some approaches. The first question you need to answer is, what do you expect to use your computer for? Here are the main possibilities; you can see them outlined in Figure 2-8.

For Entertainment and Education

Start with: An entry-level PCjr, with a joystick or two. Buy the BASIC cartridge, and any games and educational programs you want. Use your TV as a display.

Then add: A color composite monitor for better picture quality.

Finish with: The diskette and memory options, along with DOS to enlarge your repertoire.

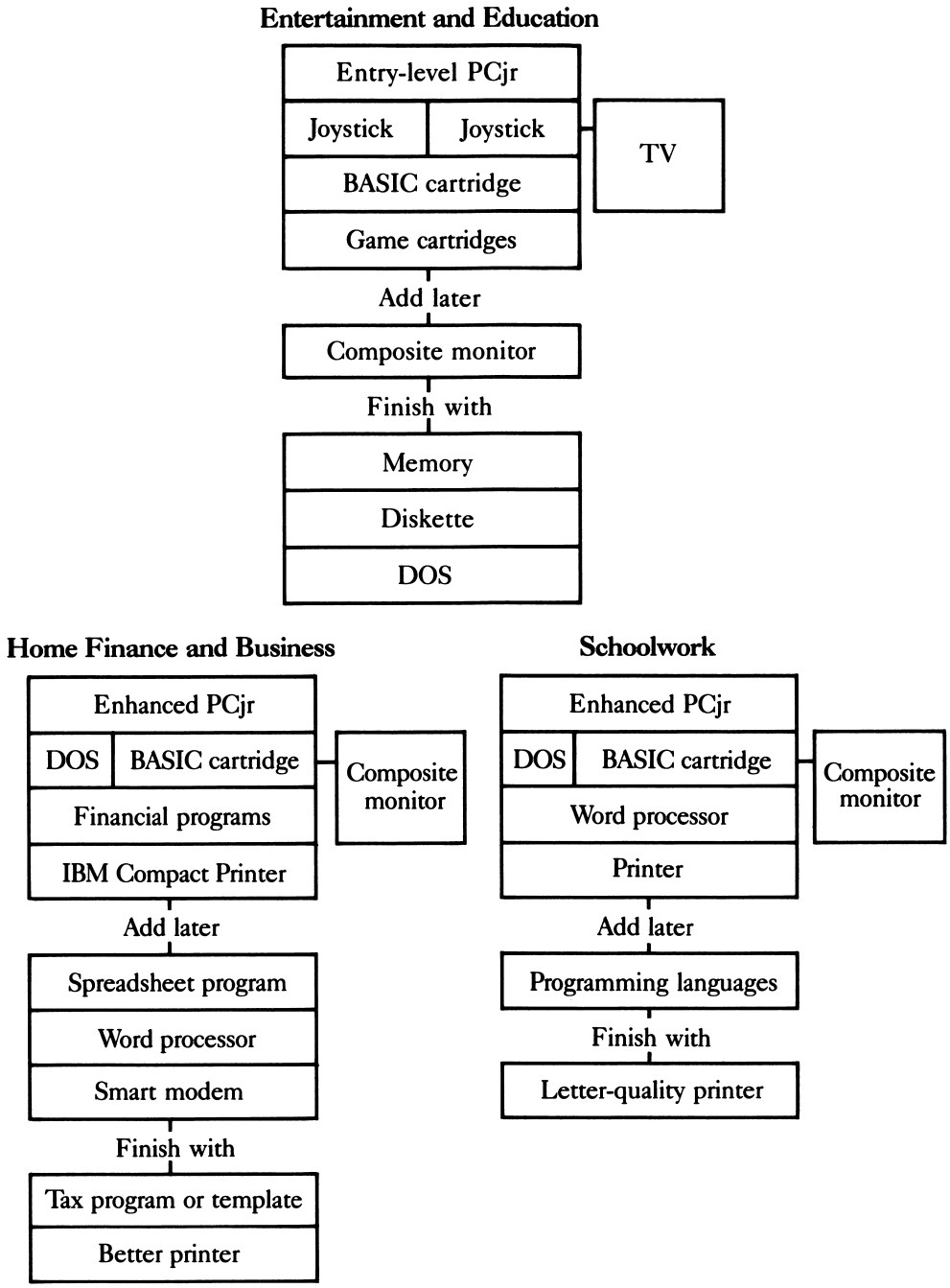


Figure 2-8. Equipment and software for the three main uses for a PCjr

For Home Financial and Business Use

Start with: An enhanced model PCjr. Buy DOS and the BASIC cartridge too, since many business and financial programs will need them; buy any financial programs you expect to need. Buy a monochrome composite monitor, or perhaps a color composite monitor for a display. Buy the IBM Compact Printer.

Then add: A spreadsheet program, such as Microsoft® Multiplan™ or VisiCalc™, and some sort of word processing program—it needn't be a sophisticated one unless you plan to do a lot of writing. Include the smart modem option if you plan to get financial information from Dow Jones or THE SOURCE℠.

Finish with: A tax-return program, or a tax "template" for your spreadsheet program; do this long before tax time, so you can use it for tax planning. If need be, get a better printer as well.

For Schoolwork

Start with: An enhanced PCjr. Buy DOS and the BASIC cartridge, and a simple word processor for reports and other homework. Get a color composite monitor, or a monochrome monitor if writing will be the main use. Buy an inexpensive printer of any kind (the IBM Compact Printer may not do, because it uses special paper).

Then add: Whatever programming languages are used in the grades ahead—probably Pascal.

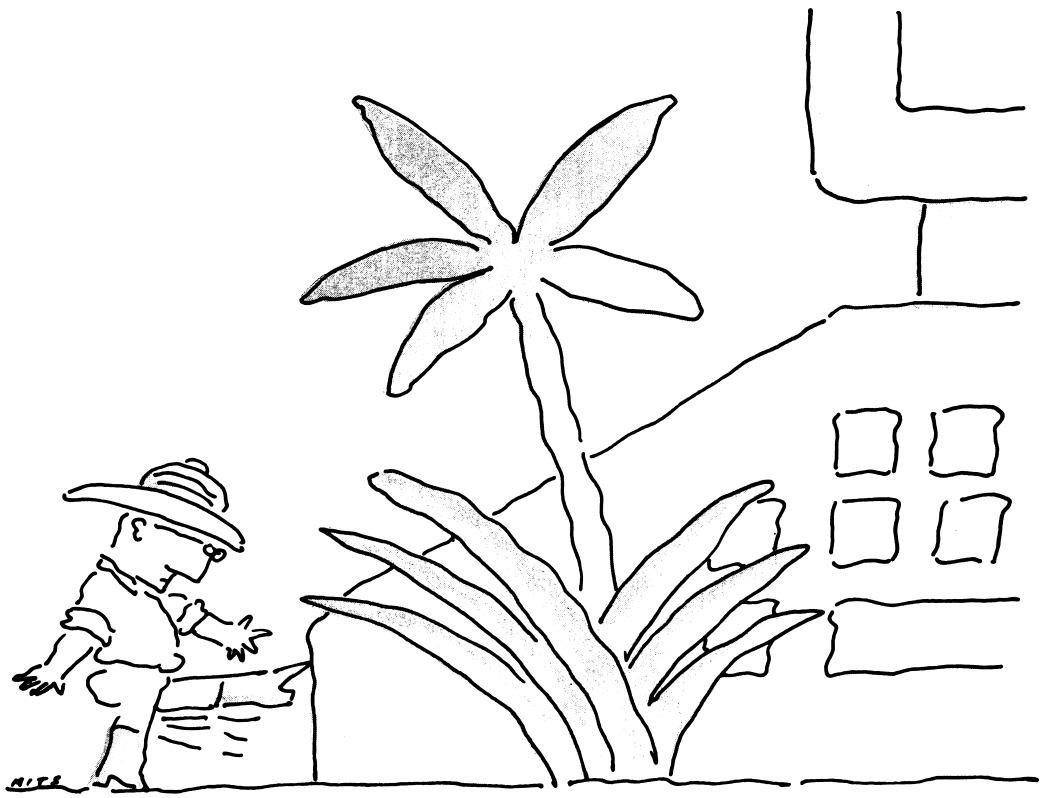
Finish with: A letter-quality printer, for good finished work.

What you actually need, want, and can afford may not match any of these outlines exactly, but the outlines can help you decide. Also, the chances are that you are interested in more than one of the uses we've outlined here. If that's the case, check all the advice and either choose from among the outlines or buy everything. When you are trying to decide what sort of display screen to get, consider that written text doesn't work well on color screens; if programming or writing are important to you, get both a monochrome monitor and some sort of color display, even if the latter is your home TV. If your different needs call for different kinds of printers, buy just one better-quality printer.

Whatever sort of PCjr you decide on, remember that you're never committed to one orientation or another. You can easily shift the focus of your computer by adding or replacing any of the options. One of the outstanding characteristics of the PCjr is its remarkable flexibility.

3
STARTING UP THE COMPUTER

It's now time to see how to start up the PCjr: You'll learn how to plug the pieces together and actually start the computer running. This chapter will give you a better understanding of the first stages of using your computer.



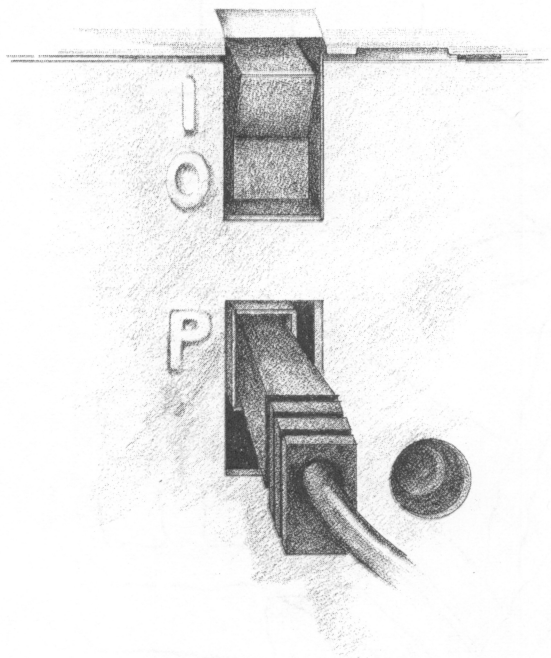
PUTTING IT ALL TOGETHER

It's remarkably easy to put the parts of a PCjr together so that they are ready to work. It's so easy, in fact, the chances are good that you'd do it right without any instructions or help at all. But when the time comes for you to get started with a PCjr for the first time, you can expect two kinds of help. First, your computer dealer should be able to show you exactly how it's done and might even do some of the steps for you. And second, IBM gives us a very clear step-by-step set of instructions in the *Guide to Operations* that comes with every PCjr.

With that kind of help, we don't need to discuss exactly how to put the PCjr together. When the time comes, you'll have all the help you will need. What we should do here, for those of you who don't have a PCjr yet, is give you an idea of just how easy it is to put the parts together.

Plugging in the Main Parts

There are four parts we have to get ready before we can use our Junior: the transformer, the system unit, the keyboard, and the display. The display is the most complicated, so let's talk about the other three first.



For the transformer, all we have to do is unwrap it, plug one end of the cord into the back of the computer's system unit, and plug the other end into an electrical outlet. Remember, from the last chapter, that the power cord from the transformer plugs into the socket labeled P, as shown in Figure 3-1.

Before we plug it in, though, we need to make sure that the computer is switched off. The on/off switch is above the P-power socket, and it's labeled 1/0, for on and off; make sure that the switch is pressed in on the bottom, where it's labeled 0 for off.

For the system unit, all we have to do is put it where we want it to be, make sure it's turned off, and plug in the power cord from the transformer. There are two things to consider when we are choosing a location for the system unit. First, we

Figure 3-1. How the transformer plugs into the PCjr

have to make sure that the ventilation slots aren't covered up, so that air can circulate inside to keep the system unit cool. Second, the front of the computer should be facing us, so that its electric eye can "see" the infrared signals coming from the keyboard.

For the keyboard, there are also two things that we need to do. First, we should make sure that its four AA-size batteries are in place. These batteries provide power for the circuits inside the keyboard and also for the infrared light that the keyboard sends to the system unit. So, turn the keyboard upside down and open the door to the battery compartment, make sure the batteries are in place, and close the compartment up again.

While we have the keyboard turned over, we can also see the two little folding legs that are under the keyboard. These legs are to prop the keyboard up at an angle; you can experiment, and see if you want to use them.

After we've checked the keyboard, we're ready to use it. Remember, wherever you put the keyboard, make sure that the computer can see the infrared light signals. Later, when you're playing with the computer, you'll discover that you can usually bounce the light signals off the walls and ceiling and the computer can still see them. But to get started, play it straight and make sure that the keyboard is aimed at the front of the system unit.

Remember, too, that you can connect the keyboard to the system unit with a cable. The cable plugs into the keyboard and into the K socket in the back of the system unit. If you want to, you can make this connection, and you won't need the batteries inside the keyboard.

Getting those three parts ready is remarkably easy. Getting the display screen ready is just as easy, but there are more ways to do it, because there are several types of displays.

The Display

Setting up the computer's display screen is more complicated to talk about, simply because the different kinds of displays use different connections on the back of the computer.

If you're using a TV set—a simple and easy way to get started with the PCjr—you need a PCjr TV connector, which includes an RF modulator, as we mentioned in Chapter 2. The TV connector plugs into the computer's T-television socket. On the connector is a switch that tunes the computer's signals to either channel 3 or 4; set the switch whichever way you want, since either channel will work just fine, and then attach the connector to the antenna leads on your TV set. You can see the PCjr's TV connection in Figure 3-2.

If you're using a composite monitor, it could be either a color composite or a monochrome composite, as we discussed in Chapter 2. Either kind plugs into the V-video jack in the back of the system unit. You shouldn't need an adapter cord;

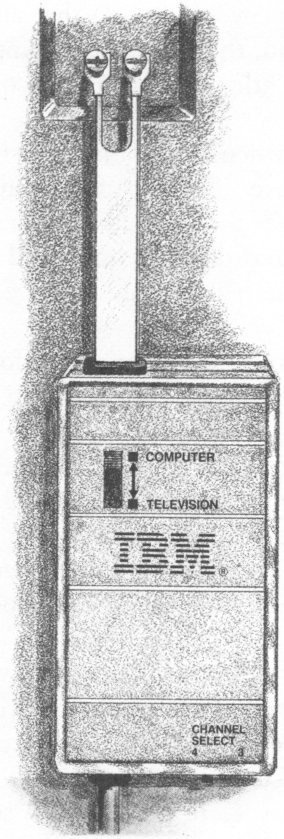


Figure 3-2. The TV connection

your composite monitor should have the right kind of cable on it already.

If you're using either an RGB monitor or a component TV as a display, you'll need to plug it into the D-direct drive socket on the back of the computer. You'll need a special cable to make this connection.

Whichever kind of display you use, setting it up is very easy. Just plug the display into the back of PCjr's system unit, connect the display to an electrical outlet, and you're ready to go.

One warning, though. Your Junior's system unit isn't designed to support the weight of a display screen. So wherever you put your computer, either set the display screen alongside the system unit or do as I've done with one of my PCjrs and get a stand to hold the screen above the system unit. As another alternative, you can put the system unit on top of your display screen, if it won't block the display's ventilation. I have my other PCjr sitting on top of a TV set, and it works just fine.

Getting Inside

Setting up a PCjr doesn't take any more work than we've just covered. But if you buy some of your Junior's options separately, you may have to open up the system unit to install them. Although your computer dealer should be able to take care of this for you, you might want to do it yourself anyway.

To get inside, where the memory, diskette drive, and modem options go, you pry off the top of the system unit with a screwdriver. There are slots in the back for this purpose; the *Guide to Operations* shows you exactly how to do it.

To get into the side, where the parallel printer adapter goes, you have to take off the side panel. Two plastic mounting pins hold it in place; again the *Guide to Operations* shows you exactly what to do.

Chances are, you'll never need to open up your computer; it probably came with all the options and equipment you need. But if you want to explore the inside, now you know how to do it. And carefully, of course.

START-UP BASICS

Once we've got all the parts of our computer connected, we can turn the power on and get it started. What happens then? Well, lots of different things, depending upon what we've put into our computer. Let's start with what happens when the computer is first turned on; then we'll go on to what we can tell our computer to do.

Power On

To make sure that everything is all right, our computer tests itself whenever we turn it on. This is called the *Power-On Self Test*, or POST. When the computer is starting and the tests are being performed, the PCjr puts a picture on the display screen that shows the IBM logo on the top, and a set of color bars below. These color bars demonstrate all the colors that our computer can generate. They're interesting to see, and they give us an opportunity to adjust the color settings on our display screen while the POST is being performed.

In the lower right-hand corner of the display screen, the computer shows what's happening while it's testing the memory. You'll see the display end up showing either 64KB or 128KB, depending on how much memory your computer has.

If anything goes wrong during the tests, the computer will tell you so with what's called an error message; you can look up the message in the *Guide to Operations* to see what it means. You shouldn't expect any errors—they rarely happen. But if you do get an error message, the *Guide to Operations* will tell you what to do about it.

Switching Gears

After we've been using our computer to do one thing, say play a game, we might want to switch to something else—to have the computer change direction. One of the ways we can do this is by turning the power off and then turning it back on again. But, if we do that, we'll have to wait while the POST tests are done.

To make it easier for us to switch from one thing to another, we have two quick ways to *reset* the computer. Resetting means that the computer drops whatever it is doing, clears its memory workbench, and starts over. Resetting is just like turning the power off and on, but the computer skips the POST and saves us some time.

We need to know how to make the computer reset itself, so we can do it when we want to and so we can avoid doing it by mistake. After all, if we're in the middle of using some interesting program, we don't want it all to go away by accident.

One of the two things that will reset the computer is plugging in or removing a software cartridge. Putting a cartridge in or taking a cartridge out of either cartridge

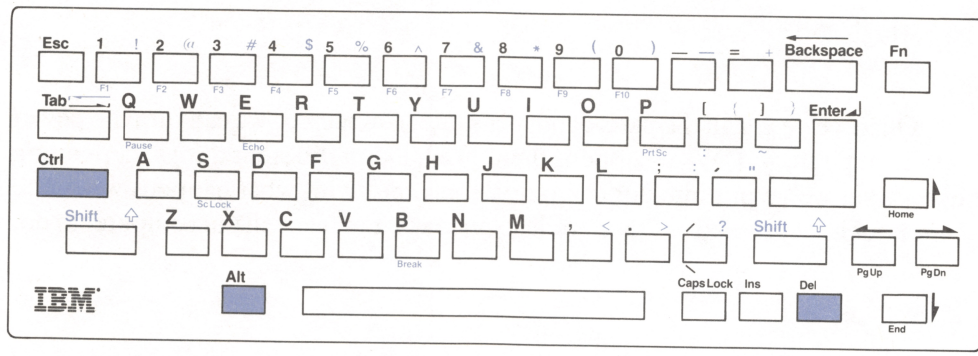


Figure 3-3. The reset keys on the keyboard

slot will automatically reset the computer. This is the easiest way to reset the computer accidentally. Sometimes, just bumping a cartridge that's not plugged in firmly will be enough to reset the computer. So be careful: When you plug a cartridge in, plug it in all the way.

The other way to reset the computer is with the keyboard. A special combination of keys is designed to reset the computer when the keys are all pressed at the same time. Since it takes three keys to reset the computer this way, this combination provides us with a safety feature that makes it harder to reset the PCjr accidentally. To reset the computer from the keyboard, we press and hold both the keys labeled Ctrl and Alt, and then we press the key labeled Del. This special key combination, illustrated in Figure 3-3, resets the computer immediately.

If you ever want to cancel one program so that your computer can go on to another one, you can use any of these methods to reset it: Turn the power off and on, pop a cartridge in or out, or press the Ctrl-Alt-Del key combination.

A Little Adventure

Once we know how to start up, one of the first things we can do with our PCjr is play a little game called Keyboard Adventure. Keyboard Adventure is a simple little game designed to show the basics of using the PCjr's keyboard.

To play Keyboard Adventure, we have to start our computer, or reset it, without any cartridges or diskettes in it. Then, the computer starts up with the BASIC program; we can tell it is doing so because this message appears on the screen:

```
The IBM Personal Computer Basic
```

If the first key we press is the Escape key (marked Esc), then Keyboard Adventure springs to life.

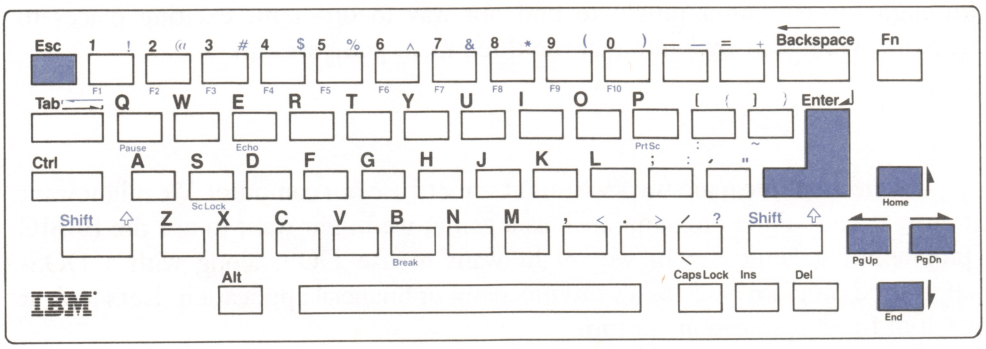


Figure 3-4. The Escape, Enter, and arrow keys

Keyboard Adventure begins with a little character, called P.C., appearing on the screen along with a box and an L-shaped block that looks just like the Enter key on the PCjr's keyboard. We play the game by using the arrow keys, on the right of the keyboard, to move P.C. over to the L-shaped block, and then to carry the block into the box. (You can see all these keys in Figure 3-4.)

Once we've gotten the block into the box, P.C. does a little dance and the game moves into a new phase, where we explore the entire keyboard. It's all explained, including some interesting tricks, in the *Guide to Operations*.

Deeper Testing

Keyboard Adventure isn't the only program built into our PCjr. There's also a special set of diagnostic programs that we can use to test our computer. There are two good reasons to use these diagnostic programs. One is that they test the computer even more completely than the POST does. The other reason is that the tests put our computer through its paces and give us a chance to learn more about how it works. For example, the display screen diagnostics show all the colors and characters that our computer can display, and that's fun to see.

To start the diagnostics, all we have to do is press a special key combination: Ctrl, Alt, and Ins (Insert). This combination is similar to the keys we use to reset the computer, but we press Ins instead of Del.

LOTS OF WAYS TO START

Once we have our computer going and we've played all we want with Keyboard Adventure and the diagnostics, we'll want to run some programs. There are many ways that we can start up programs. Each of them works a little differently, so let's take a look at how they go. There are plenty of paths we can take

through the computer jungle to find our way to different, exciting places to explore. We'll start by looking at the three main paths we can take.

Three Main Paths

There are three main ways we might want to use our computer. We might want to use a stand-alone program, such as a game; we might want to use the BASIC programming language; or we might want to use DOS, along with a DOS-supported program, such as a word processor or financial application. Let's look at each of these applications in turn.

Stand-alone programs

First, what if we want to use a program that is all ready to use, by itself? Most of the time this is what we'll be doing. We'll have some interesting program that we want to use, and we'll just put it into our computer and go.

Such a stand-alone program might come to us in a software cartridge, or it might be on a diskette. For cartridge programs, we can just plug the cartridge in and the program will automatically start. Remember that whenever we pop a cartridge in or out, our Junior resets itself; so to start a cartridge program, we just plug it into the computer.

Many of the popular game programs, such as Mouser and ScubaVenture, come as stand-alone cartridges.

Stand-alone diskette programs work in about the same way as these cartridges, but we have to reset the computer after we've put the diskette in place. We can do this, remember, with the Ctrl-Alt-Del key combination.

The BASIC language

Something else that we might want to do after we start the computer is use the BASIC programming language. With BASIC we can write our own programs, or we can just tinker with the computer and show off what it can do. Playing with BASIC can be a lot of fun, and programming in BASIC can be quite an adventure.

There are two ways we can start up BASIC on our PCjr. One is simply to start the computer without inserting any cartridge or diskette. When we do this, we get our Junior's built-in version of BASIC. This version is known as cassette BASIC (even though it doesn't come on a cassette). Cassette BASIC has all the essential features of BASIC, but it doesn't have the most advanced and interesting ones.

The better way to start BASIC is with the BASIC software cartridge. Once we have started up cartridge BASIC, it works just like cassette BASIC but offers us all the most advanced features that are available for the PCjr, such as special graphics features and complex musical sounds. Cartridge BASIC is the version that we should use if we want to do any serious programming.

To start up cartridge BASIC, we just plug the cartridge into either of the computer's cartridge slots, making sure that we don't have a diskette loaded, and BASIC will swing into action.

Once we have BASIC going, we can either enter programs or we can give BASIC commands, which BASIC and the PCjr will perform right away. We can play music, like this:

```
PLAY "ABCDEFG"
```

Or we can use BASIC to do calculations, like this:

```
PRINT "The square root of two is:", SQR(2)
```

And there's still more that we can do, besides. We'll see more BASIC magic later, and you can learn even more about BASIC with a book like IBM's *Hands-On BASIC for the IBM PCjr.*

DOS

The third of the three main paths we can take is to start up our computer with the DOS operating system, so that it's ready to run any of the many programs that work with DOS.

To start our computer with DOS, we just put a DOS diskette into the disk drive, and turn the computer on or reset it with the Ctrl-Alt-Del key combination. DOS loads itself into the computer's memory, and swings into action.

When we start DOS, it asks us for two things: the date and the time. This is so DOS can keep track of when things are done. Once we've typed in the date and time, DOS says hello with the message:

```
The IBM Personal Computer DOS
```

When that's done, DOS is ready to take our commands and run any programs that we want to use with it.

Carrying On

Once we've got our computer going, we're ready to explore all the fascinating twists and turns through the wonderful jungle of computing. And that's what we're going to do next. In the following chapters, we'll explore the ways we can use our computer for games, for education, for business, and more. We'll be discussing some examples of programs in each of these categories—but don't expect to see recommendations of the best of what's available; that's just not possible. So much new and exciting software appears every month that the magazines and buyer's guides are the places to look for that information.

On with the excitement of the PCjr! Let's go.

4
GAMES: FUN AND MORE

One of the things a computer can do best is educate and entertain us with programs we call games—but which really are much more than that.

MORE THAN JUST PLAY

Home computers have traditionally emphasized game programs, and the PCjr is no exception. Our PCjr can play games, teach us, and entertain us with the best of them, thanks to the strength of design that is embodied in all of the IBM personal computers.

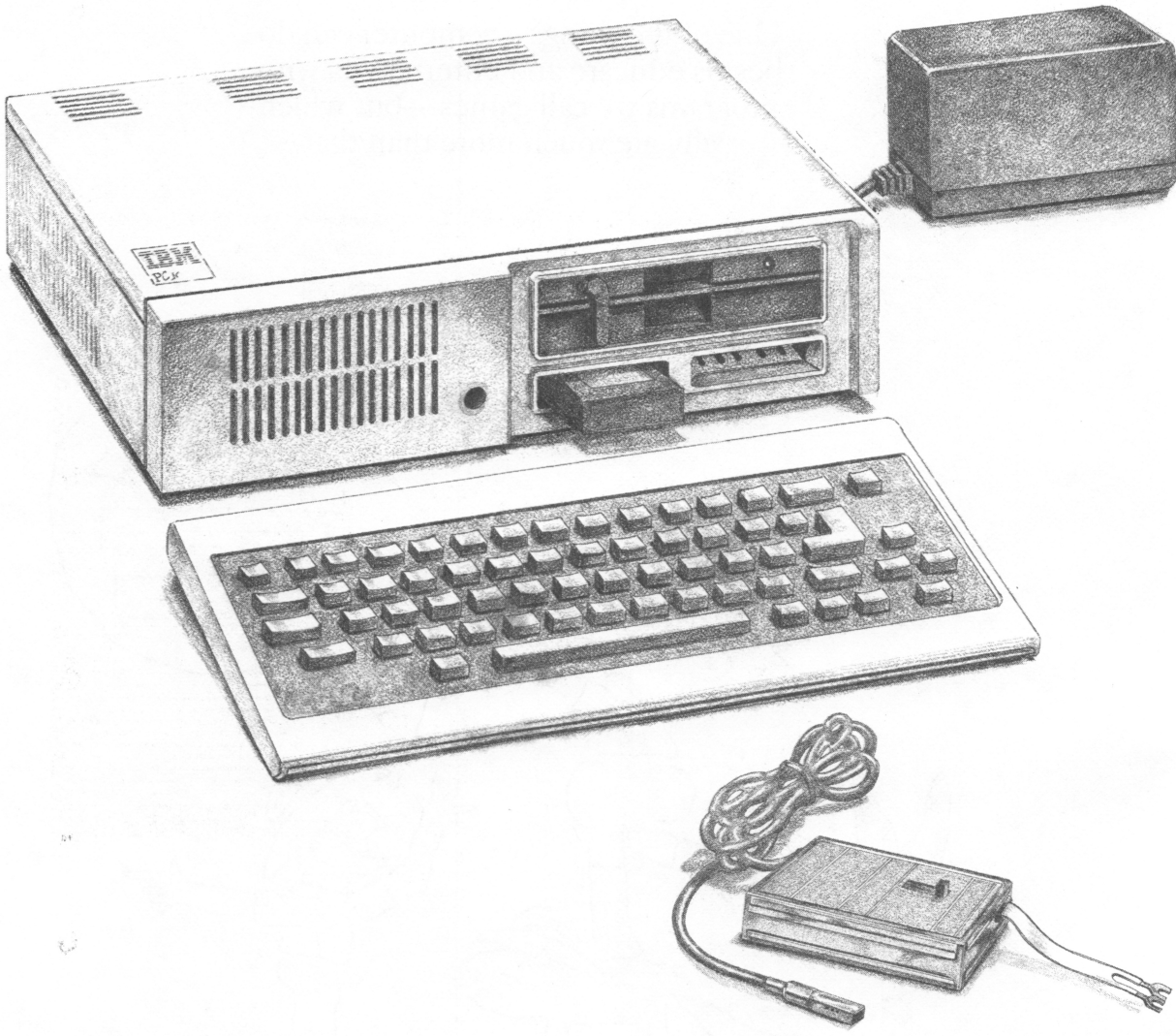
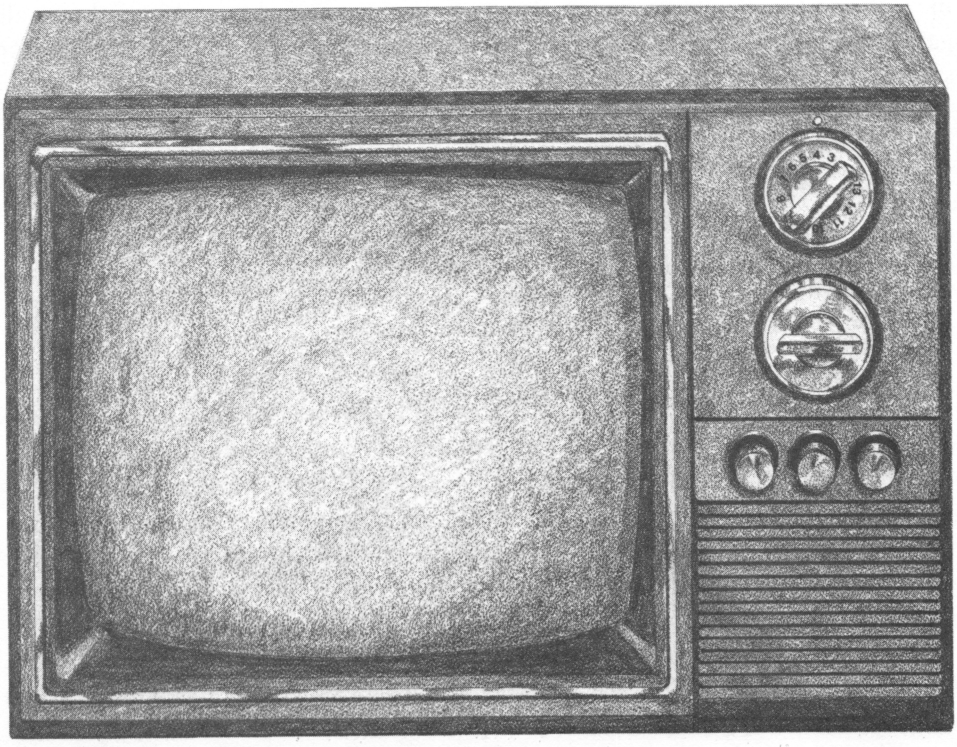


Figure 4-1. Game essentials

In this chapter we'll take a short look at games for the Junior. Games don't need much introduction, really, but there are a few things that we ought to go over, so you'll have a better understanding of how to get the most out of games on your PCjr. Let's start with the equipment you ought to have to get the most enjoyment from computer games. You can see the essentials in Figure 4-1.



EQUIPMENT FOR GAMES

When we're considering the equipment and options we need for games on a PCjr, we need to look at three things: the essentials that any game needs, the extras that can enhance a game, and the particular setups that different kinds of games need.

The Essentials

An entry-level PCjr—the most basic IBM computer equipment that we can get—and a color TV set are enough to enable us to play games. That means we can get started with games at the very lowest possible price for an IBM computer.

An entry-level PCjr, you'll recall, comes with cartridge slots and 64K of memory, and that's all we need to play many games on our Junior. Since almost all games make full, rich use of color, we'll want to have a color display screen, for sure. But a color TV set will do just fine for most games.

So our minimum essential equipment for games is:

- An entry-level PCjr.
- A TV adapter cable.
- A TV set.

The Extras

There are two extras, which you can see in Figure 4-2, that we might consider when we're equipping a PCjr for games: joysticks and a better display.

Some games just call for us to type information in on the keyboard, but arcade-style, action games call for movement. We can move a player around the screen with the keyboard, thanks to the four arrow keys in the lower right-hand corner. For the best, the fastest, and the most satisfying movement, though, we need to use a joystick.

A joystick is a relatively inexpensive addition to our computer. IBM's joystick, for example, costs only $40. When we add a joystick to our computer, we add a quick and more direct feel to the way we play action games. When we add a second joystick, two people can play a game at the same time. So, one or two joysticks should be the first option you consider adding to your computer if you want to use it for games.

The other extra to consider is the quality of the display screen. An ordinary TV set will work fine for most games, but the picture may not be as clear as it ought to be. For the best screen images, you should consider some alternatives. For example, if your TV set is one of the new component types, you ought to use the

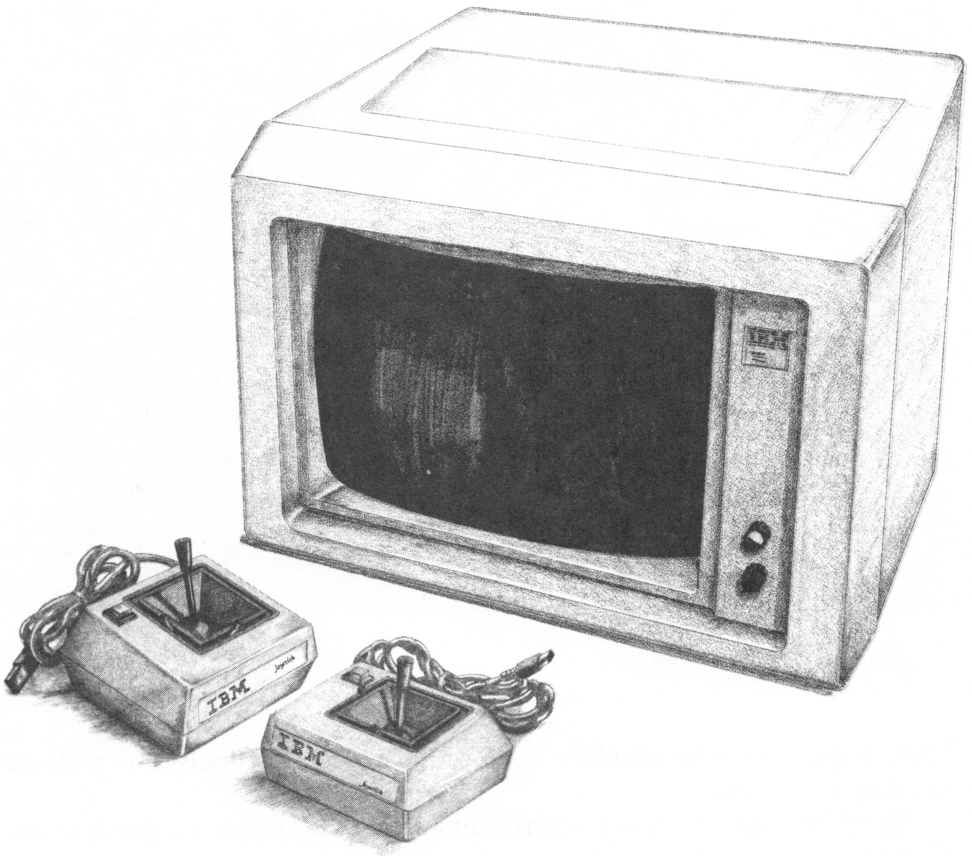


Figure 4-2. Game extras

RGB adapter cable with it, instead of an ordinary TV adapter cable. The RGB connector, by transmitting separate red, green, and blue color signals, will let your PCjr computer make full use of your TV's potential.

Otherwise, if you have a regular TV set, you should consider getting either a color composite display or an RGB monitor. A color composite display is likely to be cheaper and will probably give as good a picture as your games will need; an RGB monitor will give you the best picture possible.

The Programs and What They Need

There's one other thing we need to consider, though, in thinking about the kind of computer equipment that we need for games: the equipment that the games themselves require. Obviously, if a game comes on a diskette, we need a diskette drive to use it. But there are other combinations of game requirements.

Program Type							
1	Cartridge Program						
2	Cartridge Program	+	BASIC Cartridge				
3					Diskette Program		
4			BASIC Cartridge	+	Diskette Program		
5					Diskette Program	+	DOS Diskette
6			BASIC Cartridge	+	Diskette Program	+	DOS Diskette

Figure 4-3. Program types

These requirements have nothing to do with the games themselves, or with the way they are played.

To our computer, there are six different ways that any programs, games or not, can be stored and used. So from our computer's point of view, as far as equipment is concerned, there are exactly six kinds of programs. They are numbered and diagrammed in Figure 4-3; we'll look at each type and its requirements individually.

Type 1: Cartridges

First, there are games (and other programs) that come on stand-alone software cartridges. To use this kind of game, all we need is an entry-level PCjr to plug the cartridge into.

Type 2: Cartridges plus BASIC

Second, there are games that come on software cartridges, but need the *BASIC language cartridge*. These programs are written in BASIC, and to make the games work, we must have the BASIC cartridge. That, by the way, is why our PCjr has two slots for software cartridges. When we want to play this type of game, we just plug in the game cartridge and the BASIC cartridge, and away we go. It doesn't matter which cartridge is plugged into which slot; either way works fine.

Type 3: Diskettes

Third, there are games and other programs that come on stand-alone diskettes. To use this kind of game, we have to have an enhanced PCjr, with the diskette drive and memory options. When we want to play the game, we just slip the diskette into the diskette drive, turn the lever to lock the diskette in place, and start up our computer.

Type 4: Diskettes plus BASIC

Fourth, there are programs that come on diskettes, but also need the support of the BASIC cartridge. As with the Type 2 cartridges, these programs are written in the BASIC language and cannot work without the BASIC cartridge, even though they are otherwise self-contained enough to load themselves into memory and use our Junior without the aid of DOS. This type of program requires the enhanced PCjr and the BASIC cartridge, combined.

Type 5: Diskettes plus DOS

Fifth, there are games that come on diskettes, but need the DOS operating system as well. These games work like most of the more serious-minded programs for the IBM personal computers: They rely on DOS for essential help. To use this kind of game, we begin with an enhanced PCjr, start up our computer with a DOS diskette, switch to the game diskette, type in the name of the game, and away we go.

Type 6: Diskettes plus DOS, plus BASIC

Sixth, and finally, there are diskette programs that need the assistance of both DOS and the BASIC cartridge. Like other diskette programs, they require the enhanced PCjr.

A word of advice

You'll notice that there is a simple pattern to these six types of programs. The programs can come to us in a cartridge or written on a diskette, and the programs can stand alone, or they can need the assistance of BASIC, or DOS, or both.

By the way, you should know that most diskette programs that need the assistance of DOS are set up so that we can transfer DOS onto the program diskette. Programs like this come with instructions telling us in detail how to put DOS onto the diskette. Once we've transferred DOS onto a program diskette, then that diskette will act as a stand-alone diskette.

These six categories apply to all games, and to all other programs as well. When you get a new program, you should check to see what it requires, so that you know if you have the right things to make the program work.

SOME GAMES FOR THE PCJR

From the very beginning there have been some wonderful educational and entertainment games available for the PCjr and with each passing month, new ones appear. The library of games for the PCjr seems inexhaustible. Naturally, there is no way that we could attempt to catalog them all here, or even to name all of the most important and interesting ones. What we do want to do, though, is give you some idea of the types of games available, because doing so may give you ideas about the kinds of games you will want to seek out to get the maximum benefit from your PCjr.

Arcade-Style Games

When we think of games for the PCjr, the first thing that usually comes to mind is graphic, arcade-style games. These games can be lots of fun to play, and some people believe they even have a real benefit for growing children, because their fast action and constantly changing graphics help develop good hand-eye coordination and fast reflexes.

Mouser: A graphics game

Mouser is a fine example of what arcade-style graphics games are like.

When we play Mouser, we play the role of a farmer trying to rid a barn of mice. We begin with a bird's-eye view of the barn, where the mice hide in little rooms. We chase the mice around, trying to trap them and get them out of the barn. The female mice multiply, so we have their offspring to chase as well.

To make the game more interesting, the mice get harder and harder to trap. And then, darkness fills the barn, and we, as the farmer, are wandering around in the dark. If we can find the flashlight, we'll see a little circle of light that shows us part of what is in the barn; without the flashlight, we're completely in the dark.

With Mouser, we can chase the mice for hours if we want to. The game is so much fun that we can go on indefinitely.

Like many good graphics action games, Mouser can be played with the keyboard or with a joystick. When we're using the keyboard, we just press the arrow keys to move our farmer around. With the joystick, we move the lever in the direction that we want the farmer to move. A joystick has the advantage of making the moves direct and obvious; with the keyboard, we have to learn the arrow keys to be able to act quickly. The keyboard has one lovely advantage, though: With its infrared light signal, we can move the keyboard across the room and play from a sofa, in bed, or from one favorite chair, with the computer and a TV screen anywhere that we want to put them.

Text Games

There's another wonderful category of games that are very popular but aren't written about much: text games. When we think of computer games, we usually think of interesting, colorful pictures moving around on the display screen. But some of the most exciting images can only appear in our minds. Text games use words to describe a scene for us, and then they let our imaginations fill in the details in a way that is unique for each of us. In addition, since they don't concern themselves with pictures and graphics, they can provide us with some wonderfully complex puzzles to unravel.

Adventure: The first text game

The granddaddy of all text games is Adventure, a game that was created years ago on old-fashioned computers at the Massachusetts Institute of Technology and later adapted to personal computers. Adventure was the very first game for the IBM PC, the big brother of our PCjr, which can play Adventure as well.

In the game of Adventure, the computer gives us a fascinating and infuriatingly vague description of what is around us: We're in a building, near a stream and a forest, and not too far from the entrance to a cave. We type in verbal instructions to move around, find things, enter and explore the cave, do battle with enemies, find our way through mazes, pick up tools and treasures—and stumble around in the dark. Playing Adventure is very much like groping around in the dark, because we never know exactly what's around us, or what may happen next.

Adventure, like all really good games, can be played forever, because there is almost no end to the new corners that we can discover and explore. In many ways, playing the Adventure game is a lot like learning to use our computers: At first we're completely lost, not knowing how things go, trying what we can. Bit by bit, we learn the ins and outs and gain increasing mastery. It's maddening, and incredible fun.

Many text games are like Adventure: Essentially they're fabulous explorations. Other text games are like mystery stories: There is a puzzle, and we have to solve it; perhaps it's a murder mystery, perhaps something else. Whatever the text game is, even though it's no thrill to watch on the computer's display screen, what goes on inside our minds can be thrilling indeed.

Learning Games

Still other games are based on teaching youngsters word and arithmetic skills, and some even help develop reasoning ability, by leading them through the logic of a story. There are games for grown-ups as well, though, that can also be very valuable learning tools. For example, some games are based on the kinds of

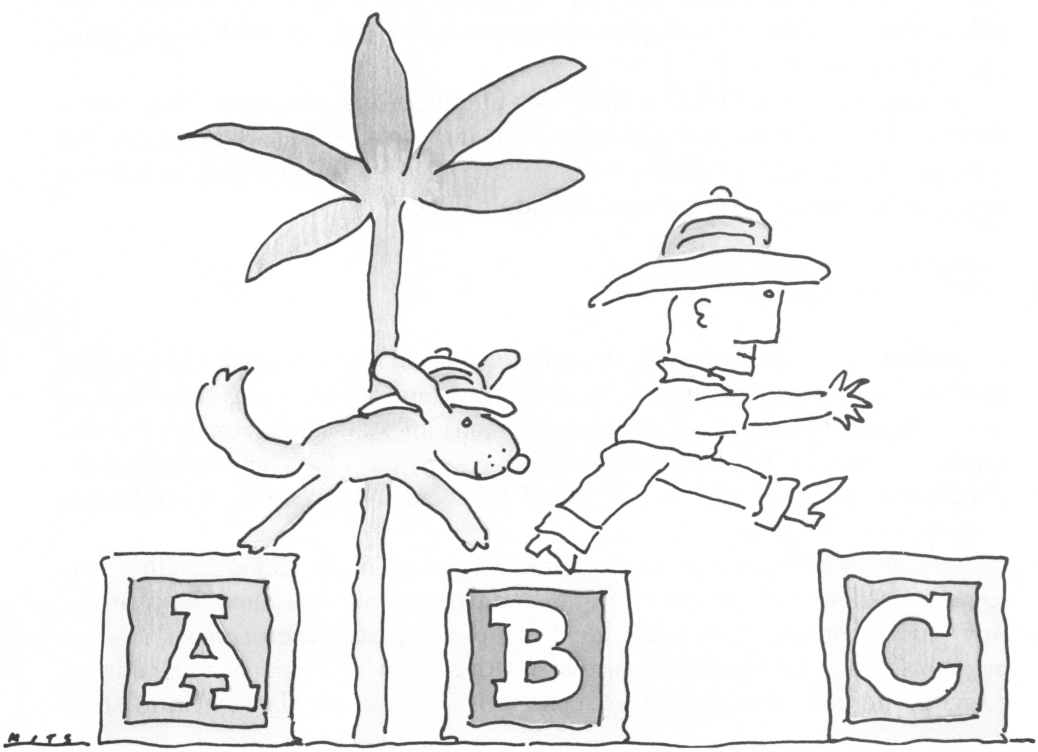
financial and strategic decisions that business executives have to make. In a modest way these game programs can help us learn management skills or learn to do better at personal and home finance. These sorts of games, for children and for adults, can be pleasurable and still teach us a lot.

There's lots of fun to be had playing games on the PCjr, and it's fun that can be very worthwhile, since we can learn a lot while we're playing. And that brings us to a closely related use for our computers: Education, the area we'll look at next.

5

EDUCATION: LEARNING IN ALL SORTS OF WAYS

While nothing can replace human teachers, the value of books, and the wonders of self-exploration, there are some aspects of education that computers help with marvelously.



THE PCJR'S TEACHING SKILLS

Why is it that one of the best things you can do with a home computer is use it for education? What makes a computer a good teacher? For one thing, a computer is infinitely patient. Our PCjr can repeat a drill over and over again, without ever growing tired or cranky—that can be a real advantage, since sometimes it seems as though one child could wear out the energy of a squadron of grown-ups.

Another advantage a computer has is that it can draw amazingly beautiful pictures to complement whatever it's teaching. That ability allows the computer to hold a child's attention in ways that are new and different, that enhance—not replace—the kind of teaching that people can do.

Also, a computer can easily do things that a human teacher might find harder: calculate a score while the teaching is going on; remember perfectly where mistakes were made; and keep track of which answers came quickly and easily, and which ones took a long time to get right. This sort of record-keeping is a part of teaching that is a snap for a computer to do.

Computers can be much more flexible in teaching than you might imagine. We tend to think that a computer would be very mechanical in how it goes about teaching. Most educational programs, though, use randomizing tricks to rearrange the order in which things are done. This sort of change keeps the activity fresh and interesting. Also, with clever programming, educational software can adjust itself to how well students are doing, repeating parts where they are weak and skipping parts where they do well.

In this chapter we'll take a short look at educational programs. Most educational programs are for young children, but that isn't all there is to educational software. We'll cover programs for kids, and follow up with a look at learning programs for high-school, college, and adult levels.

EQUIPMENT FOR EDUCATION

Before we go any further, we should take a look at how you ought to equip your computer for educational use. Educational programs aren't much different from game programs. In fact, that's the trick of a good educational program: It's a kind of game, but one that teaches. So you should equip your PCjr computer about the same for educational use as you would for recreation, keeping the following considerations in mind.

The main difference in the equipment needed for educational, rather than recreational, programs is that educational programs have less need for joysticks. Any kind of program can make good use of a joystick, even serious business programs. But the simple fact is that relatively few educational programs are designed to take advantage of joysticks. So, if you don't plan to get joysticks

otherwise, you can leave them off your list of educational needs.

There is one piece of equipment, though, that is needed when several PCjrs are used in the same educational setting: the *keyboard cable.* You'll recall that the keyboard on the PCjr normally talks to the computer's system unit with an infrared light signal. That works just fine when there is only one computer in the room, but when there's more than one Junior in the room, each computer needs to have its keyboard connected to the system unit by the keyboard cable, so that the signal from one computer doesn't interfere with the signal from another. For individual home use, there's no need for the keyboard cable—the infrared link is just dandy, and it lets us move around more freely. But in a classroom, or a computer workshop, where there are several PCjrs, the cables are essential.

The general rules about choosing a display apply to educational programs much as they do to other uses. You don't really need any special display screen, but you'll probably find that a color display is important for your children to get the most from educational programs. You can choose freely among the four possibilities of a regular color TV, a color composite monitor, an RGB monitor, or a component-style color TV.

When you are deciding on equipment, you should also keep in mind the six general categories of programs that we mentioned in the last chapter. You'll find that some educational programs need only their own disk or cartridge, some need the BASIC cartridge, and some need DOS or both BASIC and DOS. If you have particular programs in mind, you can choose the exact equipment they need; otherwise, if you get an enhanced PCjr, with the diskette drive and memory expansion included, and you also get the BASIC language cartridge and DOS, you'll be prepared for anything.

EDUCATION FOR KIDS

Most of the educational programs available are intended for small children, to teach them letters, numbers, and fundamental relationships, such as left and right, bigger and smaller. To get an idea of how these programs work, let's take a look at one of the programs that IBM introduced with the PCjr, called Bumble Games™.

Bumble Games: A Wonderful Start

Bumble Games is a set of programs for children from four to ten years old. It comes on a self-loading diskette and needs the assistance of the BASIC cartridge.

There are six different learning games in the Bumble Games set, challenging children at different levels of skill and understanding. The games are related, but each one is distinct, so there is a new interest to be found in each one.

To get an idea of what they are like, let's look at the second game in the set. It's

called Find the Bumble, and when I played it, I found it lots of fun.

In Find the Bumble, which you see in Figure 5-1, we're trying to uncover a friendly character called Bumble, who is hiding in a 4 by 4 grid. The rows and columns of the grid are labeled 0 through 3, and A through D. We start by guessing a row and column where the Bumble might be hiding. Naturally, most of the time we don't find the Bumble with a simple guess; here's where the learning comes in.

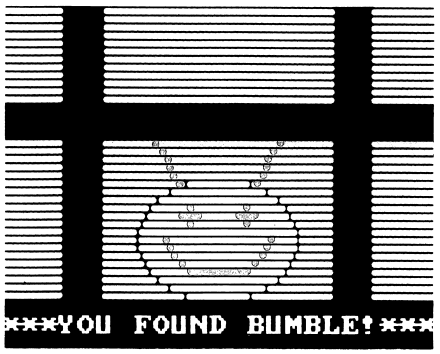


Figure 5-1. Find the Bumble

After we've guessed one row and column, the Find the Bumble game gives us a hint about which way to go next: It tells us the Bumble's hiding to the left or right, up or down from where we guessed. We have to understand what these directions mean, and guess again. When we don't understand the directions, we're gently guided; when we do, we quickly find the Bumble. Each time we play, the Bumble is hiding somewhere new, thanks to the "random action" that this game uses.

It's a very simple game, yet fun to play—and children enjoy giving the instructions.

That's the second level. The sixth, and highest, level of Bumble Games gives us a 10 by 10 grid, and we can play a game of connect the dots. We tell Bumble Games the coordinates of two dots, and the computer draws a line between them. In this version of Bumble Games, we get to create our own drawings, of kites, houses, whatever we want. While we're having fun making drawings, we're also learning about numbers, grids, and coordinates.

With six games, at six skill levels, children can find whatever part suits them best.

Some Other Educational Programs

There are many more educational games for our PCjr. Here's a quick summary of some of the other programs you can buy.

Bumble Plot™. A follow-on to Bumble Games, Bumble Plot is for slightly older children, eight to thirteen. Bumble Plot uses the same ideas, but takes them further. If kids loved Bumble Games, they'll love Bumble Plot, too.

Juggles' Butterfly™. This game is good for ages three through six. Juggles' Butterfly, which you can see in Figure 5-2, works on the simple relationships of above and below, left and right; it's good preparation, in fact, for Bumble Games, which works on relative relationships. Juggles' Butterfly also teaches some of the letters of the alphabet, P, D, B, and Q. Those letters were chosen because they are

the most challenging for kids. So Juggles' Butterfly also helps with the hardest part of the alphabet to learn—and that's where a learning program is needed the most.

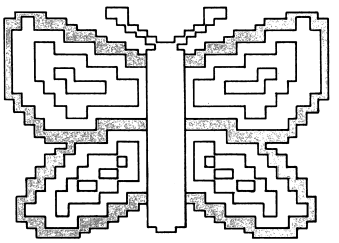


Figure 5-2. Juggles' Butterfly

Monster Math. This game is designed as an arithmetic drill. Since it's for older kids, there's less sugarcoating on it, yet it's still fun to play. In the game, we're given a series of arithmetic problems to solve in a 60-second time period. At the start of each play, a drawing of a monster appears, and each right answer makes a part of the monster disappear (you can see what happens in Figure 5-3). We can keep track of how well we're doing by looking at the monster. At the end, we get a numeric score, and we can begin another round. I had fun playing it, but my score wasn't anything to brag about.

This is just a sample of the educational programs that are available for the PCjr. There are lots more, and new ones appear all the time. You'll find plenty of learning programs to put your computer to good use.

HIGHER EDUCATION

Educational programs aren't just for little kids, although that's what we hear about most. There are some very useful ones for the rest of us, too.

Drills and More

These days we hear a lot about how important it is to learn computer-related skills and to become computer literate. Since computers are becoming easier and easier to use, the need for computer literacy might be exaggerated, but there's one skill related to computers that all of us could use: touch typing.

Figure 5-3. Monster Math

Every student has to write lots and lots of papers, and more and more adults use a keyboard one way or another in their work. This makes typing skills very important. So some of the most useful educational programs for our PCjr computers are typing programs.

There are several typing programs to choose from. IBM publishes one called Typing Tutor, which runs through all sorts of typing drills, checks how fast typing is done, and keeps an individual record of each person's progress.

In addition to typing programs, there are educational programs that teach all sorts of computer-related skills: How to use the computer itself, how to become familiar with the use of DOS, how to do elementary programming in BASIC, and much more.

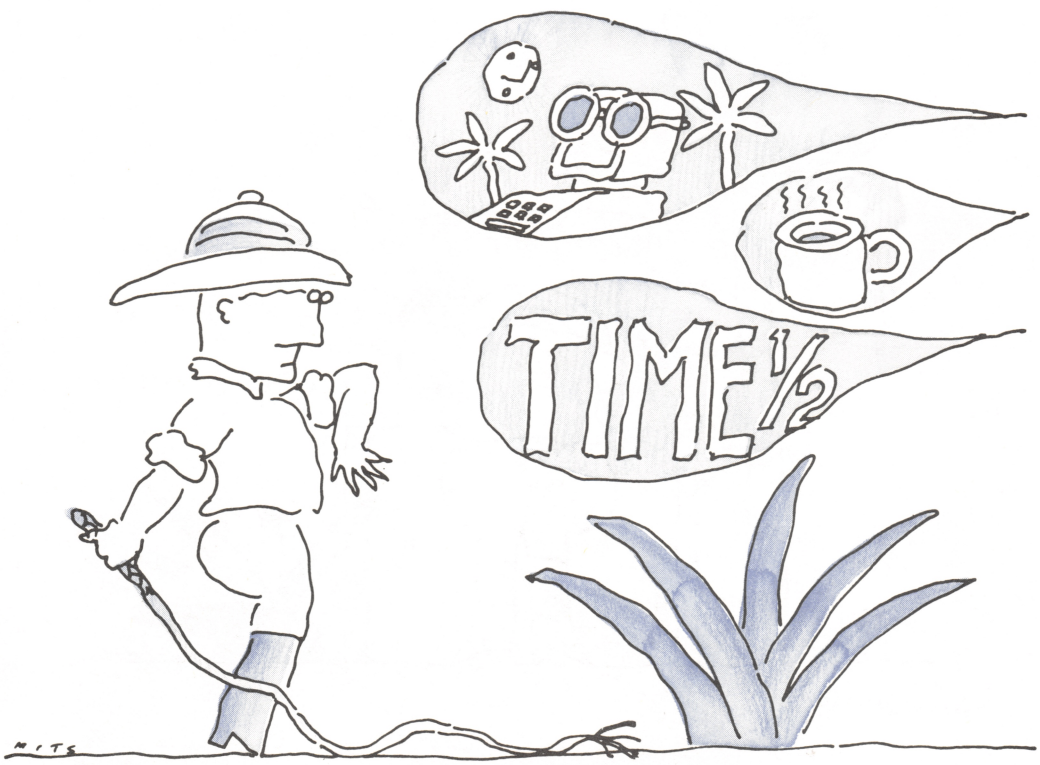
In short, there's no end to the educational programs that are available for use with our Junior. To find them, all you have to do is go to your computer store, look them up in any of several program buyer's guides, or check the many computer magazines. There's a lot to find.

Those Term Papers

We've been talking about the use of the PCjr for teaching, but computers are also used simply as a tool in education. This brings us to the PCjr as a word processor, a very smart typewriter. Plenty of students will be using the Junior to help with term papers—it's a superb writing tool. We'll be looking at using the PCjr for writing, together with other professional and business applications, in the next chapter.

6
SERIOUS BUSINESS: PUTTING JUNIOR TO WORK

As wonderful as entertainment and educational programs are, there is a lot of serious business that can be done with a home computer with the power and flexibility of the PCjr.



WHAT'S SERIOUS BUSINESS?

What is this serious business that we're talking about? It's all sorts of things. One of the most important is the use of the computer as a writing tool—and most of us do plenty of writing, either for school or for our work. Another is financial arithmetic and business record keeping—something every household could be doing with the help of a computer, and that every business must be doing. Yet another category includes the use of the computer as a professional work enhancer. We'll expand on each of these general categories and more in this chapter. The kinds of serious business we can do are important enough, and valuable enough, to justify all by themselves the expense of buying a PCjr. You'll find, as you explore the uses for your Junior, that the computer is likely to be a far more valuable tool than you had imagined possible.

And now, let's see what some of this serious business can be.

WRITING ON THE PCJR

History and legend tell us that computers are whizzes at arithmetic, but that they aren't always suited to other kinds of tasks. Yet we've seen that computers are

"Hard at work."

wonders at running action-packed, highly graphic video games, which aren't numeric at all. As it turns out, one of the things that computers can do best for the ordinary person who wants to use them is help enormously with the chore of writing.

Schoolchildren do lots of writing: essays, reports, and term projects, and almost everyone who works does plenty of writing as well. Even those who don't think of their work as involving writing compose plenty of job estimates, insurance forms, and expense reports. And then there are letters to friends and relatives, recipes to swap, holiday newsletters of here's-what's-happened-in-our-family, and much more. Let's face it—we all do plenty of writing, and we'd probably do more if it weren't such a chore. Writing with a computer can eliminate lots of the tedium of writing. But to understand how a computer can help, we first need to think about the steps involved in writing. After that, we'll talk about what programs you might want to get to help with these steps.

The Stages of Writing

Writing begins with planning what to write. That's a task for us alone; the computer can't help us there. But after the planning, there are several writing steps a computer can help with. We'll call these steps composing (for simplicity, we'll tuck revising and adapting in here), checking, formatting, and printing. You can see them outlined in Figure 6-1.

One of the things that sometimes makes ordinary writing such a chore is the fact that we want to avoid the tedium of redoing our work. Often, we try to go straight from idea to finished work, without any intermediate stages.

When we try to work this way, we frequently find that, while we should be thinking about what we want to write, we may be worrying more about how to arrange our work neatly on the page—a situation that has nothing to do with the task of composing our words. The wonder of writing with a computer is that the computer helps us divide and conquer. With the computer, we can more easily

Writing Step	How the Computer Can Help
Composing	Editor program
Checking	Spelling program
Formatting	Formatting program
Printing	Computer printer

Figure 6-1. Ways the computer can help us write

break down the job of writing into separate steps, so that we don't fret about one part, when we should be working on another part. Best of all, the computer will even do some of the steps completely for us.

Composing

The first step in writing is composing—putting the words together in the first place. The computer helps us with this step through a type of program known as an *editor*. An editor program is the tool that accepts what we pound out on the keyboard and keeps it all together. An editor program can be a program that we get all by itself, or it can be part of a complete word processing package. But, whether we use a separate editor program or a word processor, it's the editor part that accepts the words we type in.

Revising and adapting

There is a saying that writers don't write, they rewrite. When we do our writing with a computer and an editor program, instead of with a typewriter or a pencil, we can very easily sketch out our ideas, then elaborate on them and then change parts here and there, rewriting, rearranging, and rephrasing as much as we want. This flexibility is one of the biggest benefits in using a computer. As we compose what we are writing, we can make changes with a minimum of effort—the computer does the work of putting our changes together so that we can see the revised work immediately on the screen.

There is another way that computers help us compose our writing, and that is by letting us borrow from what we've done before. Some kinds of writing are very repetitious; expense reports, for example, all follow the same outline, and holiday letters to different friends may have many paragraphs in common. When we compose with a computer, we can easily incorporate what we've written before into what we are writing now. If we have a standard outline (like an expense report) that we're following, the computer can copy and save the outline for us to use whenever we wish, and with a minimum of effort. Also, if we're borrowing part of what we've already written, the computer makes that easy, too.

Checking

After we compose what we want to write, the computer can check our work in two ways. The first, and most common, is checking our spelling. *Spelling checkers* can quickly and easily find words we've misspelled or mistyped. Of course, since they're quite mechanical about it, spelling checkers won't tell us when we used the wrong word—such as "their," when we meant "they're."

Another, more sophisticated way that the computer can check our writing is just beginning to be used. This checking is done with programs that can check grammar and punctuation, or even advise us about good and bad usage.

Formatting

The next step in writing that the computer can help with is formatting—arranging how the writing will appear on the page. Formatting programs do the work of laying out what we have written; they follow our instructions to provide us with correct margins, proper paragraph layout, headings and footnotes when they are needed, and so forth. Formatting programs can save us lots and lots of work, because they sort out the details of what will fit where.

Printing

The last part of the tedium of writing is getting the final, printed result. If we're using a typewriter or a pen, this job is especially laborious after we've composed and corrected what we're writing. But with a computer, the job of printing is completely mechanical. With just a few seconds' effort on our part, we can tell the computer to print what might take us hours to type. And if we want more than one printed copy, we just tell the computer to print more, and it's done—with no typing errors.

Now that we have some idea of how the computer can help us with writing, let's look at the hardware equipment and the software programs we'll need for writing. Choosing the right hardware can be easy; choosing the software might be more difficult.

Equipment for Writing

To write on a PCjr, we'll need the enhanced model, with its extra memory and diskette drive. We'll need the diskette drive so that we can use diskettes to save what we've written. We'll also need to choose the right kind of display screen, and a printer.

As it turns out, the kind of display screen that works best for writing, or any other professional-style use of the computer, isn't what works best for games and education. The main reason is simple: When we're working with a screen full of numbers and letters, we need the crispest, clearest image possible. Colors may not help at all—in fact, they may make it harder on our eyes.

So, while good-quality color displays, and even a TV set, can be used for writing, if you expect to do lots of work with words and numbers, I highly recommend using a composite monochrome display. It's very likely that you'll be much better off with a composite monochrome, rather than even the best and most expensive color display. It's also good news that composite monochrome displays are just about the cheapest that we can buy, so adding one to your computer will be a modest expense. Use a composite monochrome for work, and borrow your color TV for play.

Writing on a computer calls for a printer as well. Here, there's no simple standard advice that I can give you, because the printer you buy will depend on the writing you do. You may absolutely need the best-quality printed result that you can get, or a rough printed image may suit you just fine.

The IBM Compact Printer represents one printing extreme. It's very low priced, about $175, which is nice. It does, however, create print images with heat, so it uses specially coated, thermal paper, which some people don't like; also, its dot-matrix print quality is among the plainest.

At the other extreme there are letter-quality printers, such as the NEC 3550 Spinwriter™, which costs more than the PCjr, but produces very nice, professional results. In between these two extremes, there are lots of other printers to choose from. Especially attractive to many people are some of the newer dot-matrix printers, which combine the low price of traditional dot-matrix printers with a better, nearly letter-quality print image.

Whatever you decide on for a printer, here is something to consider. Many active families may decide to get more than one PCjr, because there's just too much need for one computer to satisfy. After all, there may be several reports that have to be written the same evening. While each Junior will need to have its own basic equipment, you can probably share one high-quality printer among several computers. That sharing can save you a good deal of money right there.

Software for Writing

There's a lot to choose from when we set out to select software tools for writing.

The main choice to be made is whether to get a complete word processing program, or to get a separate editor program.

A complete *word processor* allows us to center headings, set tabs and margins, and generally control all such niceties of appearance; this ability can be a major convenience. However, many word processors don't like to share their work with other programs. They save their work in a form that is useful for themselves, but which cannot be read by other programs. This can be a problem if we want to use word-processed material with other programs, as well as for writing letters and reports. For example, if a word processor doesn't share its work well, we may not be able to use some spelling checkers with it.

Despite this disadvantage, though, the chances are that you'll want to get a word processor. To give you an idea of what's available, let's look at two of them, HomeWord™, and cartridge VolksWriter™.

HomeWord was specially written for use with home computers like the PCjr, and so it's tailored to the needs of the beginning home user, not to full-time writers. Among the advantages of HomeWord are that its command are very easy to understand and use, and it takes almost no time to become proficient in using it. Also, HomeWord has an unusual feature that is very attractive: As we're writing, it

shows us a small drawing of what the finished page will look like, so we get a good feel for the format of our end result. There are some disadvantages to HomeWord, though. Its features are simple, and many people may find that it can't do things they want it to do. Also, HomeWord, at times, doesn't keep up with even a moderate typing speed—and it can be disconcerting to have the display screen lag behind what we've typed in.

Cartridge VolksWriter is an example of a word processor that is oriented toward more serious users. Professional word processing programs, like VolksWriter, provide more features and more power than programs like HomeWord, but they also take longer to learn to use effectively. The cartridge version of VolksWriter has a special advantage, in that we don't have to switch between a program diskette and our data diskettes to use it. (Most programs, including HomeWord, make us switch diskettes in and out of our PCjr's diskette drive, depending on whether we are using the program or using our information.)

There are many other word processing programs to choose from. These two just give us an idea of the range of choices that we have.

There are also lots of spelling checking programs that we can use. Since I write a lot, and since I'm a terrible speller, I have plenty of experience with them. My favorite, of all the ones I've tried, is IBM's Word Proof™. It's very efficient, fun to use (a rarity in a serious program), and does an excellent job of suggesting what word you meant when it finds a misspelled word.

Whatever specific programs you choose, you'll find that you need a word processing program (possibly, if you don't write a lot, just an editing program) and a spelling checker. These two programs will give you everything you need to use the PCjr as a marvelous electronic pencil.

Writing with a computer—word processing—is probably the most universal application that people put on their computers. It will probably be one of the first non-entertainment uses you try with your PC Junior. How easy or difficult it is to learn will depend on which programs we use and how quickly we can understand the thinking behind them. Sometimes getting started with a word processor is a snap, and sometimes it's frustrating. Fortunately, though, word processing is one of the easiest things to get going on, with our Junior.

FINANCE

The next kind of serious business to consider using a PCjr for is business and finance. What are these programs? What can they do? How can they help us—at home, and in our work? There is a wide spectrum of business and financial programs, including home budgeting, business accounting, general financial planning, and income taxes. Let's take a brief look at the equipment we need, and then we'll look at each of these categories.

Equipment for Finance

The computer equipment that we'll need is very much like what we need for writing. For sure, we'll need an enhanced model of PCjr, with its diskette drive and extra memory. If we'll be doing a lot of work with numbers, then a monochrome display is likely to be a better choice than a color display. Finally, we'll need a printer; but here, an inexpensive one, with a plain print quality, is likely to do just fine.

Home Budgeting

Home budgeting programs are designed to help us manage our budget and the simpler parts of our financial planning. There are quite a few home budgeting programs to choose from. Each has different specific features, but they all have some broad characteristics in common. They will, for instance, help us set up a target budget for the different elements of our expenses—such as clothing, transportation, and entertainment—and then they'll let us see where we are spending more or less than we think we should. Home budgeting programs will also help us keep track of categories of expenses that are tax deductible, such as medical costs and sales tax. And, they can help us answer such questions as: "If I buy this new car, will I have enough money next year for a vacation?" A good home budgeting program can help us judge whether we are setting aside enough money for emergencies or whether we are living too close to the edge.

Be forewarned, though, that home budgeting programs can't perform miracles, and some of them don't necessarily tell us anything we can't figure out more easily by hand. Among the most notorious examples of this sort of application are checkbook balancing programs; many people find that it's more work to use them than to do without them. The only real advantage to using a home budgeting program comes when it does something that we can't do, or can't easily do, without the computer.

Among the many budgeting programs is IBM's Home Budget Jr™, a diskette-based program that has been specially tailored to the abilities of the PCjr. Home Budget Jr keeps track of your budget—or money plan, as the program calls it—and monitors your income and expenses to see how you're doing.

Business Accounting

Accounting programs represent another type of serious business use for your home computer. If you have a small business, or you are thinking of starting one, a business accounting program can help you keep your records in good shape.

The general bookkeeping needs of a small business can be taken care of by what is known as a general ledger program. A general ledger will accumulate

day-to-day accounting transactions and produce the main reports needed by a business, such as an income and expense statement, a balance sheet, a statement of changes in financial position, and all sorts of posting reports, budget comparisons, and other financial analysis reports.

I keep track of my own software business with a general ledger program, and I find it very practical. After getting past the initial effort of learning how to use this program and learning to understand its quirks, I find it now takes me only about three hours a month to do all my bookkeeping work. A professional bookkeeper would probably take up more of my time than this tidy software package does, and cost me a lot more, to boot. For me, my general ledger accounting program is a classic example of how a computer can help in business.

There are more elements to business accounting, though, than just general ledger bookkeeping. If you are in business, in a big way or small, you are probably aware of them all. There is payroll accounting, accounts receivable, accounts payable, and inventory control. These, together with a general ledger, are the standard elements of computerized business accounting. Complete business accounting programs usually provide all these elements, in varying degrees of complexity, either as a combined package, or as separate parts that you can mix and match.

If you have a business to run and want computerized accounting help, you can set up the parts that you need. The accounting needs of my own business can give you a practical example of what you might do: I use a general ledger, and I find it very helpful. I use an accounts receivable program, and find it worth the effort required to use it, but just barely. I could use a payroll program, but I've decided that it would be more trouble than it's worth. And, in my kind of software business, there is no need at all for accounts payable or inventory control.

No matter what business you're in, you can find the mixture of accounting features that suits you best. And with the right advice, some effort, and a little grief, your business accounting can be smoothly computerized. Some warning is due here, though: In my experience, accounting programs rarely come with the sort of advice and guidance that a beginner needs to set them up properly. I was able to computerize my own accounts—with no accounting training—only because I had already worked with professional accountants in computerizing two other businesses, so I knew some of the ropes. Unless you have a strong accounting background and are good at solving puzzles, you should probably get professional help in setting up your own business accounting. Once your accounts are set up, though, the benefits can be plentiful.

Finally, one more note of caution is in order. Our PCjr is a dandy little computer, but there are some computing jobs that involve just too much work for it to handle. Business accounting is complex, and many of the business accounting packages available were designed for the greater capabilities of our Junior's bigger brothers. And even accounting programs that can be used on the PCjr may not be

able to hold and manage all the information that you need in a practical way. So, before you put your business records on the line with a computerized accounting system, be sure that you have enough computer to handle the load.

The Famous Spreadsheets

One of the most famous and versatile financial tools for small computers is a type of program known as an *electronic spreadsheet*. There are quite a few spreadsheet programs available, and the chances are good that you are familiar with the names of several of them. The original spreadsheet program, the granddaddy of them all, is VisiCalc. One of the most advanced and highly respected spreadsheets is Multiplan.

Even though spreadsheets are very sophisticated, most of them work just fine on our PCjr. For example, both Multiplan and VisiCalc, two of the most popular, run on our Junior.

An electronic spreadsheet is a very special kind of program that acts as a sort of structured calculating machine for us. A spreadsheet begins with the idea of a two-dimensional grid consisting of rows and columns of little open spaces called cells. We can put numbers into these cells and then we can define formulas that perform calculations on the numbers. The simple example shown in Figure 6-2 illustrates the idea and general form of a spreadsheet.

We can set up a grid like this, and then develop formulas that, perhaps:

- Add the months across the grid to calculate the year's totals; for example, row 3, column 2 + 3 + 4 . . . . = row 3, column 13
- Subtract the expenses from the income to calculate the savings: for example, column 2, row 3 −column 2, row 4 = column 2, row 6.

A spreadsheet program will let us build worksheets in this way, and it will then do the calculations for us. As we put in numbers for each passing month, the spreadsheet will recalculate new totals automatically. Our example may be very

	Column 1	Column 2	Column 3	Column 4	Column 13
ROW 1		January	February	March	Year Total
ROW 2					
ROW 3	My income	450	500	0	950
ROW 4	My expenses	300	550	0	850
ROW 5					
ROW 6	My savings	150	−50	0	100

Figure 6-2. A spreadsheet outline

simple, but it gives an idea of what can be done. Even complicated sets of numbers can be organized in a spreadsheet, and the calculations can become quite sophisticated.

All sorts of things can be set up in a spreadsheet format. For example, as we've seen, it's relatively easy to set up a grid that roughly follows income tax calculations. Each row of the grid could represent one major entry on the tax return, and the columns could represent years (rather than the months that our example shows) to give us a picture of our changing situation from year to year. It's even possible to incorporate the varying tax tables in a spreadsheet.

The most popular use for spreadsheets is *financial planning*, and that's how we're most likely to use them. Here's an idea of how it is done. We set up a basic grid of cells, called a model, that outlines our home budget, perhaps, or what we think might happen with a business we want to try. Then, once we've set up the model, we can play with different numbers and see what would happen as a result of this or that assumption. For example, we could try a formula that calculates expenses growing at the rate of ten percent a month, or income that goes up and down seasonally. We can test the effects of a rise or fall in interest rates. As long as we have an idea of something meaningful that might happen, and as long as we can use numbers to measure the situation, the spreadsheet can calculate what the result would be.

The reason a spreadsheet program is so useful for financial planning is because it enables us to plan and to analyze alternatives quickly and easily. And often, our calculations turn up surprises. A financial model might show us that the success of a venture is much more sensitive to changes in inflation than we thought, or much less. It might confirm that what we thought all along was right, or that we were far off target. The power and flexibility of a spreadsheet makes it very practical to test possibilities that would not be practical to do by hand or with an ordinary computer program. At times, this ease of planning can be a real factor in how well we make business decisions.

Because of the complex kinds of situations they're designed to handle, spreadsheets aren't the easiest computer programs to use. Some programs are fairly easy to understand, with just a little tinkering and trial and error; HomeWord, for example, is like this. Usually, though, spreadsheet programs seem quite mysterious and confusing until we've mastered their basics. Fortunately, they have gotten better in recent years—easier to understand and easier to use. Still, if you want to try using a spreadsheet, you should expect to undergo an initiation; and reading up on the subject would help, too.

Ah, Taxes

Income tax programs can help with some of the nuisance of filing our taxes, and they can make tax planning easier. Tax programs fit into that in-between area of

things that we can do ourselves without a computer, but which we might do better or more easily with some automated help. Generally, tax programs can be useful in three ways: First, in accumulating tax records over the year, to keep running totals of the various tax items. Second, in projecting our taxes, to help us estimate whether we will have a refund or a bill at the end of the year. And third, in preparing the actual figures for the tax return.

There are two different forms of tax programs. One is a specialized tax package that contains all the programming necessary to help with income taxes. I use one of these myself, and I've found it helps me both in tax planning and in the final preparation of my tax return. It doesn't take all the pain out of doing my taxes, but it is useful enough to justify both its cost and the effort that is needed to use it.

The other form of tax program works as a supplement to the generalized spreadsheets, such as Multiplan and VisiCalc. Spreadsheets can perform all sorts of calculations on a rectangular grid of numbers. By using a spreadsheet with what is called a template program, we can tell it to organize its calculations for some particular kind of work. An income tax template can thus act as a crude form of tax program. The advantage of a tax template is that it borrows the skills of the spreadsheet, and so it can be much less expensive than a complete, specialized tax program. But the disadvantage of a tax template is that it can only work in the format of a spreadsheet—that might be OK, but it's not ideal.

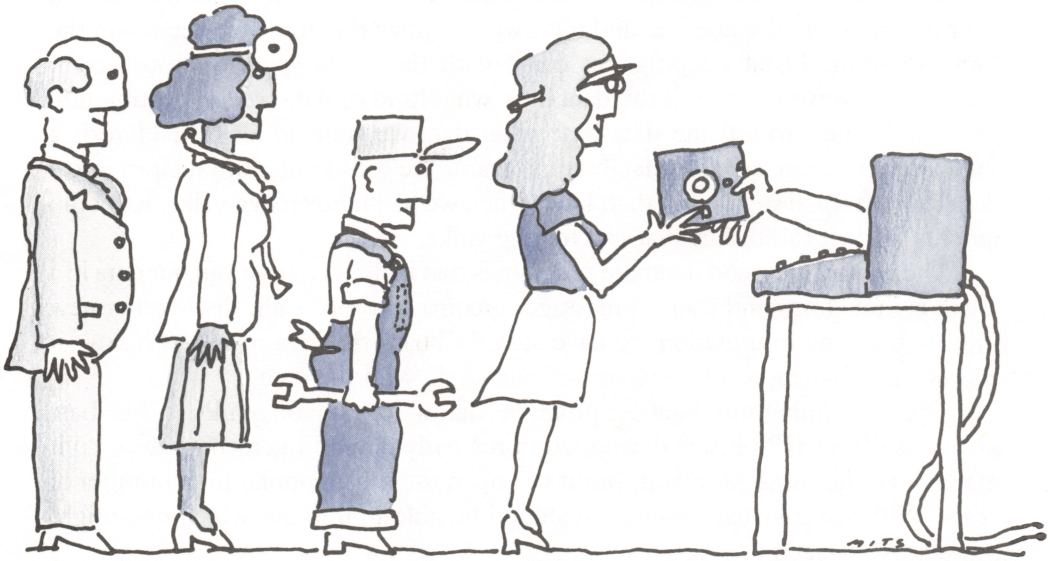
OTHER TOOLS

We've talked about some general-purpose working tools for our computer: word processors, spreadsheets, and general accounting programs. There are also all sorts of programs to suit the specific needs of various businesses and professions.

There isn't a whole lot that we can say about these programs, except that, if you have a need, the chances are good that someone has already written a program to fill that need or a need very similar to it.

Engineers, for example, will find a host of programs to do stress analyses, cost estimating, earthmoving calculations, and so forth. Real estate agents will find programs for brokers' statements, property record keeping, and tenant information. Insurance agents will find programs for client record keeping, prospect workups, application-form typing, and more. Medical and dental offices will find specialized client-billing systems, automated Medicare insurance filing, and supplies inventory and ordering.

Bankers and financial advisers will find software for portfolio analysis and individual retirement account (IRA) calculations. Tax preparers will find professional income tax programs, including corporate and partnership tax returns, that go far beyond the abilities of tax programs designed for the individual. Churches will find specialized membership and contribution programs. Heating-oil services



"Something for everyone."

will find programs that use local temperatures to calculate schedules for service.

And so it goes. The list of programs that are custom written for various professions, types of businesses, and organizations seems endless. If you have a need, it is almost certain there is a program close to your need that can be used on the IBM PCjr. We have only hinted at the full range of professional tools that are available. To find the ones that you might be able to use, you'll need to check one of the many comprehensive catalogs of programs for the IBM personal computers to find your category and to make sure that the programs you want can be used on your PCjr.

If you want, you can also read the ads in magazines—particularly the classified ads, where most of the specialized programs appear, and you can check with groups of computer users. If you have a modem, so your Junior can communicate by telephone, still another way to find out about special programs you need is to use the computerized bulletin boards we'll be talking about in the next chapter.

KEEPING TRACK OF DATA

One of the things that computers do best is keep track of unwieldy amounts of information. Specific kinds of data—such as our home budget information—have

programs that are tailored to their specific needs. But how do we handle lots of information in general? With a database program.

A *database program* provides a generalized way for us to store information, and later to find, sort, alphabetize, and otherwise retrieve the items we need—in the form we want. Database programs come with their own special-purpose commands, which we use to tell the data base what kind of data we are putting into place and, later, to tell the data base what data we want to look at. Here is a somewhat frivolous use for a database program: We could enter our recipes into a data base set up in such a way that, later, when we had leftover egg yolks, we could quickly find all the dishes that call for egg yolks.

The beauty of a good database program is that it allows us to input our data in a relatively raw form and then to massage, groom, and sort it all, discovering new things about the information we have stored. That's the idea of a data base—it allows a flexible approach to what we want to do with the data.

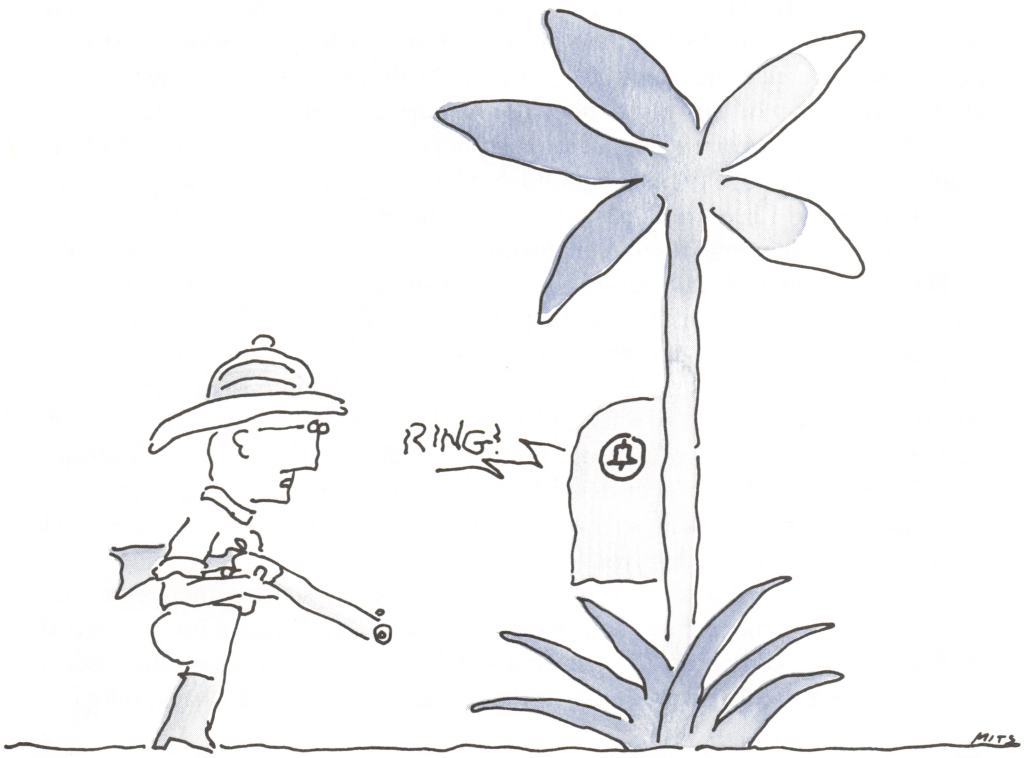
There are numerous database programs that we can use on our PCjr. The best known is dBase II™; it was developed in the early days of microcomputers, so it may not be the most advanced, but it has been the most popular for a number of years. With some investigation, you should be able to find out which one might best suit your needs.

This is a book for newcomers to personal computing, and I want you to know about database programs so you'll be aware of them when the time comes that you need one. But you should also know that a data base is a rather advanced application. If you think that you need a data base, or that you might need one in the future, I'd advise you not just to buy one and play with it. Instead, I recommend that you do some homework first—study up on the subject by reading a book or two on data bases for personal computers. And wait, if you can, until you're experienced with your computer, before you plunge into the deep and interesting world of data bases.

With this outline of the ways that we can put our Junior to work, it's now time to discover how we can get it to communicate over the telephone—a fascinating area of computer use that opens up remarkable vistas by giving our PCjr a link with the world outside.

7
COMMUNICATIONS: JUNIOR ON THE PHONE

When computer folks use the word communications, they're referring to the computer's ability to use the telephone to talk to other computers.



THE PUBLIC SIDE OF COMPUTING

Personal computing usually means private computing; the wonderful sense that the machine is completely at your command. But personal computing can be public, too—when we get on the telephone with our computer and call the world.

Using an ordinary, everyday word—communicate—to describe a technical subject is very appropriate, because using a telephone opens up the world of our computers just as dramatically as learning to talk opens up our lives.

Let's see what we need to allow our PCjrs to communicate.

EQUIPMENT FOR COMMUNICATIONS

In order to set up for communications, we'll need an enhanced PCjr, with its memory expansion and diskette drive, plus two special things: a modem, and a communications program.

Modems

A *modem*, as we mentioned early in this book, is a piece of electronic equipment that translates between computer-style signals and telephone-style signals. Telephones and computers weren't designed for each other, and they speak different electronic languages. Fortunately, though, it's not hard to translate between the two. In this translation, our Junior's computer signals have to be "modulated," or converted, so that the telephone system can carry them, and the telephone's signals have to be "demodulated" for our computer. A modem's job is to MOdulate and DEModulate, which is how the modem got its curious name.

When we go looking for a modem for our Junior, we have two main choices. We can get the modem that was specially designed to fit inside the PCjr, or we can get a standard, external modem that can be plugged into the back of the Junior.

There are advantages and disadvantages to each choice. Our Junior's internal modem is inexpensive as modems go, about $200; it fits into the PCjr's system unit, so it's out of the way, and the PCjr's internal modem is a smart modem (which we'll explain shortly). That's all good. But the internal modem has some disadvantages: There's no choice of features, its speed is modest (more on that shortly, as well), and it only fits the PCjr, so if we get another computer or move up to an IBM PC, we can't use this modem with it.

A standard modem may cost us more, and it takes up some extra desk space, but we can choose the features and price that we want, and we can use a standard modem with any computer. Also, since a standard modem is outside the system unit, we can see the indicator lights on the modem that show us what's going on.

My friends who use their modems a lot really like seeing those indicator lights.

There is one other factor that we ought to mention when we're considering whether to use the internal modem or an external one. If we get an external modem, it will plug into the PCjr's serial port, the S socket on the back of the system unit. That means we can't use the serial port for anything else at the same time. On the other hand, the internal modem comes with its own independent serial port, which is part of the PCjr's M socket. This leaves the S socket free for us to connect a printer, such as IBM's Compact Printer, and thus gives us the advantage of being able to make a printed copy of the information that's coming in over the telephone.

Now, what are the features that we might want to choose from in a modem? Mainly, they are speed and smarts. Let's explain speed first.

Speed

Computers and modems can communicate at any of several different speeds, which are measured in *baud*. Baud rates vary from a low of 110 to a high of 9600; the two rates that we're most interested in are 300 and 1200. In practical terms, for every 10 baud we'll get one character transmitted per second. So, if we're communicating at 300 baud, we'll send or receive about 30 characters per second; at 1200 baud, we'll send or receive about 120 characters per second.

Most personal computer communicating is done at either 300 or 1200 baud. The most common rate is 300, but many people find that 300 baud is frustratingly slow, while 1200 baud is gratifyingly fast. The difference between the two isn't just psychological; using the faster rate can cut down on the amount of time we tie up the phone, and it can cut down the cost of communicating by shortening our communication time. The PCjr's internal modem works at 300 baud, but not at 1200; that's probably fine, but you may prefer the faster speed.

Smarts

Modems are referred to as smart or dumb, depending on whether or not they can perform some operations for us other than just modulating and demodulating. We can send commands to a smart modem and it can recognize them and carry them out. For example, a smart modem can dial a number, answer the telephone automatically, and do other tricks. The extra features of a smart modem make it much easier to use. Communications enthusiasts wouldn't want to do without the smart features; and if you use your Junior a lot for communications, you're likely to become an enthusiast, too. As we mentioned, our Junior's internal modem is smart; if you buy an external modem, you can choose a dumb one if you wish.

That takes care of the hardware that we need to communicate. Now, what about software?

Software

To make communications work, we need a program that supervises the operation. This kind of program is called a *communications package;* this term can be a little confusing, so when you encounter it, remind yourself that a communications package is just a program that oversees the process of using your computer with the telephone.

Communications packages

There are lots of communications packages available for us to use. Two that you might consider are the Personal Communications Manager™ and PC-Talk. The Personal Communications Manager is an IBM product that has been custom tailored to take advantage of the PCjr's smart modem; the IBM brand and the Junior tailoring are two things that might make you want to use this one. PC-Talk, on the other hand, is a clever and highly regarded program written by Andrew Fluegelman. PC-Talk is published as what is called "Freeware," which means that you can try it out without buying; if you find that you like the program, you can then pay for your copy. Personal Communications Manager is about $100; PC-Talk is about $35.

Terminal emulators

The communications packages we mentioned are used for general-purpose communications on home computers. There's another kind of communications that we might want to use, though, and it calls for another kind of program.

Large mainframe computers like the giants IBM is known for are usually set up to work with lots and lots of computer terminals. These computer terminals have a display screen and a keyboard, and they're dedicated to the job of working with a mainframe.

Personal computers are much smarter than ordinary computer terminals; our PCjr can do much more than a computer terminal can. But sometimes we might like to use our home computer as if it were a terminal—for example, to get some data stored in the mainframe computer at the office. To do this, we need a kind of program known as a *terminal emulator;* an emulator knows the tricks necessary to make our home computer act just as if it were a terminal.

Since there are different kinds of computer terminals, there are different terminal emulator programs. If you need to work with a particular mainframe that expects to talk to a particular kind of terminal, you need to get a terminal emulator designed for that specific job.

In addition to the terminals that are specifically designed to work with one mainframe or another, there is a simple, universal terminal known as an ASCII terminal or, sometimes, a dumb ASCII terminal, since it has few smart features.

If we want to use our PCjr as an ASCII terminal, we don't have to buy any special programs: There's an ASCII terminal emulator built into the PCjr's BASIC language cartridge. To use this emulator, we just start up the BASIC cartridge, and enter the command TERM. When we do that, we leave the BASIC language itself, and find ourselves using the terminal emulator program. It's simple to use, and it doesn't cost anything extra. By the way, the TERM program is written in BASIC and you can take a look at it if you want, to learn how it works. There's a special trick to seeing this program, so I'll tell you how to do it. First, start your Junior, with the BASIC cartridge plugged in. After BASIC begins, switch to TERM by keying in the command TERM. Then switch back to BASIC, by pressing Fn-2. Now the TERM program is ready in memory and you can display its inner workings by keying in the command LIST.

SOME TECHNICAL NONSENSE

Generally, throughout this book, we've been avoiding complicated technical information, because it gets in the way of understanding the basics of home computing. Technical details can be very interesting and fun to play with; I've made most of my reputation, after all, from making PC technical information easy to understand. For this book, though, it's been my idea to show just how much we can learn about our PCjr without resorting to lots of technical terminology.

But with communications, things change. When we start using our PCjr for communications, we're quickly faced with more technical details than we usually have to put up with—and they're details we have to understand, at least somewhat. To help cope with this onslaught, we're going to pause to cover some of the technical nonsense that comes up when we use our computers to communicate.

What Makes It Technical

Computer communications is a very technical subject because it covers two big uncertainties.

The first uncertainty is about the telephone connection. Since our computer is soundly wired to its own parts, such as the memory and the diskette drive, it can count on those parts either to work or to be broken; it can be one way or the other, but the connection to these parts won't be irregular. With remote communications, though, we don't know what sort of rusty old phone lines might be used, and any static or disruption on the line can cause problems.

Remember that when we're talking on the telephone, we hardly notice little irregularities, but a computer talking on the telephone needs to have every bit of information passed exactly right. To deal with this problem, a computer has to use special techniques designed first to avoid errors, then to detect errors when they occur, and finally to correct errors after they are detected. So the potential irregularity of the telephone connection is one of the two main reasons computer communications is so complicated.

The other factor that makes communications complex is its use by so many different kinds of equipment. If computer communications were designed just to allow our home computers to talk to each other, then things might be fairly simple. But the ground rules and procedures for computer communications had to be created without any certainty about just what sort of machine would be using the telephone. Potentially, one end of the communications line could have a computer as smart and as flexible as our PCjr on it, while the other end of the line could be connected to something as dumb and as simple as a teletype printer. The mechanisms for computer communications had to be set up to accommodate all sorts of equipment, without any prior knowledge of what that equipment would be in any one particular instance.

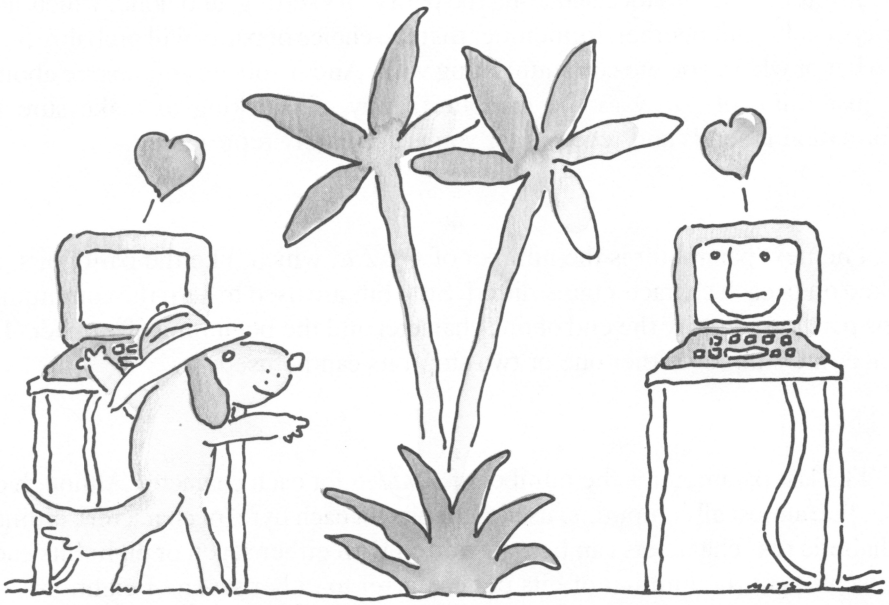
The result of all this uncertainty is that communications is just about as complicated a subject as there is in dealing with computers. Fortunately for us, a lot of the complexity is hidden away; but still, quite a bit of it springs up to frighten the unwary. What I'd like to do in this section, then, is introduce you to some of the terminology that you are likely to bump into when you try to start your Junior communicating.

What You Need to Know

Some of the messiest terms you'll run into are called the communications line parameters. Computers can talk by following different rules. The simplest and most commonly used rules are known as *RS-232*, or *serial, communications*. That is what we use with our PCjr. Within the RS-232 rules, there are some parameters that can be varied to accommodate different needs, and you are likely to see them, or even be asked to specify them.

The first thing to remember is that normally we don't get to choose what parameters are being used—instead, they are determined by whatever we are communicating with. So if you ever find a program asking you what to set these parameters to, your task isn't to decide what you want; instead, you have to find out what is needed by your communications partner.

The communications parameters we have to deal with are the speed, or baud rate; the parity; the stop bits; and the character size. We'll go over these parameters one by one.



"Baud . . . 300, parity . . . even. Check!"

Baud rate

The *baud rate,* as you already know, is the speed at which we send and receive information. A single baud is equivalent to one bit per second, and it takes roughly 10 bits—including the overhead and safety factors we discuss next—to transmit a single character, or byte, of data. So if you divide the baud rate by 10, you get a pretty accurate idea of how many characters per second can be transmitted.

As we mentioned, the most common rate for home computers is 300 baud, or about 30 characters per second, which is as fast as the PCjr's internal smart modem can work. The next most common rate is 1200 baud, which we can use with a faster, external modem. You'll probably find that, when you use communications on your PCjr, 300 baud is just fine when short messages are being sent, but it is painfully slow if you're receiving an entire screenful of information.

Parity

The next communications parameter is *parity,* which is used to test for errors in the transmission of our data. A parity bit is set electronically by a formula based on the data bits being transmitted; if any of our data get lost or scrambled, that fact can usually be detected when the recipient machine checks the parity bits (as it does automatically). There are three choices in parity: odd and even, which are

two different formulas for calculating the parity bit's setting, and none, which skips parity checking altogether. Remember that the choice of parity will probably be set by what or whom you are communicating with. And if you are still unsure about it all, just think of parity as the computer's way of checking to make sure the information it sends and receives for you is accurately represented.

Stop bits

The next parameter is the number of *stop bits*, which, like the parity bits, are tacked on to each character transmitted. Stop bits are used to help the communications partners identify the end of one character and the beginning of another. The choices are simple: Either one or two stop bits can be used.

Data bits

The last parameter is the number of *data bits* for each character. Although our PCjr, like almost all computers, uses eight bits for each byte (or character), ordinary alphabetic text characters can be transmitted with either seven or eight bits each. So, we can set the number of bits per character to either seven or eight.

With those technical details out of the way, we're now ready to look at some of the exciting things that we can do with communications.

EXCITING THINGS TO DO

There are four main things we can do when we use our Junior's communications skills. We can talk to our friends and to other personal computer users; we can talk to people via public bulletin boards; we can talk with communications services; and we can talk to computers at work. Let's take a look at each choice.

Talking to Friends: A Direct Line

When we call up our friends' computers, we can send programs and information from computer to computer. This can be a quick and easy way to share things we've written, or to get a friend's help or advice.

Suppose a friend has just written an interesting program that we'd like to try. By communicating computer to computer, we can get a copy of the program quickly and easily. Or suppose that we've just written a report; we need to get it printed out and our printer isn't working. We could send it over the telephone to a nearby friend who could print it out for us. This is the sort of thing that friends can do with computer communications.

To make this kind of communications work, all we need do is arrange with a friend when the call will be made and agree on the communications parameters

and other such details. We can do this with a human-to-human phone call. Then we can both connect our computers to the telephone, set things up, and send anything that we want to, back and forth. With the right kind of communications package, it's easy to do.

This is a simple and interesting way we can use our computer's communications ability to talk with one other computer. There's also a way to talk to many other computers: through the magic of computer bulletin boards.

The Bulletin Board: A Party Line

A computer *bulletin board* is very much like a regular notice board except that it is a personal computer that has been set up to act as a public place for us to share information. On a bulletin board, we can leave messages for others, collect messages that were left for us, post public notices for anyone to read, broadcast questions that we need answered, and so forth.

Often, a bulletin board is set up by a computer club, so that members can easily share information. The bulletin board itself consists of a computer dedicated to the job of answering the telephone and collecting and sharing information. Special communications programs have been written to handle the special requirements of a bulletin board; these programs can work automatically, answering the phone and collecting, storing, and passing out information to whoever wants it.

"Here I am."

With our PCjr, then, we can call up public bulletin boards and participate in the dialogue—receiving and contributing interesting information. To find out more about bulletin boards, contact a local computer club, or look in the club and bulletin board listings that appear in the computer magazines. You'll find all sorts of bulletin boards listed there—some for everyone, some for people with particular interests to share. By contacting the people who run these bulletin boards, you can find out the details on how to use these services.

A public bulletin board provides one particular kind of shared resource for us—a place to swap messages and information. There are also broader communications services, which give us much more, as paid services.

Communications Services

The commercial *communications services* expand the idea of a bulletin board into a vast range of services.

There are several of these services for us to use. First, there are general-purpose ones, such as THE SOURCE and CompuServe®, which provide just about every imaginable service we could want. Then, there are specialized services, such as Dow Jones News/Retrieval™, which cover a particular range of needs, such as financial information.

Generally, the communications services work this way: We begin by subscribing to the service, at which time we usually pay a one-time membership fee. After we've joined, we get an account number, a password, and local telephone numbers to call to use the service.

When we call the service, we're charged a certain amount for each hour that we're dialed in; the rate may vary according to what time of day we call, very much like long-distance telephone charges do. Our membership fee and the hourly charge cover all the standard items in the communications service; there may be a special charge for certain extra items.

What do we get with a communications service? Let's look at some examples.

THE SOURCE provides an almost inexhaustible list of different services. There is news—general news, including UPI bulletins, weather, sports, and financial information. There are business services, including portfolio management and employment services. And THE SOURCE has bulletin boards for every kind of computer interest, hobby interest, and professional interest. Private bulletin boards can be set up too, so that a business or club can have its own bulletin board that outsiders can't see.

THE SOURCE also provides shopping information and carries members' classified ads. There is travel information, including up-to-date airline schedules, so we can make airline and hotel reservations through THE SOURCE. There are computer games to play. There are also specialized computing services; for example, THE SOURCE has programs to do standard engineering calculations

and statistical work. There are educational drills, in case we don't have educational software for the subject on our own computers.

And there is more, and still more. When we look at the full list of offerings that services like THE SOURCE and CompuServe provide, it's amazing just how much they offer.

Dow Jones News/Retrieval, on the other hand, emphasizes a more specialized interest. It can give us historical and current information on more than 6,000 securities, plus detailed records on companies whose stocks are publicly traded. There's also lots of miscellaneous information, such as movie reviews, weather and sports information, and more. Dow Jones News/Retrieval tries to provide every bit of financial and general information that anyone active in finance might want.

This is only a taste of what communications services offer, but it ought to whet your appetite for more.

Talking for Work

In addition to all the kinds of communicating that we've covered so far, we can also use our Junior's communications skills for work. Some people are able to tap into their employer's computer from home and use what is called telecommuting—"traveling" to work via telephone and computer.

There are also other ways that we can use communications for work. For example, you could choose to become a free-lance travel agent by connecting to a reservation service. Architects and engineers, doctors and lawyers, can use specialized computing services, and so forth.

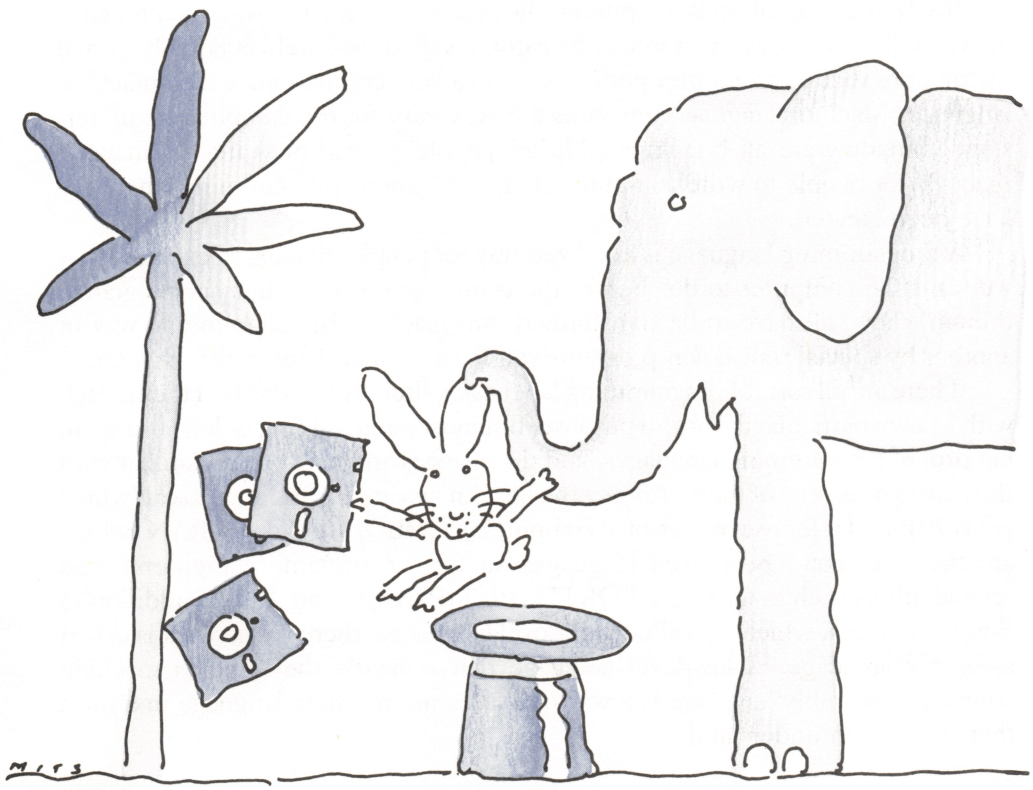
In any way that our work allows the use of computers and communications, it allows a PCjr to participate in the work, to give us greater flexibility in how and where the work is done.

Now that we've taken a look at communications for our PCjr, it's time to discover the realm of programming that we can do ourselves on this wonderful machine.

8

PROGRAMMING: MAKING YOUR OWN MAGIC

Programming is the key to creating your own magic on the PCjr. In this chapter, you'll be introduced to the main ideas that underlie programming, so you can understand what programming is all about, in a general way.



UNDERSTANDING PROGRAMMING

The whole subject of programming computers is a rich and complicated one, so even though we'll cover a lot in this chapter, we'll barely have begun. My goal here is to give you an idea of what programming is, how you can get started with it, and what is involved in doing serious programming—including both techniques and tools, and computer equipment as well. Then, in Chapter 9, I'll go on to show you the elements of the popular BASIC programming language. There, we'll put our ideas into practice and build an interesting and useful demonstration program in BASIC.

These two chapters will serve as an introduction to the whole business of programming the PCjr. If you want to go on, into your own programming, you'll be ready for books that will guide you into the details. Even if you're not planning on doing your own programming, these chapters should remove some of the mystery of the subject for you.

What Are Programs and Programming Languages?

Computers are machines that can be told what to do, and will then do it—they'll carry out our instructions. These instructions are computer programs.

Inside the heart of each computer is the processor that carries out the program; in our PCjr, this processor, or microprocessor, is called the Intel® 8088. The actual instructions that the computer performs are in a very cryptic form called machine language. Machine-language programs are very easy for the computer to understand, but they are all but impossible for people to read or write. To make it practical for people to write computer programs, various programming languages have been created.

A programming language is a stylized way for people, like us, to express what we want the computer to do. Before the computer can carry out our programs, though, they still have to be transformed into machine language in one way or another by special translation programs you'll learn about later in this chapter.

There are all sorts of programming languages, literally hundreds of them, each with its own particular flavor and purpose. But most programming is done in a small handful of programming languages, and the chances are that you are familiar with the names of several of them. Among the best known are BASIC and Pascal (which are very popular for use on personal computers), COBOL and FORTRAN (which are the most commonly used languages on large mainframe computers), and several others such as C, Logo, FORTH, PL/I (PL/one), and Ada. In addition to these languages, which are called high-level languages, there is also what is called assembly language. A low-level language that is nearly the same as machine language, assembly language is a way of expressing machine language in a form that people can understand.

So that you don't get lost in the forest of programming languages, I'll mention the languages that are used most often on the IBM personal computers. For non-professional, do-it-yourself programming, BASIC is by far the most widely used; Logo is the second most common, and it's becoming increasingly popular, especially for youngsters to play with. For professional programming, BASIC is used a lot for writing programs that don't put a demanding load on the computer (BASIC is rather slow, and sometimes unwieldy); for heavy-duty professional programming, we find assembly language, Pascal and, increasingly, C being used.

Although there are lots and lots of programming languages, each with its own particular style and format, they all have some basic ideas in common, and that's what we'll look at next.

Data

First, programs work with data or information. Data could be numbers—numeric values—such as 12, 2.5, or 1776. Or, the data could be *character strings*, which are letters of the alphabet and other characters, such as punctuation marks. Here are some examples of character strings, each enclosed in quotation marks to indicate where the string of characters begins and ends:

```
"a character string"
"A"
"Columbus sailed the ocean blue"
"1492"
"  "
```

There are some things that you should notice about these numeric and character values. Among the numeric values, there can be whole numbers, like 12, or numbers with a fractional part, like 2.5. BASIC and most other languages make a distinction between these two kinds of numbers. Part of the reason a distinction is made is that whole numbers can be represented exactly, while fractional numbers, such as 0.33 or 3.14 (the value of pi), may be only close approximations of the actual number.

For the moment, as you're just learning the basic ideas of programming, you can consider numbers to be numbers. But, when you get into the specifics of programming, be prepared to keep in mind the distinction between whole and fractional numbers. We'll cover the difference here, so that you'll be familiar with the vocabulary that's used when numbers are discussed. Then you can file this information away in the back of your mind, ready for whenever you need it.

Whole numbers are called *integers*; integers can be positive, like 12, or negative, like −52. With programming languages and computers, there is usually some limit to the size, or range, of integers we can use. In our PCjr's BASIC, integers must fall between −32,768 and +32,767.

The other kind of number has many names, including *real* and *floating point*; in BASIC two terms are used, but they mean nearly the same thing: *single precision* and *double precision*. Both single and double precision refer to numbers that can have fractional parts and that can be larger or smaller than the range of integers; the only difference between single and double precision is how accurately, or precisely, the numbers are represented.

With character values, you'll notice that they can be quite lengthy, or very short, like "A". The string "A" is one character long, but you can even have a string that has no characters in it; it's written like this: "", two quotes, with nothing in between. We can also have a character string with nothing but blank spaces in it, like the last of our previous examples.

You should also notice that 1492 is a number, while "1492" is a string, even though it is made up of numeric digits only. It is possible to translate a number into a string that represents it; and the right kind of string, such as "1492", can also be translated into a number—but still, numbers and strings are quite different things to the computer.

What We Do With Data

Our computer programs can work with these values, these numbers and character strings, doing all sorts of things with them. For example, we can do arithmetic with numbers, like this:

```
12 + 54 - 17
```

The result would be a number, 49. We can also do things with strings. For example, we can "add" two strings, like this:

```
"Columbus sailed " + "the ocean blue"
```

The result would be a longer string value, which is the two put together:

```
"Columbus sailed the ocean blue"
```

Variables

All the values, numeric and string, that we have looked at so far have been the values themselves—what in computer-talk are called *constants*. In order to do useful things with values, we need another way to work with them; this is done with *variables*. As the names imply, constants stay the same—12 is always 12, and "a rose" is always "a rose"—but the value of a variable can be changed.

Variables are one of the most important keys to understanding the whole idea of computer programming, so we need to know clearly what they are.

Essentially, a variable is a "place" in the computer where we can store a value, such as a number or a string. Think of a variable as a box that we can use to hold something. We can put something into the box: a value. Later, we can change the contents of the box: change the value of the variable. Or, think of a variable as one small part of a blank sheet of paper. We can write something (a value) on the part of the paper that we've set aside as our variable. Later, we can read what's written there, or we can erase it and write another value in its place.

This comparison to a piece of paper is a good one, because we can think of the computer's entire memory as the full page. When we use a variable in a programming language, a small part of the computer's memory (a small part of the sheet of paper) is set aside for the variable. We can read and write anything in the computer's memory, just as we can read and write anything on the page. Part of the computer's memory "page" is used to hold the program (written instructions); other parts of memory are used to hold variables (equivalent to the part of the page where we work out our calculations). When our program is done, the memory "page" is erased, and our programs and all our variables go away.

Each variable that we use in our programs is identified by a name that we give it. The name is a way for both the computer and us to keep track of the variable. We get to choose our variable names, and we can make them almost anything we want. Usually, we try to choose names that make the purpose of the variable clear.

Here are some examples of variable names that we could use with BASIC (the periods are used to punctuate the names and make them easier to understand):

```
NAME
MY.NAME
YOUR.NAME
A.RATHER.LONG.VARIABLE.NAME
```

To the computer program, each variable represents a place in the computer's memory where we can store a value. We can put a value into a variable and it will stay there safely, while the program is running, until we need to use the value. And any time we need it, we can use the value simply by referring to the name of the variable in which it is stored.

The reason we use variables is very simple: We can work with variables and the values stored in them just by referring to their names. Our programming language handles the chore of finding the right place in memory to keep the variables; we don't have to bother.

Using Variables

To see how variables are used, we'll work our way through an example. Our example will follow the format that BASIC uses, but the same principles apply to other programming languages.

If we give our computer an instruction like this:

```
THIS.YEAR.IS = 1984
```

then we have asked the computer to reserve some memory location for a variable named THIS.YEAR.IS, and we've also told it to store a value of 1984 in that location, under that variable name. That's what the equal sign, = , actually means. In computer notation, = means take the value on the right (1984) and store it in the variable on the left (THIS.YEAR.IS).

Later, whenever we wanted, we could put a different value into the variable. We could just put another constant in, like this:

```
THIS.YEAR.IS = 1985
```

Or, we could ask the computer to add one to the old value, and store the result as a new value in the same location set aside for our variable name:

```
THIS.YEAR.IS = THIS.YEAR.IS + 1
```

In this last example, THIS.YEAR.IS + 1 tells the computer to take two numbers—the number that was previously stored in the variable THIS.YEAR.IS and the number 1—and add them together. The other part of our instruction, THIS.YEAR.IS =, tells the computer to take the new value it has calculated and store it in our variable. If we wanted, we could store the new value in some other variable, like this:

```
NEXT.YEAR = THIS.YEAR.IS + 1
```

We've used numbers so far, but we can do similar things with strings, like this:

```
THIS.MONTH = "August"
NEXT.MONTH = "September"
```

So far, we've seen that computer programs work with values—numbers and characters—and that they can use named variables to store and later refer to values. What else can a computer program do?

What a Program Can Do

The main things a program can do for us fall into three categories: *operations*, *input/output*, and *logic*.

Operations

Operations are calculations, such as addition, subtraction, and multiplication, and similar work—exactly the kind of thing we saw in the preceding examples. Operations are the meat-and-potatoes of computer programs, the ordinary drudgery of getting things done.

Let's consider one of our examples:

```
NEXT.YEAR = THIS.YEAR.IS + 1
```

This example tells the computer to do two operations. First, the + tells the computer to do an arithmetic operation: to add two numbers. Then the = tells the computer to perform the operation of storing the number it has calculated into the variable named NEXT.YEAR. Both the plus sign, +, and the equal sign, =, are operations in our programming language.

When you read up on BASIC, or another programming language, some of the first things you learn are the various operations that the language can perform for you. You'll find that there are plenty—lots of fancy arithmetic, and many other operations as well. We'll leave the adventure of learning about them for the time you dig into the specifics of whichever programming language you want to use.

Input and output

Input/output, or I/O as it is called, is how a program gets its data and how it shows the results of what it has done for us. In the case of input, the program takes in data, reading it from somewhere; for example, from what we type on the keyboard. In the case of output, the program writes out data; for example, making it appear on the computer's display screen.

A program running on our PCjr can input, or read, data from several places. For example, in addition to reading what we type in on the keyboard, it can input data from diskettes. For output, our Junior's programs can write information on the display screen, or make sounds on the speaker, or write on a printer, if we have one. If we use a modem to connect our computer to a telephone line, the PCjr can also do I/O, read and write, over the telephone.

The basic idea of I/O is very simple—a program reads the data that it needs, performs the necessary operations on that data, and then writes out the results. The details, unfortunately, can get quite complicated. When you set out to learn the details of a programming language, it's easy to feel overwhelmed. So, here's some advice that's particularly important when you are learning the I/O features of a language: Don't try to master it all at once; learn the simple stuff first, then move on to the rest later.

Logic

Logic is the third category of things a computer program can do, and in many ways it is the most interesting. Logic allows us to give both power and some intelligence to our computer programs; it has two main uses here.

The first use is in making a simple choice. For example, if we had a program that was working with a variable named THIS.YEAR, we might want to test to make certain that the right kind of number was stored in the variable; it shouldn't

be a negative number or, say, earlier than 1980. So we might test for THIS.YEAR being less than zero, or less than the earliest year (1980) that we want our program to allow. To check for this in a program, we would have a logic statement like this:

```
IF  THIS.YEAR  <  1980  THEN something
```

If the < symbol is new to you, we should explain that it's a common mathematical symbol meaning "less than." So, this statement means "If the number stored in the variable called THIS.YEAR is less than 1980, then do something." When you learn programming, you'll need to become familiar with a few symbols like this, including the opposite symbol, >, which means "greater than."

In this example, the word "something" stands for whatever we would do when the year was too early. When the computer came to this part of the program, it would test whether the value stored in the variable THIS.YEAR were less than 1980; if it were, the computer would do whatever "something" was; otherwise, the computer would bypass "something." This is the essence of computer logic.

I used the word "something" here to simplify the example, so that we wouldn't get bogged down in too much detail. That's actually quite important in learning how to program well. Programming works best when we figure out the main parts first, and then come back to fill in the details. We'll talk more about this idea later in the chapter.

Loops: Logical repetition

The other main kind of computer logic is used to control the repetition of work. One of the main strengths of computer programs is that they can repeat the same work over and over again, quickly and tirelessly. But to accomplish this, we need a way of telling the computer when to repeat some work it is doing and when to stop.

The part of a program that is being repeated is called a *loop*, since the computer loops through those program instructions. So, the logic that controls repetition, or looping, is called loop logic. To get an idea of what loop logic is like, without getting distracted by too many details, let's look at an example of a program loop, but one written in our words, not in any particular programming language. We'll make our example a game, like tic-tac-toe:

```
DISPLAY THE START OF THE GAME

WHILE THE GAME ISN'T OVER DO THE FOLLOWING STEPS
      INPUT THE PLAYER'S MOVE
      IF THE PLAYER DIDN'T WIN
            THEN CALCULATE THE COMPUTER'S MOVE
END-OF-WHILE LOOP—LOOP BACK TO THE WHILE ABOVE

DISPLAY THE RESULTS OF THE GAME
```

Between the DISPLAY commands at the beginning and end there is a loop that continues to repeat until the game is over. When the game is over, the final display instruction is carried out.

This example shows us the idea of how a loop, a repetition, can be used in a program. In this case, we're using the idea of "while"—meaning that we repeat the program's action as long as some condition is met (here, the condition is "the game isn't over"). Most programming languages have several kinds of loops, each used in its own particular way to control the repetition of part of a program.

Our example also illustrates two very useful ideas that help in the development of good programs.

One of these ideas is to work out the outline of a program's logic before going into any of the details. A program, like a story, should have a beginning, a middle, and an end. You can see that our example does. The beginning of a program is often called the initialization; in our case, it's DISPLAY THE START OF THE GAME. The middle of our program (and of most well-written programs) repeats the main action until it's time to stop. The end, like the beginning, does whatever is needed to support the action in the middle. That's the essence of outlining a program's logic.

The second idea illustrated in our example is clear formatting. You'll find it enormously helpful, when working on a program, to write it in a form that makes the outline easy to follow. You can see two things that I did in this example, and they are both good programmer's tricks: I used blank lines to separate the sections of the program visually, and I used indentation to show which parts of the program (the IF and the WHILE) were controlled by the logic.

EQUIPMENT FOR PROGRAMMING

With this introduction to programming under our belts, let's consider the sort of equipment we need for programming. How should we outfit our PCjr, so that it is ready for programming work? First, we'll look at the hardware we'll need, and then we'll consider the software.

Programming Hardware

To do any serious programming on the PCjr, we'll need the enhanced version, with its diskette drive and extra memory. Without the diskette drive to give us access to other languages, the only programming that we can do is in BASIC; and even then, we can't copy our programs onto diskettes. So, an enhanced PCjr is really necessary for any programming other than simple tinkering in BASIC.

You can use any kind of display screen for programming, but the easier it is to read the screen, the better it will be for your eyes. Writing programs is very much

like writing words; so, for intensive programming, just as for word processing on the Junior, the best display screen is a monochrome composite monitor.

But we also need the display screen to test the results of our programs after we have written them. So, if color or graphics figures in your programming plans, you need a display screen that will show you what your programs create. Often, the best combination for programming is a monochrome composite monitor, to use while you are writing the program text, together with the borrowed use of your color TV set, to use while you are testing the program's display screens.

When we're writing words, we're certain to want a good printer to print our compositions. But the same thing isn't necessarily so with programs. Some people like to print copies of their programs, to study or work with, but many programmers don't find printing necessary. When I program, I never print out my programs—I just work with them on the display screen. You'll probably find the same to be true when you do your programming, so you don't need any kind of special printer for programming. If you already have a printer, it should do just fine.

That's essentially what we need for programming, as far as Junior hardware goes. We need the extra memory and the diskette drive of the enhanced PCjr, and we need a display screen or two that we can read comfortably and that can show us what our programs create. We need a printer only if we want to see our programs printed out, or if our programs will be generating printed results.

Programming Software

Next, let's take a look at what we need in the way of software for programming.

Program editors

We begin with something to write the programs with—a program editor. After all, we write programs just as we write reports, so we need something that will help us do that writing.

Some programming languages come with a built-in program editor. BASIC is like that, and so is Logo, another popular programming language. Having a built-in program editor means that the same software that translates our programs into action also assists us in the task of writing the programs. For example, the editor that comes with BASIC has some nice features that are specially tailored for BASIC. Many of the key words that BASIC uses, such as LOCATE, SCREEN, and WIDTH, can be entered just by pressing the Alt key in combination with a letter key. This type of feature can make writing BASIC programs much easier.

On the other hand, the editor features built into languages aren't the handiest or most powerful editors available. After all, BASIC's first job is interpreting BASIC programs; helping us create and edit programs is secondary. The same is true of most other programming languages that come with built-in editors. Many active programmers prefer to use a good general-purpose editor to create their programs,

even when they write in BASIC, and there are very good reasons for this choice. For example, they (and you) may want to move part of a program from one place to another—just like rearranging some of the sentences or paragraphs in an essay. With an editor program, it's easy to do; with the built-in BASIC editor, the same thing can't be done.

Of course, if we're programming in a language that doesn't have a built-in editor, we have to have some kind of editor to help us create our programs.

If you have a word processor program, it may do the job for you—but be sure to check. Most programming languages require programs to be kept in a standard format known as an ASCII text file. Some word processing programs can create ASCII text files, but others can't. Before you waste a lot of effort, make certain that your programming language can work with the programs you're creating.

If your word processor can't do the job, you need to get a general-purpose editing program. Several very good ones are available and can be used on our PCjr. IBM's Personal Editor™ is excellent; PMATE™ and EDIX™ have many fans. I use VEDIT™, so I can testify that it is very good for writing programs; the programs that made my reputation on the PC were all written with VEDIT.

So, when you set out to do any programming, in addition to choosing the right language for yourself, make certain that you have an editor or word processor that can work with your language. And then, you'll be in business for the wonderful and creative activity of writing your own programs.

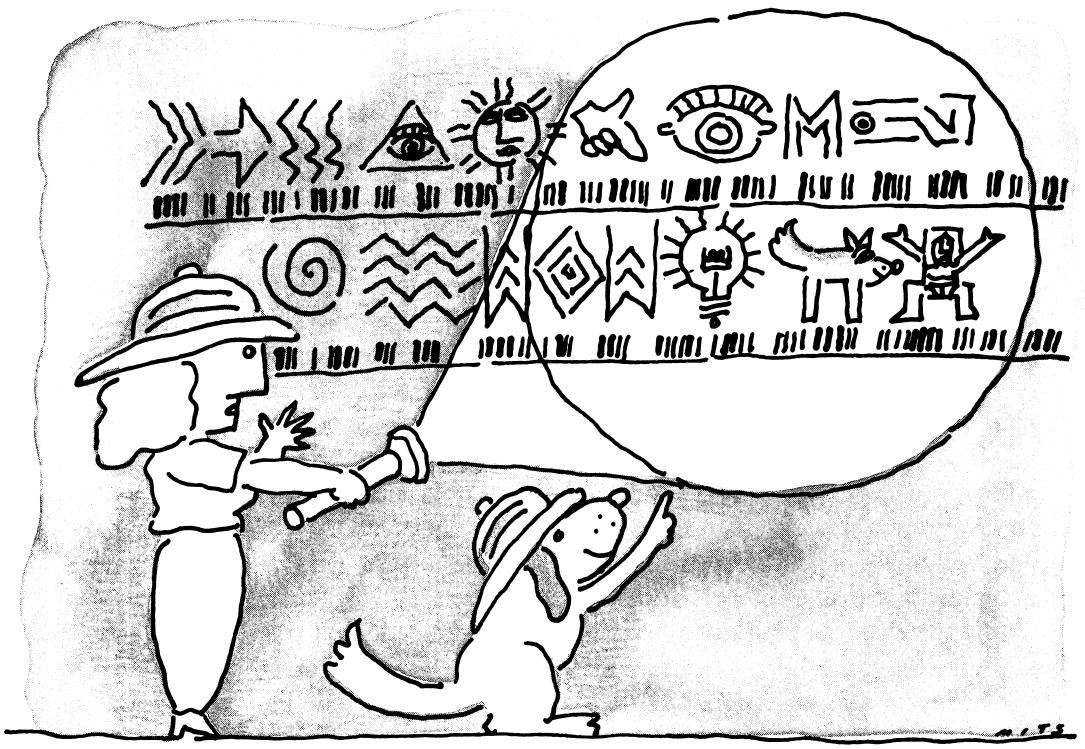
Language translators

The key piece of software we need for writing programs is, of course, the language translator that makes our programming language come alive by translating our programs into working machine language.

This translation can be done in more than one way. BASIC, for example, is an *interpreted* language. That means BASIC is translated into our computer's machine language as we're using it. Most other programming languages are *compiled*, which means that programs written in these languages are translated into machine language as a prior, separate step before the program is used. Interpreted languages are more flexible, because we can make interactive changes to our programs. Compiled programs are faster, because the translation process has already been completed before the program is used.

This choice between the speed of compiled programs and the flexibility of interpreted ones is important. Most languages, such as Pascal and assembly language, are compiled. Logo and the BASIC we work with are interpreted. There is also a compiled version of BASIC, so it is possible to use BASIC either way.

Getting the language translator that you need begins with choosing your programming language—which we'll discuss in a moment. But that isn't all. There can be several different versions of each language available. When there are, you need to choose among them.



"That's BASIC for dog."

Often, there is one choice that seems natural. For example, if we want to program in BASIC, then our natural choice is the cartridge version of BASIC that's specially made for our PCjr. But, there are other BASICs to consider. There is the version provided by IBM's BASIC compiler; it has many fewer features than cartridge BASIC, but it runs much faster. Then there is the interpreted BASIC that the other IBM personal computers use. It's very much like our cartridge BASIC, but has fewer features. If we are using cartridge BASIC and want our programs to be able to run on the other IBM personal computers, we have to develop the programs with our cartridge BASIC, but must carefully avoid using any features that are specific to our PCjr.

In some cases, several versions of a language are available, but not all of them will work on the PCjr; some may need more memory than our Junior has. Consider Pascal: There is an official IBM version of Pascal for the PC, and it would seem to be our first choice for writing Pascal programs. But IBM Pascal requires more memory than our PCjr has. So we have to look to one of the versions of Pascal that will fit into our Junior's memory.

The choices can be complicated. I wish that I could offer some simple advice, or tell you which versions of each language to use. But that's just not possible, for

the same reason that I can't recommend the best word processor or whatever: So many new programs are appearing for our PCjr that today's best recommendation could be tomorrow's second best. I'll recommend what I can in the following discussion, but you should look to your computer dealer, the PCjr-related magazines, and your fellow Junior users for the most current advice.

With that said, let's start considering what programming language you might want to use to create programs for your Junior.

CONSIDERING LANGUAGES

The programming languages we can use with our wonderful PCjr seem endless in number, but there are only a few that are really popular and really well suited for our use. We'll take a look at them here.

Before we start, you should be forewarned that the subject of programming languages is something like the subjects of religion and politics: People's opinions about them vary widely, and those opinions are often passionately held. What you'll read here is a blend of years of expert judgment, biased by my own opinions. If someone else offers you a very different opinion on programming languages, don't assume that they're wrong—or that they're right.

BASIC

The beginning point in discussing programming languages for our PCjr is BASIC, for sure. BASIC is by far the most widely used programming language for personal computers, and it's a particularly natural choice for us to use with the Junior for several reasons. One is that it is so handy. There is a simple version of BASIC built right into our Junior, and a very advanced cartridge version plugs right into the computer. You can't get any handier than that.

Another reason to use BASIC is that it includes many, many features tailored to the special abilities of our computer. When we write programs in BASIC, we have easy access to everything that makes the PCjr so wonderful. Every single part of Junior's skills is on tap when we write programs in BASIC.

BASIC is so important for our PCjr that I can safely assume that if you do any programming on the Junior, you'll at least be trying BASIC. That's why I'll be devoting all of the next chapter to showing you the basics of BASIC and why I'll be using BASIC to illustrate the process of evolving a working program.

But BASIC isn't everything; let's look at some other languages we can use.

Pascal

For programming microcomputers like our PCjr, the most widely accepted language after BASIC is Pascal, and there are two good reasons for using it.

For one thing, programming experts consider Pascal an excellent language: lean, clean, and safe to use. What do I mean when I say that Pascal is safe? I mean that the Pascal language has features designed into it that make it easier for us, first, to avoid programming mistakes, and, second, to find mistakes quickly when we do make them. Pascal was designed to incorporate many of the newest ideas about the kind of language that makes for good programming, so it can be a very good choice. My most popular programs for the IBM personal computers were written in Pascal, and I, personally, found that Pascal served my demanding needs very well.

There is also a second and very special reason for considering Pascal as a programming language for your PCjr, and that is, if school-age children will be using it. Pascal was actually created not as a working language but as a teaching language for use in university computer science courses. Many schools and colleges have adopted Pascal as their primary language for teaching programming and computer science. Also, the college advanced placement tests in the United States are now oriented toward Pascal.

Obviously it can be a real advantage for a student to play at home with a language that will later be used in school. If you know what programming languages will be used in your schools, use them; if you don't know, then BASIC and Pascal are your best bets. This educational advantage alone is a very good reason for using Pascal, in addition to BASIC, for programming the PCjr.

The flavor of Pascal

Since Pascal is so important, we ought to take the time to taste the flavor of Pascal, to see what it is like. So what is it like?

It has a structured form, which means that the elements of the language are designed to reflect the logic of a program. That's unlike BASIC, in which a program's logic can easily be lost in the BASIC format. Pascal makes use of English words for its main elements (as does BASIC).

To get an idea of what Pascal programs look like, Figure 8-1 shows the skeleton of a program we might write to play tic-tac-toe or some other board game. This isn't a complete program; it's only the outline, so we don't get lost in too much detail. The idea is just to give you a feel for what Pascal is like, and the outline will give you a basis for comparison, when we dive into BASIC in the next chapter. Read our sample program through and look at it carefully, and you'll learn a lot about the nature of Pascal.

Although Pascal will take some effort to learn, it is really no more difficult to learn than BASIC.

If you want to use Pascal with your PCjr, you'll find several versions available (but not, as mentioned earlier, the standard IBM Pascal, which was created for the PC). One version I think highly of is SBB-Pascal™, from Software Building Blocks; there are other excellent ones as well.

```
Program play_a_game;
  begin;
   set_up_playing_board;
    while not done do
      begin
        get_keystroke;
        if keystroke = end_of_game then
          done := true
        else
          begin
            move_players_piece
            if player_won then
              done := true
            else
              begin
                calculate_computers_move;
                move_computer_piece;
                if computer_won then
                done := true;
              end;
          end;
      end;
    report_results_of_game;
  end.
```

Figure 8-1. The outline of a Pascal program

Assembly Language

For the ultimate in power and control over the computer, nothing can match assembly language. Assembly language is by far the most difficult programming language to learn and the most technically demanding to use. By programming in assembly language, we use the computer's own instructions directly, without the assistance or safeguards that we get by using any other language. Using assembly language is sometimes called programming "right down to the bare metal," with everything that phrase implies.

When we write programs in assembly language, we're in complete charge of the computer. It can be a daring challenge, and a thrilling experience, but it also means that we have to do a lot more work to get anything done. While high-level languages do a lot of the work for us, in assembly language, we have to do it all for ourselves.

There are two approaches to using assembly language. One is to write complete programs in assembly. When we do this, we invest a lot of extra effort,

and in return we get programs that don't have the excess overhead, or program size, that usually comes with high-level languages.

The other approach is to use assembly language only for those few things that another language, such as BASIC or Pascal, can't do. When we take this approach, we write almost all of our program in Pascal or another high-level language, and then use what is called an assembly language *interface* to do the things that we can't do in the main language.

Assembly language interfaces are a very good choice for those who want the advantages of a high-level programming language, but occasionally need the extra power of assembly-language access to the bare metal of the computer.

My own programs are as good an example as any: For a very small part of the programs, I've needed to perform magic that could only be done in assembly language; but I wanted to avoid the massive effort that writing a large program in assembly involves. So for me, as for many other sophisticated personal computer programmers, the natural solution was to write very short assembly-language interface routines, and to write the remaining 99.9 percent of the program in my chosen high-level language, Pascal. The same thing can easily be done with BASIC and other languages. When you are ready for it, you can learn the details of performing this kind of magic from sources such as my *Mastering the PCjr, a Programmer's Reference Guide* (Microsoft Press, 1984), IBM's *BASIC* reference manual, and IBM's *Technical Reference* manual for the PCjr.

If there are two approaches to using assembly language, there are also two reasons why we might use it. One is because we have to—witness the magic assembly-language interface routines that I mentioned before. The other is because we want to. In terms of efficient use of a programmer's effort, assembly language is very wasteful. But when we want to program for the challenge of it, or when we simply want to program to deepen our understanding of the computer's inner workings, then there is nothing finer than assembly language.

When we want to work with assembly language, the natural choice is IBM's Macro Assembler™ for the PC family. Other assemblers are also available, including the famous "cheap assembler," CHASM; it doesn't have the frills of the Macro Assembler, but its cost matches its name.

Other Languages

There are other languages you might be interested in besides BASIC, Pascal, and assembly. We'll look at three of them: Logo, C, and FORTH. First, though, a warning against heartbreak.

You may be familiar with three other languages, COBOL, FORTRAN, and PL/I, that are widely used, primarily on large, mainframe computers. Don't set your heart on using them on the PCjr, though. Each of these languages is large and complicated. Because of this, all the PC versions that I know of are too big to fit

into our PCjr's memory. Of course, a smaller version may come along, and then we could use it. But for now, as far as I know, we have to put aside any thoughts of using these languages on the PCjr.

Logo

Logo is an interpreted language with some unusual features that make it very good for young children to play with. So if you want kids to get interested in programming, Logo is a good language to try.

One element of Logo, called turtle graphics, is very good for creating interesting drawings. One of IBM's programs, Turtle Power™, is produced specifically for the PCjr and is intended to introduce children to Logo's turtle graphics. That's a good place to start looking at Logo.

For the complete language (and not just the sampler that's in Turtle Power), use IBM's Logo; it runs just fine on the Junior. There are also other versions of Logo available.

C

C—that one letter C—is a powerful programming language that is quite a bit like Pascal. Where Pascal emphasizes safety and sound programming, C emphasizes raw, lean power. On the whole, though, Pascal and C are very much alike.

Expert programmers tend to favor C. It's being used more and more on personal computers, and that's why you might want to consider it. C is definitely gaining in popularity for programming personal computers.

There are several versions of C that we can use on our Junior, including the one that I use, Microsoft® C.

FORTH

FORTH is an interesting and peculiar language. Like BASIC and Logo, it is interpreted, so we can develop our programs interactively—testing them while we are still writing them. FORTH has an unusual style and flavor all its own. People who try this language usually either love it or hate it.

There seem to be more versions of FORTH than I can count; you can choose any of them.

9
GETTING A TASTE OF BASIC

Since BASIC is the most popular programming language, we should know what it's like. This is the place for us to get acquainted.

THE FLAVOR OF BASIC

BASIC is a fairly simple, easy-to-learn programming language. Although it isn't the first choice of most programming professionals, BASIC has some wonderful advantages, as we outlined in the last chapter. BASIC is very handy, since it comes on a convenient cartridge; it has a built-in editor, which means that we can use it interactively (now writing our programs with the editor, now running them, switching back and forth effortlessly); and best of all, BASIC gives us direct access to our Junior's best features, such as advanced sound production.

Let's talk a little more about BASIC's interactive quality; it's worthwhile to understand. What does it mean when we say BASIC is interactive? It means that we can write and test our programs in a smooth, continuous operation. If we find an error in the program, or if BASIC finds something wrong, we can fix the problem right on the spot and go on testing the program. Most other languages aren't interactive like this, so testing programs is a much more laborious process.

BASIC has this interactive quality because it is an interpreted language: It is translated into machine language as we're using it. And interpreted languages, as you'll recall from Chapter 8, are more flexible than compiled languages, because we can make these interactive changes to our programs. (Compiled programs are faster, because they are translated into machine language before they are used. Since there's a compiled version of BASIC, we can actually use BASIC either way.)

Trying These Examples

To show you BASIC, we'll be seeing lots of examples in this chapter. There will be two different kinds of examples here. First, we'll just demonstrate various parts of BASIC, in independent examples. Then, we'll build a complete program, piece by piece. Building this program will do three things for us: It will give us a chance to tie different parts of BASIC together; it will let us illustrate the process of creating a program in steps; and finally, the program itself will give us a useful demonstration of all the PCjr's color combinations.

If you want to try these examples yourself, here is what you do. Turn your PCjr on, after making sure that there isn't a diskette in the diskette drive. If you have the BASIC cartridge, plug it in. (If you don't have the BASIC cartridge, you can still use the Junior's built-in BASIC, but some of the advanced features that we use aren't included in the built-in BASIC.)

When you turn on the computer, without a diskette loaded, your PCjr will start up with BASIC running. As soon as BASIC is in operation, it will indicate that it is ready for your commands by displaying the word *Ok*. Feel free to key in these examples if you want—but be forewarned that what you'll be seeing isn't intended to be interesting to look at; it's intended to illustrate the basics of BASIC.

BASIC Lines

BASIC programs consist of a series of instructions, written out in lines. In order to keep track of the different lines, we give each line a line number. Here is an example of a three-line BASIC program:

```
10 CLS
20 PRINT "Hello there,"
30 PRINT "from your friendly IBM PCjr"
```

We can give our program any line numbers that we want, from 1 through BASIC's built-in limit of 65,535. Our numbers can be as arbitrary as we want, but we usually number them ten apart, as you see here, so that if we decide to add some instructions between the ones we've already entered, there are line numbers available for use.

This program consists of three lines and uses two different instructions, both of them output instructions that act on the display screen; CLS is the instruction to clear the display screen, and the PRINT instruction writes whatever information follows onto the screen.

Two Modes of Operation

There are two ways we can work with BASIC: the *direct mode* and the *indirect mode*. When we put our instructions in with line numbers, as in lines 10 to 30 of our example, we're working in indirect mode. Each line that we put in is tucked away by BASIC as part of a program. When we want to execute, or run, that program, we give BASIC the command:

```
RUN
```

and then BASIC carries out the steps of the program.

If you want to try it yourself, start up BASIC and type in each of the three lines, pressing the Enter key at the end of each line. Then, type in the word RUN and press the Enter key. You'll see this little BASIC program spring into action.

We don't have to work this way, though. We can have BASIC perform our instructions as soon as we put them in; this is known as direct mode. To tell BASIC to perform the instructions immediately, rather than saving them as program instructions, we simply leave off the line numbers. So, we could have entered our sample program without line numbers, like this:

```
CLS
PRINT "Hello there,"
PRINT "from your friendly IBM PCjr"
```

and each instruction would have been carried out as it was entered.

If you're using your computer, try it both ways. Type in the three lines without numbers, and compare the results with what happened when you typed them in with line numbers and told BASIC to RUN them.

Most of the time that we work with BASIC, we work in indirect mode, putting programs in and then running them. In fact, to write programs in BASIC we have to work in indirect mode, since BASIC doesn't hang on to any instructions that we give it in direct mode. Yet, direct mode can be very handy, particularly when we're learning what each of the BASIC instructions does. All we have to do to try a BASIC instruction is type it in, in direct mode, and see what happens. Direct mode is particularly good for learning the features of the PCjr and of BASIC.

Comments, to Tell What We're Doing

When we write a program, we ought to put *comments* in it, to help us and others understand what the program is doing. Comments are ignored by the computer, but they help us understand what is going on. Comments are put in like this:

```
10 REM this is a REMark or comment
20 SCORE = 0 ' anything after a quote is comment
30 ' or use a single quote in place of REM
```

Each of these lines shows a slightly different way we can put a comment into a program. If we try to read anyone else's BASIC programs, we need to know how to tell a comment from a working part of the program.

BASIC Variables

BASIC variables can have names as long or short as we want them to be, from this:

A

to this:

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Usually, we give our variables some meaningful name that indicates what we're using them for. We can use periods inside variable names to punctuate them and to make them easier to read, like this:

TOTAL.SALES
HIGHEST.GAME.SCORE

As we mentioned in Chapter 8, there are character string variables and numeric variables. We have to let BASIC know what kind of variable we want, and one way to do this is with a special symbol that we put at the end of the variable

name. If we want a variable to hold a string, we make $ the last character of the name. If we want it to be an integer variable (which has whole numbers for its value), we end the name with %. Finally, for fractional numbers, we use ! or #.

You'll recall, from the last chapter, that programming languages treat fractional numbers separately from whole numbers. In BASIC there are two kinds of fractional numbers, called single and double precision. They both hold the same kinds of numbers, but double-precision numbers can be more accurate. We identify single-precision variables by ending their names with !, and double-precision by ending their names with #.

Here are examples of all four types of variable names in BASIC:

CITY$	character string variable names end in	$
DAYS%	integer variable names end in	%
TOTAL!	single-precision variable names end in	!
SCORE#	double-precision variable names end in	#

But you'll notice that all the variable names we've looked at so far haven't ended in any of these symbols. If we don't indicate which kind of variable, BASIC assumes we want it to be single precision. Single precision is the everyday workhorse of BASIC. We use single precision for most numbers, unless we want the faster calculation speed we get by using integers or the greater accuracy we get by using double-precision numbers.

We set variables like this:

```
10 CITY$ = "London"
20 DAYS% = 14
30 TOTAL! = TOTAL! + SUB.TOTAL!
```

Lines and Instructions

It's good to put each BASIC instruction (or statement, as it is also called) on its own line, but we don't have to. We can put several instructions on one line, with a colon, :, separating them:

```
10 CITY$ = "London" : DAYS% = 14
```

This line has two statements, or instructions, in it. The first assigns a string, "London", to a string variable, and the second assigns the number 14 to an integer variable.

A Short Sample Program

Next, let's look at a little program that illustrates some logic, combined with input and output.

```
10 PRINT "What is the capital of England?"
20 INPUT CITY$ ' get a reply on the keyboard
30 IF CITY$ = "London" THEN PRINT "Right!"
   ELSE GOTO 10
```

In this example, line 10 asks a question by printing a message on the screen. Line 20 uses the INPUT instruction, which waits for us to type something in on the keyboard; whatever we type in is placed in the string variable CITY$. In line 30 we test for the right answer; if it's correct, we congratulate ourselves—otherwise we use the GOTO instruction to start over at line 10.

Trying a WHILE Loop

We can rewrite this program to show how loop logic can be used to repeat part of a program:

```
10 CITY$ = "" ' set the city name to nothing
20 WHILE CITY$ <> "London" ' repeat until right
30   PRINT "What is the capital of England?"
40   INPUT CITY$
50 WEND ' this marks the end of the WHILE loop
60 PRINT "Right!"
```

This version of the program takes more lines, but this kind of loop logic can make a program easier to write and understand. The more complicated the program, the more valuable WHILE loop logic can be.

Tidy Formats

You'll notice that lines 30 and 40, which are the instructions that are being repeated by the WHILE loop, are indented. We don't have to do this, but the indentation helps us see just where the WHILE loop is, and what instructions are being repeated by it. This is the sort of thing we can do to help make our programs clearer and easier to understand. In the last chapter, we talked about this idea in general, and we also saw an example of how indenting is done with the Pascal programming language. Now we're starting to see how it can be done with BASIC.

If you get deeply into programming, you'll find tidy formatting a tremendous help in keeping your programs straight. Professional programmers are often fanatical about keeping their programs tidy; they've learned through experience that it's much harder to write an error-free program when it's written messily.

Repeating with FOR

WHILE loop logic works best when we want to repeat our action until something indicates that we're done (in our example, we're done when the answer

is right). There is another kind of loop logic we use when we want our programs to repeat some specific number of times; this is the FOR-NEXT loop, which is used like this:

```
10 FOR I = 1 TO 10
20   PRINT "Using the PCjr is fun!"
30 NEXT I
```

This loop will repeat 10 times, and each time the variable named I will take on a new value, from 1 to 10. Again, you'll notice we indented line 20, just to make things clearer.

Here's a little sidelight you might find interesting, as well as useful, when you do your own programming. When we're using variables to hold some information that means something (like TOTAL.SCORE or CITY$), we usually give the variable a good, meaningful name. But when we need a variable just for convenience, as in keeping count of some repeated activity, it's customary to give the variable a terse name, like the I in this example. Programmers often use I, J, and K for such variables; it's a tradition that mathematicians use.

Going Somewhere Else

When we need to jump from one part of our program to another, we can use a statement called a GOTO. You saw the GOTO being used in the first version of our "What is the capital of England?" program. There is also another kind of jump-somewhere-else instruction that is used for a very special purpose—for something called a subroutine.

Getting Help with Subroutines

When we write programs, we often find that there are some instructions we need to duplicate in different parts of the program. For example, it's common for programs to pause so that we can look at what they have on display, and then wait for us to press a key to indicate that we're ready to continue. These instructions might be needed in many different parts of the program, and it would be wasteful to duplicate them each time. The solution to this problem is called a *subroutine*—a part of itself that our program can jump to, to carry out instructions, and jump back from when it's done.

When we find that we have some instructions we need to use from different places in a program, we set them up as a subroutine. Here is how it is done: First, we find some line numbers that we won't be using in the rest of the program—for example, maybe line numbers starting with 500. Then, as in the next example, we place our common instructions in those line numbers.

```
500 ' this is the beginning of our subroutine
510 ... this line stands for whatever
        work our subroutine does
520 RETURN ' this ends the subroutine
```

The RETURN instruction in line 520 is the end of the subroutine; it tells BASIC to jump back, or return, to whatever part of the program used this particular subroutine.

In order to use a subroutine, we have to jump to it in a special way that lets BASIC know that it should keep track of where we came from. For an ordinary jump from one part of the program to another, we use a GOTO; but for a subroutine, we use a GOSUB (meaning GO to a SUBroutine). So, each time we need to use our subroutine starting at line 500, we just put in a program instruction that tells BASIC to:

```
GOSUB 500
```

Then BASIC will jump to the subroutine, and the RETURN statement will jump back.

In our example, line 510 stands for whatever work we might have the subroutine do. What sort of work might that be? As we mentioned before, it could be some instructions that we need to use at several points in the program. Instead of duplicating our instructions each time, we just put a GOSUB at each place where we need the work done. Even when we don't invoke subroutines more than once, it can still be a good idea to use them, just to tidy up the program's logic. In this case, we create subroutines to help break down a program into its separate logical parts. You'll see a sample of that shortly.

There is a lot to learn about the BASIC programming language, and as you can imagine, we've only just scratched the surface here. But what you've seen so far should give you a fair idea of what BASIC is like.

BUILDING A PROGRAM

Now, to finish off our introduction to BASIC, we're going to build up a program both to illustrate how the process of writing a program can be done, and to give you a good-sized sample of BASIC program instructions.

Our program will do something interesting and useful for the PCjr—it will show all the color combinations that can be created on the display screen. This program can help you choose what colors you might want to use with your Junior.

Follow along with this program listing and the discussion that goes with it, and you'll learn a lot about BASIC. Don't worry if you don't understand everything that is going on in this program, though. Think of this as something like jumping into the deep end of the pool to see how well you can swim.

Start With an Outline

We start, as we always ought to start, with a simple outline of the program:

```
1000 ' Demonstrating all the color combinations
1010 '
1020 ' main program outline
1030 '
1040 GOSUB 2000                      ' initialize
1050 FOR BACKGROUND = 0 TO 7         ' background color
1060   FOR INTENSITY = 0 TO 1        ' intensity
1070     FOR FOREGROUND = 0 TO 7     ' foreground color
1080       GOSUB 3000                ' show the info
1090     NEXT FOREGROUND
1100   NEXT INTENSITY
1110 NEXT BACKGROUND
1120 GOSUB 6000                      ' do finish-up
1130 END
```

Lines 1000 to 1030 are just introductory comments separated by blank lines. Line 1040 invokes a subroutine to do any start-up work. We'll put our start-up instructions in a subroutine just to keep them from cluttering up the program outline and to help make the program easier to understand. That's an example of what we just mentioned: using subroutines to help make the logical organization of a program simpler and cleaner. It is also an example of something we covered in the last chapter: creating the logical outline first, then going back and filling in the details. Our subroutines will hold the details of the work we've outlined here.

Next, in lines 1050 to 1110, we set up some FOR-NEXT loops to work through all the color combinations that the PCjr can display. There are background colors and foreground colors (that's the 0 to 7 you see in lines 1050 and 1070), and the foreground colors have two intensities, bright or dim (the 0 to 1 in line 1060). We set up three different loops, so that all the combinations of these colors can be worked out. (When you learn more about the PCjr's technical details, you'll understand how these colors work; for now, all you need to know is that there are foreground and background colors, and two intensities. Later in the program we'll translate the color numbers, 0 through 7, into names.)

The loops, taken together, select the color combinations one by one. Inside all the loops, at line 1080, is what we do—show the information—after each possible color has been selected. As we did for the starting instructions, we've moved that part off to a subroutine, because it will be easier to work on as a separate section of the program. The final part of our program outline, line 1120, uses a subroutine to perform any finishing-up instructions.

Begin at the Beginning

Now that we have the outline, let's start filling in the blanks. For our start-up program, we set up a subroutine at line 2000:

```
2000 '
2010 ' initialization  subroutine
2020 '
2030 KEY OFF          ' clear the keys off the screen
2040 SCREEN 0, 1      ' use the right screen
2050 WIDTH 40         ' set the display width
2060 COLOR 2, 0, 0    ' choose our colors
2070 CLS              ' clear the screen
2080 PRINT "Color demonstration program"
2090 PRINT "        from"
2100 PRINT "Discovering the IBM PCjr"
2110 PRINT
2120 RETURN
```

Here, lines 2000 to 2020 just tell us what this subroutine is for; it isn't necessary, but it's a very good idea. Lines 2030 to 2070 do what's sometimes called "housekeeping" among programmers. Housekeeping is just taking care of those annoying preparatory details that have to be done. In this case, our housekeeping gets the display screen ready; since there are so many ways the screen can work, we make certain it is set up the way we want it. When you learn more about BASIC, you'll understand what we're doing here, and why.

Then lines 2080 to 2110 announce what we'll be doing. Finally, line 2120 tells the computer that this subroutine has come to an end, and that it should return to the rest of the program. Remember, that's the way all subroutines finish up.

Doing the Main Work

With that introductory programming out of the way, we can put together our subroutine at line 3000, the one that acts on each color combination the FOR-NEXT loops create. Here is that subroutine:

```
3000 '
3010 ' report on one color combination
3020 '
3030 COLOR FOREGROUND + INTENSITY * 8, BACKGROUND
3040 PRINT
3050 PRINT "  ";
3060 IF FOREGROUND = BACKGROUND THEN COLOR 7, 0 :
     PRINT "(this would be ";
```

```
3070 IF INTENSITY THEN PRINT "bright ";
3080 COLOR.NUMBER = FOREGROUND
3090 GOSUB 4000 ' translate number into a name
3100 PRINT "on ";
3110 COLOR.NUMBER = BACKGROUND
3120 GOSUB 4000 ' translate number into a name
3130 IF FOREGROUND = BACKGROUND THEN PRINT ")";
3140 PRINT "  ";
3150 GOSUB 5000 ' check if time to pause
3160 RETURN
```

As always, the first few lines, 3000 to 3020, tell us what the subroutine does. Line 3030 is the one that actually tells BASIC to set the colors that have been chosen. Lines 3040 to 3050 get ready to print. Line 3060 checks whether we have a color combination that can't be read (such as white characters on a white background); if that's the case, the color is changed to something readable (black and white), and a message is printed to indicate that we are bypassing the unreadable color combination. Line 3070 tells us if the intensity of the foreground is bright.

Lines 3080 and 3090 use a subroutine to translate the foreground color number into its name; lines 3110 and 3120 do the same thing for the background color.

Finally, we have an example of a subroutine used more than once, to avoid duplicating our work.

This program will be running through all the color combinations, and if we just let it run, the colors would roll off the screen before we had a chance to look them over carefully. So line 3150 uses a subroutine that pauses every few lines to let us look at the colors closely.

There's one aspect of this subroutine that we ought to pause to mention, because it has a great deal to do with what separates good programs from poor ones. When we're creating a program, we're usually caught up in the main idea of it; we want to get the overall purpose of the program accomplished. That's all very good, but before we're finished, we need to pay attention to the details, too. In the overall plan of a program, many of the details can seem annoying or distracting; but it's a willingness to polish all the details that distinguishes the best programmers.

As examples of attention to detail, consider two parts of this subroutine: One part, in lines 3060 and 3130, checks for the unreadable combination of foreground and background colors that are the same. The second part, in line 3150, makes the display pause periodically, so the information doesn't disappear from the screen before we can see it. When you write your own programs you should look for, and include, such polishing touches.

(By the way, though I've been patting myself on the back for good attention to detail, there is a flaw in what I did here. See if you can figure out the flaw; then see if you can discover the good reason why I left this flaw in. Write me, and I'll tell you if you're right.)

Naming the Colors

Now, let's see the subroutine at line 4000:

```
4000 '
4010 ' translate a color number into its name
4020 '
4030 IF COLOR.NUMBER = 0 AND INTENSITY = 0
     THEN PRINT "black ";
4040 IF COLOR.NUMBER = 0 AND INTENSITY = 1
     THEN PRINT "gray ";
4050 IF COLOR.NUMBER = 1
     THEN PRINT "blue ";
4060 IF COLOR.NUMBER = 2
     THEN PRINT "green ";
4070 IF COLOR.NUMBER = 3
     THEN PRINT "cyan (blue-green) ";
4080 IF COLOR.NUMBER = 4
     THEN PRINT "red ";
4090 IF COLOR.NUMBER = 5
     THEN PRINT "magenta ";
4100 IF COLOR.NUMBER = 6 AND INTENSITY = 0
     THEN PRINT "brown ";
4110 IF COLOR.NUMBER = 6 AND INTENSITY = 1
     THEN PRINT "yellow ";
4120 IF COLOR.NUMBER = 7
     THEN PRINT "white ";
4130 RETURN
```

As you can see, this subroutine is very simple—it just prints a different name for each color number. You'll notice some more attention to detail: In two cases, the name of the color is different, depending upon whether it is the bright or dim intensity. Again, that's the sort of thing your programs should do.

A Subroutine to Pause

The subroutine at line 5000 is next:

```
5000 '
5010 ' check if time to pause
5020 '
5030 IF FOREGROUND < 7 THEN RETURN
5040 COLOR 2, 0
5050 PRINT
5060 PRINT
5070 PRINT "Press any key to continue..."
5080 IF LEN (INKEY$) = 0 THEN GOTO 5080
5090 RETURN
```

This subroutine checks to see whether we have come to color 7; if so, it prints a message and then pauses, in line 5080, waiting for us to press a key on the keyboard. It pauses at color 7, because that's the end of each set of foreground colors—a natural place to stop.

And a Subroutine to End

Finally we have our finish-up subroutine, at line 6000. This routine just prints a couple of messages to tell us that we are done:

```
6000 '
6010 ' finish-up  subroutine
6020 '
6030 PRINT
6040 PRINT "End of color combination program."
6050 PRINT
6060 PRINT "Returning to BASIC... "
6070 RETURN
```

This program is fairly simple, but it illustrates most of what goes into writing a BASIC program. The whole listing is shown in Figure 9-1; if you key this program into your own PCjr and run it, you will see a nice demonstration of the colors on your computer, and which ones work well together. And that could be very useful to you, whenever you write your own programs.

```
1000 ' Demonstrating all the color combinations
1010 '
1020 ' main program outline
1030 '
1040 GOSUB 2000                        ' initialize
1050 FOR BACKGROUND = 0 TO 7           ' background color
1060   FOR INTENSITY = 0 TO 1          ' intensity
1070     FOR FOREGROUND = 0 TO 7       ' foreground color
1080       GOSUB 3000                  ' show the info
1090     NEXT FOREGROUND
1100   NEXT INTENSITY
1110 NEXT BACKGROUND
1120 GOSUB 6000                        ' do finish-up
1130 END
```

(continued)

Figure 9-1. The color combination program

```
2000 '
2010 ' initialization  subroutine
2020 '
2030 KEY OFF              ' clear the keys off the screen
2040 SCREEN 0, 1          ' use the right screen
2050 WIDTH 40             ' set the display width
2060 COLOR 2, 0, 0        ' choose our colors
2070 CLS                  ' clear the screen
2080 PRINT "Color demonstration program"
2090 PRINT "       from"
2100 PRINT "Discovering the IBM PCjr"
2110 PRINT
2120 RETURN

3000 '
3010 ' report on one color combination
3020 '
3030 COLOR FOREGROUND + INTENSITY * 8, BACKGROUND
3040 PRINT
3050 PRINT "  ";
3060 IF FOREGROUND = BACKGROUND THEN COLOR 7, 0 :
     PRINT "(this would be ";
3070 IF INTENSITY THEN PRINT "bright ";
3080 COLOR.NUMBER = FOREGROUND
3090 GOSUB 4000 ' translate number into a name
3100 PRINT "on ";
3110 COLOR.NUMBER = BACKGROUND
3120 GOSUB 4000 ' translate number into a name
3130 IF FOREGROUND = BACKGROUND THEN PRINT ")";
3140 PRINT "  ";
3150 GOSUB 5000 ' check if time to pause
3160 RETURN

4000 '
4010 ' translate a color number into its name
4020 '
4030 IF COLOR.NUMBER = 0 AND INTENSITY = 0
     THEN PRINT "black ";
4040 IF COLOR.NUMBER = 0 AND INTENSITY = 1
     THEN PRINT "gray ";
4050 IF COLOR.NUMBER = 1
     THEN PRINT "blue ";
4060 IF COLOR.NUMBER = 2
     THEN PRINT "green ";
```

(continued)

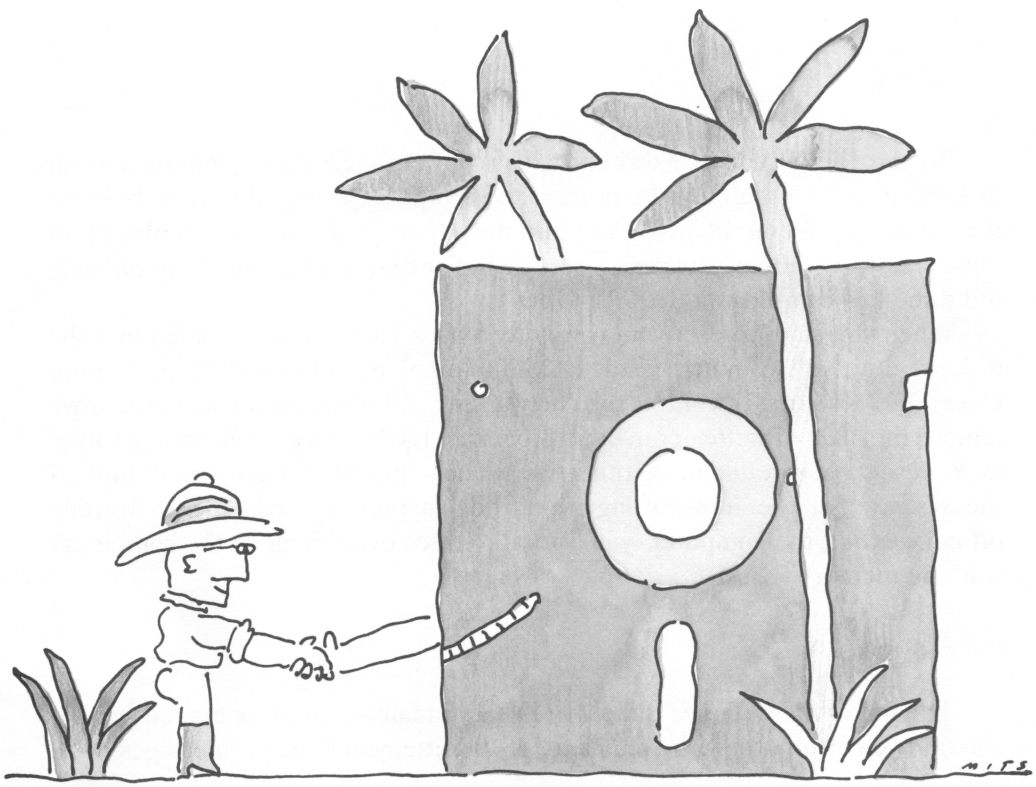
Figure 9-1. The color combination program (continued)

```
4070 IF COLOR.NUMBER = 3
     THEN PRINT "cyan (blue-green) ";
4080 IF COLOR.NUMBER = 4
     THEN PRINT "red ";
4090 IF COLOR.NUMBER = 5
     THEN PRINT "magenta ";
4100 IF COLOR.NUMBER = 6 AND INTENSITY = 0
     THEN PRINT "brown ";
4110 IF COLOR.NUMBER = 6 AND INTENSITY = 1
     THEN PRINT "yellow ";
4120 IF COLOR.NUMBER = 7
     THEN PRINT "white ";
4130 RETURN

5000 '
5010 ' check if time to pause
5020 '
5030 IF FOREGROUND < 7 THEN RETURN
5040 COLOR 2, 0
5050 PRINT
5060 PRINT
5070 PRINT "Press any key to continue..."
5080 IF LEN (INKEY$) = 0 THEN GOTO 5080
5090 RETURN

6000 '
6010 ' finish-up subroutine
6020 '
6030 PRINT
6040 PRINT "End of color combination program."
6050 PRINT
6060 PRINT "Returning to BASIC..."
6070 RETURN
```

Figure 9-1. The color combination program (continued)

10 WORKING WITH DOS

Now that we've covered the elements of programming, it's time for us to learn about DOS, the Disk Operating System that makes it possible for our PCjr to work in the same league as the other IBM personal computers.



DOS AND DISKETTES

The subject of DOS and the subject of the diskettes our PCjr uses are thoroughly intertwined, because lots of the things DOS can do for us are related to diskettes and the information stored on diskettes. So, we need to understand one to understand the other. We'll concentrate here on learning about DOS, but we'll also include some important sidelights on diskettes and the data that we store in diskette files.

Remember while you are reading this, that you can only use DOS on your PCjr if your computer has the memory expansion and diskette drive option.

WHAT DOS DOES

What is DOS and what does it do for us? First of all, DOS is a master set of programs that we buy on diskette. The main role of DOS is to supervise the running of our computer, so DOS relieves the rest of our programs—and us—from many tiresome chores.

To use DOS, we have to start it and put it in charge of our computer. So let's look at that first—how we get DOS going.

STARTING DOS

To start DOS, we place a diskette with a copy of the DOS programs on it in our diskette drive and then turn on or reset the computer. Remember that there are three ways to reset our PCjr: We can turn the power off and on; we can plug in, or remove, any cartridge; or we can reset our Junior from the keyboard, by pressing the three-key combination Ctrl-Alt-Del.

When you start or reset your computer with a DOS diskette loaded into the diskette drive, the computer reads DOS into memory and sets DOS into action. Once DOS is started, it continues to be working and ready for service, even if we remove the DOS diskette or run other programs. DOS stays in memory as a sort of ever-present background to anything else we do—provided that we don't turn off the computer, or reset it by putting in a cartridge or pressing Ctrl-Alt-Del. Turning off or resetting the computer, you'll recall, erases everything in the computer's working memory (RAM).

Greetings from DOS

The following sequence is the way DOS normally starts up operations, and if you start your computer with a regular DOS diskette, you'll see the messages we're

about to describe. But it is possible to tell DOS to skip over these starting steps and begin by running some program. When you start your computer with a DOS diskette set up for some program that you have bought, be aware that the start-up process might go differently.

When DOS starts operation, it normally asks us to key in the date and time. DOS will automatically keep track of the date and time, and this is useful to us when we create and update files of information.

DOS asks for the date like this:

```
Current date is  Tue 1-01-1980
Enter new date:
```

The current date DOS shows when you first start up doesn't mean anything, since DOS hasn't had a chance to learn the true date. (Later, as you'll see, we can ask DOS to change the date; then, it's handy to have DOS show us its current date.) We key in the date using the same format that DOS shows. For example, we might type in:

```
7-4-1984
```

for July 4th, 1984; we don't put in the day of the week (DOS figures that out), but we do have to press the Enter key after we finish typing in the numbers.

DOS is reasonably flexible and intelligent when it comes to how we can type in the date. We don't have to type the day as 04, instead of 4, or the month as 07, instead of 7 (but we can if we want to). Likewise, we can leave off the century, and just type the year as 84.

After we enter the date, DOS asks for the time:

```
Current time is  0:00:12.34
Enter new time:
```

And we respond with something like this:

```
10:23
```

Just as with the date, we can either put in or leave off zeros in front of the hour and minute. But, we do have to indicate a.m. or p.m. in the number we put in for the hour. DOS uses a 24-hour clock, so if the time is after noon, we add 12 to the hour. For example, we'd type 14:00 for two o'clock in the afternoon; if we typed 2:00, then DOS would think we meant two o'clock in the morning.

After we have entered the date and time, DOS gives us a starting message like this:

```
The IBM Personal Computer DOS
Version 2.10 (C)Copyright IBM Corp 1981,
1982, 1983

A>
```

That tells us DOS is up and ready for work.

The last part of this message, A>, is called the DOS "command prompt." When we see the command prompt on the screen, followed by the blinking cursor, DOS is asking us, or prompting us, to enter a command—and that's one of the things we'll talk about next.

THREE THINGS DOS DOES FOR US

Now that we've seen how to get going with DOS, we need to find out what it can do for us. As we've mentioned, DOS takes care of a lot of behind-the-scenes details that are necessary to keep the computer running smoothly, but which are mostly hidden from our eyes. What we can see, however, are the three main things that DOS does for us: accepting our commands, running our programs, and managing our diskettes. Let's take a look at each of these three jobs.

Accepting Commands

First, DOS accepts our commands. DOS takes on the job of asking us, "What do you want the computer to do?" and then it responds by doing what we tell it. This works very simply. When DOS is ready to accept a command from us, it gives us its command prompt, A>.

A command to DOS is the name of something that we can ask DOS to do for us. In actuality, DOS is asking us to give it the name of some program we want it to start up for us. In computer-talk, starting a program is called executing it, or running it. So, when we type in a command for DOS, we're really asking DOS to run a program with that name.

For example, if we want to use BASIC, we can just key in the command:

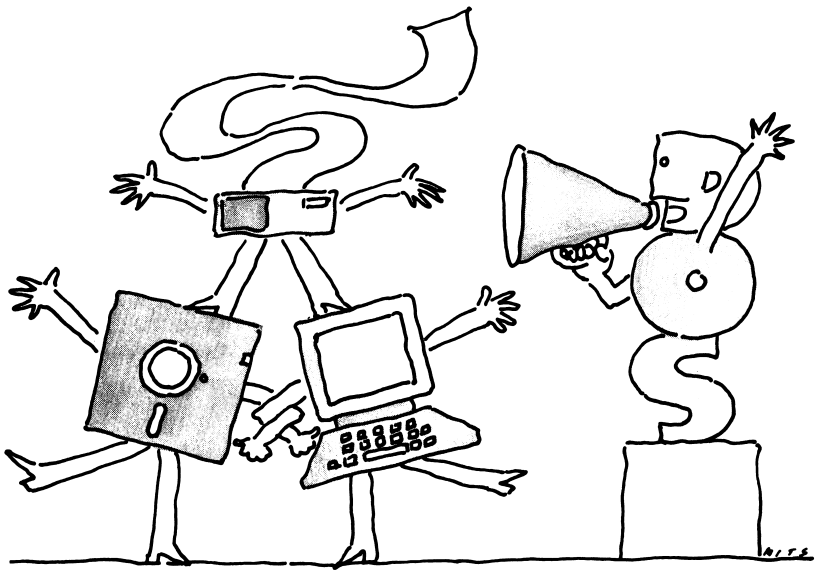
```
BASIC
```

What we key in will appear on the screen immediately after the command prompt. So, what we see on the screen is:

```
A>BASIC
```

The A> part is DOS's prompt message to us, and the BASIC part is our command message to DOS.

When we've typed in a command, we follow up by pressing the Enter key. DOS will then look at what we've typed in and carry out whatever command we asked it to, if the command makes sense to DOS. In this example, when we type in the command BASIC, DOS will activate the BASIC programming language—which it can do for us as long as we have the BASIC cartridge plugged into our computer. (If we don't have the BASIC cartridge plugged in, DOS won't know



"A one and a two. . . ."

what to do with our command, because the PCjr's built-in BASIC must be activated without the aid of DOS.)

There are many other commands we can enter. For example, we can change the date and time with the DATE and TIME commands. If we enter either of these commands, DOS will show us what it thinks is the current date or time and then ask us for a new date or time. We can then either key in a new value, or we can leave the date and time unchanged by just pressing the Enter key.

DATE is the name of a program that displays and changes the date; TIME is also the name of a program. BASIC, too, is the name of a program—a program that makes the BASIC programming language work for us.

Command etiquette

By the way, you should know that when we give commands to DOS, we can type them in either upper- or lowercase, or even a mixture of both. It would make no difference to DOS whether we asked for the BASIC language like this:

```
BASIC
```

or like this:

```
basic
```

or even like this:

```
BaSiC
```

Uppercase or lower, it's all the same to DOS.

In most books, including the DOS manual itself, commands are usually shown in capital letters to help make the commands stand out. But actually, we usually type commands in lowercase, since that's more convenient. Don't worry that you have to type things carefully in capital letters.

What's that again?

When we give DOS a command, such as DATE, TIME, or BASIC, DOS tries to find a program with that name. If we put in some gobbledegook, or if we type in the name of a command that isn't available at the moment, DOS will complain to us with this message:

```
Bad command or file name
```

That is DOS's way of telling us we didn't give it a command it knew how to carry out. For the moment, we'll pass over what a file name is, or why a command might be available to use at one time and not at another; we'll discover the answers to those mysteries later. What we need to know now is that, when we see this message, DOS is telling us that it can't carry out whatever command we gave.

Running Programs

When we do give DOS the name of a command it knows how to carry out, DOS performs the second of the tasks that we mentioned before: DOS executes, or runs, a program.

As long as a program is running, it is more or less in charge of the computer. For example, if we run the BASIC program, then BASIC will perform all of its magic until we tell it to stop. Each program has its own unique way of being told to stop—for BASIC, it is the command:

```
SYSTEM
```

When we give BASIC this command, it quits operation and returns control of the computer to DOS. With DOS back in charge, we get a new command prompt:

```
A>
```

telling us that DOS is in action, and that it is awaiting our next command.

Using DOS programs

Once DOS is running, we can use any program designed to operate with DOS—and they are the vast majority of programs for the IBM personal computers.

Most programs can't just run by themselves without the benefit of DOS, any more than most cars can drive around without the benefit of roads. DOS provides

the equivalent of paving, street lights, and traffic control to make it easier and safer for programs to get their work done. While some programs operate like off-road vehicles, most programs for the IBM personal computers take advantage of the civilized benefits of DOS.

Some of the programs we use with DOS are included on the DOS diskette. One of them is the FORMAT command, which prepares our diskettes to be used with DOS. (The FORMAT command turns out to be one of the keys to the successful use of our computers, so we'll be talking more about it later.)

But our main reason for using DOS is so we can use all those programs that are written to take advantage of DOS's services. After we have gotten DOS into operation by resetting or starting the computer with a diskette containing the DOS programs, then we can place any program diskette into our computer's diskette drive and run it simply by giving DOS the name of the program as a command. Any program diskette that we buy should include clear instructions about the names of the programs that are included. Later in this chapter, we'll also see how we can use DOS to discover the names of all the programs that are on a diskette.

Managing Diskettes

The third of the three jobs DOS does for us is manage our diskettes. Our diskettes are like filing cabinets or storage drawers that are available to our old friend, the handyman. When we use the analogy of the computer as a handyman and the memory as a workbench, we can think of diskettes as being like a toolbox that we can bring to the handyman.

We can have many different diskettes, and so we can present many different "toolboxes" to the computer. Each diskette can have stored on it its own collection of programs—tools for the handyman. When we place one diskette into the computer's diskette drive, any programs on that diskette become program commands that DOS can carry out for us. If we take the diskette out, then DOS can't use those command programs—those tools aren't available to our handyman.

Toolboxes can be quite orderly, or they can be a jumbled mess. It's the same with diskettes. One of DOS's main jobs is keeping our diskettes orderly and supervising our use of the storage space that is on each diskette. DOS not only lets our computer read programs and other information off our diskettes, DOS also lets us write, or store, information on each diskette. Naturally, for everything to run smoothly, we need to be sure that all the programs and data stored on each diskette aren't jumbled together, that each item on the diskette has a name so we can identify it, and that we don't run out of room on the diskette. All these tasks and more, are part of the diskette management function supervised by DOS.

And all these things we've mentioned are what DOS does for us: It accepts and processes our commands, finds the programs we ask it to run, assists in the running

of those programs with the equivalent of road maps and traffic lights and, finally, it provides organization and management of the contents of our diskettes.

Summing Up DOS

Let's summarize how we use DOS. First, we have to set DOS into operation, by turning on, or resetting, our computer with a DOS diskette loaded into the diskette drive. After we enter the date and time, we get DOS's starting messages and command prompt (A>); then, we can run any programs that we want. We can use some of DOS's own programs, including BASIC. Or, we can load a program diskette and use any of the programs on that diskette. Any program that we run will make use of the services that DOS provides, including use of the diskette for data storage. When a program ends, it turns control of the computer back over to DOS, which asks us for the next command with its command prompt.

Once we have DOS started, using it is effortless—DOS doesn't require any special attention or work on our part, and that is the beauty of using an operating system, such as DOS.

Next, we'll see how we can team our diskettes and DOS. In the process, we'll take a look at how we prepare and use diskettes. We'll also have a chance to learn some of the most important and commonly used DOS commands.

TEAMING DOS AND DISKETTES

To make the best use of DOS and our programs, we need to team them up on our diskettes. This teamwork is particularly important with computers like our PCjr, which have only a single diskette drive. As you'll see when you learn more about using DOS and programs for DOS, there are slightly different steps to follow when a computer has two disk drives, as the more expensive IBM personal computers usually do. Using diskettes is just a bit more complicated when our computer has one diskette drive, but it's still quite easy to get the full benefit of IBM personal computing with our PCjr and its single drive.

As you might imagine, all the details of how to use DOS and our PCjr is subject enough for a book all to itself; we can't go over every important topic here. What we will do is go through what it takes to get DOS and our diskettes working in close harmony, so you can begin to understand what's involved in working with DOS.

Let's go over the main things we have to do to get DOS and our diskettes to work together. The best place to start is by considering what we use diskettes for.

Three Uses for Diskettes

There are three uses that we have for diskettes. First, we use them to start up DOS itself; to do this, we have to have what is called a DOS system-formatted

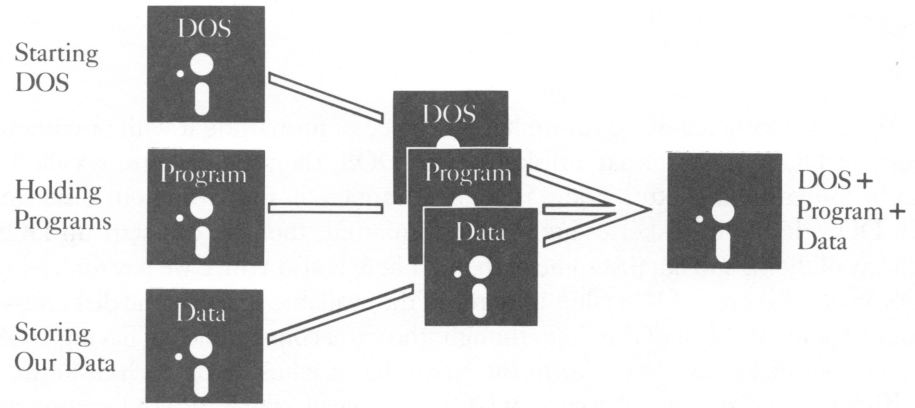


Figure 10-1. Three uses for diskettes

diskette—that is, a diskette that contains the DOS program. Second, we use diskettes to load programs into the computer, so DOS can find and execute them. Third, we use diskettes to hold any information, or data, our programs work with.

We can, if we need to, use different diskettes for each of these three purposes, but that can be rather inconvenient. We may have to keep changing diskettes in and out of our Junior's diskette drive; this process is called disk swapping, and after awhile it can get quite annoying. Sometimes, though, it's possible to have one diskette that will serve all three purposes, as you can see in the diagram in Figure 10-1. We'll explain how to put such a diskette together, after we cover some of the basics we need to know.

Formatting Diskettes

First, whenever we get new, blank diskettes, we have to format them with the FORMAT command (which comes on our master DOS diskette) so that they are ready to use. What is formatting?

Formatting is analogous to taking a blank piece of paper (a blank diskette) and ruling guidelines on it (formatting it). A diskette can't be used until it has been formatted, but once the FORMAT command rules these "guidelines" on the diskette, the rest of DOS, and all our DOS programs, can read and write information on it.

Let's pause for a warning here. We format new, blank diskettes when we want to get them ready for use. We can also format old diskettes that we've been using, when we want to wipe them clean of anything that has been written electronically on them. But we *don't* format diskettes that are holding something we need; when we buy programs, we don't format the diskettes they come on—that'll destroy the programs themselves.

System Formatting

When we format a diskette, we have a choice of formatting it with or without a copy of DOS. If we format a diskette with DOS, then the diskette is called a system-formatted diskette. There is a real advantage in formatting our diskettes with DOS. If all our diskettes are system formatted, then we can start up DOS with any of them, and not just some of them. There is also a price we pay for having DOS on our diskettes: DOS takes up some of the available space on the diskette—about 10 percent. Most of the time, though, the extra convenience of having DOS on all of our diskettes is well worth the space that it takes up on each diskette.

To format a diskette with a copy of DOS on it, we use the FORMAT command like this:

```
FORMAT/S
```

The /S part of the command is our instruction telling DOS to make the diskette a system-formatted, rather than an ordinary diskette.

Formatting Practice

It's very good practice to format some diskettes when you're just starting out with your computer, for a couple of reasons. First, the formatting gives you some hands-on experience with a diskette operation; and second, it leaves you with a stack of diskettes that are ready to be used. You'll find that it's always good to have some formatted diskettes ready for whenever you need them.

Here's how to format diskettes, step by step.

First, we have to start up our computer, with a copy of our master DOS diskette; this diskette has what's needed to start DOS, and it also has the FORMAT command on it.

Next, we type FORMAT after the A> command prompt. If we want to format our diskettes as DOS system diskettes, we type /S after the command. Then we press Enter, and the formatting begins. DOS tells us:

```
Insert new diskette for drive A:
and strike any key when ready
```

At this point, we remove our main DOS diskette and put one of our blank, new diskettes into the diskette drive. If we forget to remove our main DOS diskette, we'll accidentally reformat it: *oops!* No more DOS.

We tell FORMAT that we're ready by pressing any key on the keyboard, and FORMAT replies:

```
Formatting...
```

Then the diskette drive works for a while, formatting our diskette. When it's done, we get these messages:

```
              Format complete
    362496 bytes total disk space
    362496 bytes available on disk
Format another (Y/N)?
```

It's handy to format diskettes in batches, so the FORMAT command is set up to repeat the operation. If we want to format another diskette, we press the Y key, and FORMAT starts over from the "Insert new diskette" message. When we don't want to format any more, we press N.

When we format with the /S option, to include a copy of DOS on the diskette, things go pretty much the same, except that when we're done, we get a more detailed message.

```
              Format complete
System transferred
    362496 bytes total disk space
     40960 bytes used by system
    321536 bytes available on disk
Format another (Y/N)?
```

This shows that a copy of DOS is taking up some of the space on our diskette.

When we format a system diskette this way, the diskette contains a copy of the working heart of DOS; that's what takes up the "40960 bytes used by system" reported in the preceding message. We don't, however, get a copy of all the helper programs that come with DOS, such as the FORMAT program itself. If we also want any of those on our diskettes, we have to copy them from our main DOS diskette onto our other diskettes. We'll learn about that in the next section.

Copying Onto Our Diskettes

Once we have formatted a blank diskette, we can copy whatever we want onto it with another DOS command, the COPY command. Except when programs are copy protected—to prevent them from being passed on to people who haven't bought them—we can make copies of our programs from one diskette to another.

Making copies of programs is very important for two reasons. First, whenever possible we should avoid using our original copy of any program, in case we damage the diskette that it is stored on. Instead, we normally copy programs from the original diskettes to the diskettes where we will use them.

The second reason it is important to copy programs is that we can combine different programs on the same diskette, to put together handy combinations. Among the programs that we're likely to include in our combinations are some of DOS's helper programs, such as FORMAT. (Later in this chapter we'll also look at

some other commonly used helper command programs: MODE, CHKDSK, and DISKCOPY.)

Copying our programs onto formatted diskettes is valuable, but copying them onto system-formatted diskettes puts us even further ahead. Then, we can use the same diskettes both to start DOS and to run the programs we want. And, if we include on the same diskettes any of the information, or data files, the programs expect to find, then we're still further ahead.

If one diskette contains DOS, a copy of a program we want to use, and also any data files that the program uses, then we have a diskette that is a complete functional unit with everything we need to start the computer and do a particular kind of work.

That's the teamwork we were talking about earlier, in saying that we could prepare one single diskette that would take care of all three functions: starting DOS, loading our program, and storing the data that the program uses.

To take this same idea one step further, consider that sometimes we need to use several different programs to accomplish what we want. So, the way we really want to put our diskettes together is to have on them—if there is room for everything—the heart of DOS, copies of each of the programs we want to use together, and also any data files that those programs use. Some of the programs that we are likely to want to include on our working diskettes are a few of the helper programs, such as the FORMAT program, that come with DOS. Whatever DOS programs we want to combine with our other programs, we can copy onto the diskettes that we use.

When we can put all of these parts together on a single diskette, we gain tremendously in convenience and ease of use. Setting up diskettes like this is one of our most effective ways to team DOS and our diskettes.

After you have some experience with your computer, you'll be able to put this idea to good use; but don't expect to work it all out at once. To be successful, you must have a good idea of which programs you want to use together, and what data files you need to keep with them. Also, you have to know how much room you have on your diskettes. In some cases, you'll have plenty of room; in other cases, you'll run out of space and you'll need to decide what you really want to keep together on your diskettes and what you can do without. All this comes with experience.

When I first got started with my IBM personal computers, it took me awhile to figure out these details, and as I went along, I found ways to refine what I was doing. After a few tries, though, I had my diskettes very handily organized—and you will too, before very long.

One Important Extra

There is one more thing we need to consider about the teamwork between DOS and our diskettes. If any of our diskettes are not DOS system diskettes, there

is one part of DOS, called COMMAND.COM, that we should copy onto them anyway. Here's why.

When we finish using a program, sometimes DOS has to reload part of itself from the diskette. The reason is simple: To make as much as possible of the computer's memory available for our programs, DOS is willing to sacrifice the memory space where its own COMMAND.COM works. If one of our programs does use this space, DOS must reload COMMAND.COM from diskette in order to continue working when the program is done.

You may be wondering what happens when DOS does need to reload COMMAND.COM, but the program isn't on the diskette that is currently in our diskette drive. In this case, DOS gives us the message:

```
Insert COMMAND.COM disk in drive A:
and strike any key when ready
```

and we know what to do when we see that message.

Any system-formatted diskette will have this COMMAND.COM part of DOS on it. If we aren't using a system-formatted diskette, DOS may still need to read in a fresh copy of COMMAND.COM from diskette after we have finished with one of our programs. So, to complete the teamwork between DOS and our diskettes, we need to make certain that even our non-system diskettes contain a copy of COMMAND.COM. Remember, then, that our system diskettes will automatically have a copy—it's only our non-system diskettes that need it.

How do we copy COMMAND.COM onto our diskettes? It's very simple. First, we put either our main DOS diskette or a system-formatted diskette in our Junior's drive. Then we enter the command:

```
COPY A:COMMAND.COM B:
```

This command tells DOS to copy the file from one diskette ("A") to another ("B"). DOS will tell us when to switch from one diskette to another.

Naturally, if you are just beginning to work with your PCjr and with DOS, you may not be sure of everything that is important in getting DOS to cooperate with your diskettes. Don't worry. As you become more familiar and comfortable with the use of DOS, you will come to understand everything you need to, in order to get this teamwork going.

UNDERSTANDING FILE NAMES

The information we keep on diskettes is stored in files. When we use these files, we refer to them by name. At this point, we ought to learn a little more about file names, so we can better understand how they are used, and why they are important as, indeed, they are.

The Structure of File Names

Each diskette file has a name, and the name has two parts. The first part is called the filename (itself); the second part is called the extension to the file name.

Filenames proper

The first part, the filename itself, can be as short as one character, or as long as eight. These are some filenames:

A
FILENAME
1776

Filenames are usually made up of letters of the alphabet; we're allowed to use numbers, or even some punctuation symbols, but we can't use blank spaces. So, for example, this is a proper filename:

1-2-3

We've shown the filenames in capital letters, which is the way that DOS works with them. When we type them in, however, we can use upper- or lowercase letters freely—it doesn't matter at all to DOS.

Extensions

The second part of the file name, the extension, is used as a sort of supplement to the main part. A file name doesn't have to have an extension, but if it does, the extension can be up to three characters long. When we write out a file name with its extension, we separate the two parts with a period, like this:

COMMAND.COM

You'll recognize that file name—it's the part of DOS that I recommended you make sure is on all your diskettes. The filename is COMMAND, and the extension is COM. Together, the complete file name is COMMAND.COM.

Here are some examples of files with various names and extensions:

FORMAT.COM
SAMPLES.BAS
LIST.1
LIST.2
1.ABC
PHONES.OLD
PHONES.NEW

The use of extensions

What are the extensions used for? They are used as a way of categorizing files, to help tell what kind of information is in them.

When we create files on our own, we can do anything we want with the extensions. Usually, though, extensions are used to tell something about what's in the files, and DOS makes use of this information. For example, as we'll see later, it's the extension that lets DOS know which of our files are programs and which aren't. Other programs use the extensions to help tell which data files belong to them and which don't. All in all, this part of a file name can be very helpful.

FINDING OUT WHAT'S ON A DISKETTE

To keep track of the files on a diskette, DOS maintains a directory that lists the names of the files; the directory also includes the sizes of the files, and the date and time each of the files were created or last changed.

(By the way, this date and time marking of each file can be very useful when we're trying to figure out such things as, "When did I do that?" or "Which file has the latest version of my data?" so I would recommend highly that you always enter the correct date and time when DOS asks you to during the start-up routine.)

The DIR Command

There is a very simple way for us to find out what files are on a diskette. DOS provides us with a command program called DIR, which shows us the directory listing of the files on a diskette. For example, here is what DIR tells us about a newly formatted DOS system diskette. We enter the command like this:

```
DIR
```

And we are shown, along with some other information, the directory entry for one file:

```
COMMAND  COM     17792  10-20-83   12:00p
```

What does this line tell us? It tells us that there is a file on the diskette with a filename of COMMAND and an extension of COM. (The full name of the file is COMMAND.COM, but, as you can see, the DIR command has its own way of showing us the name.) The number that follows the name, 17792, is the size of the file in bytes. Following that are the date and time that the file was created or last changed. If there were more files on the diskette, we would see them all listed, one by one, each on a separate line.

When we use the DIR command, we get even more information than just the list of files from the diskette. We also find out some things about the diskette itself. Let's look at all the information that DIR shows us about a freshly formatted system diskette:

```
 Volume in drive A has no label
 Directory of  A:\
COMMAND  COM     17792  10-20-83  12:00p
       1 File(s)     321536 bytes free
```

To help you understand all this, let's talk our way, piece by piece, through the various things that DIR has told us.

As it turns out, DOS can identify a diskette by a label that can be recorded on it. This is a magnetically recorded label that DOS reads; it has nothing to do with the paper label pasted on the top of the diskette. We can use a DOS option to label a diskette when we format it, or we can put a label on a diskette with a special labeling program.

Usually, however, a diskette doesn't have any label, and that is what the first line of our DIR information tells us: The volume (meaning diskette) in drive A (that's our one and only diskette drive) has no label.

The second line of information announces that a list of files follows; DOS calls this the Directory of A:\, which is just computer-talk for the files on the diskette in drive A. And after that, of course, comes our list of files—only one file in this case.

The last line of the DIR listing is especially interesting and useful for us. It gives us a count of the number of files (handy when there are lots of them) and, even more importantly, the line shows us the amount of storage space left on the diskette. In this case, we have 321,536 bytes of space free, or available for use.

Once we become familiar with the sizes of our files, and with the kinds of programs and data we want to put together on a diskette, then this information about how much free space remains on a diskette is very useful. With it, we can judge what the diskette has room for, and we can thus tell whether we have plenty of space or whether we are running short and perhaps should remove some of the files we use the least.

Finding Our Programs

With this background, we are now ready to learn one of the handiest tricks of all. With any diskette, we might wonder, "What programs are on that diskette? What commands can DOS execute from that diskette?" The DIR command gives us the means of finding out.

Four kinds of programs

In addition to the data files we create and name, there are four different kinds of commands and programs we might find on a diskette. Each of them has its own identifying filename extension. Three of the four work as DOS commands; the fourth kind work as BASIC programs, which are used slightly differently.

The three extension names for DOS command programs are BAT, COM, and EXE. If a diskette contains a file with any of these three extensions, we can just type in the name of the file, and DOS will carry out the command for us. When we type in the command name, we don't have to include the extension—just the filename part itself will do.

As you might imagine, there is a reason why three different filename extensions are used for DOS command programs; when you learn more about DOS, you'll learn how each of these three types of programs takes a different form.

From a practical point of view, though, these three types are nearly the same: We can key in any of their filenames as the name of a command, and DOS will find the command and execute it. For example, if we put a copy of our main DOS diskette into the diskette drive and request the directory, we'll see in the DIR listing that there is a file named FORMAT.COM, which is the format command. And as we've seen, to run the format command, we just type in its filename, without the extension.

Our diskettes might also hold BASIC programs. We can ask DOS to run them, but the procedure is a little different. First, before we can use any BASIC program, we must have BASIC ready on our computer. This means we must have the BASIC cartridge plugged into the PCjr's system unit. So, once we have the BASIC language ready, in one of our Junior's cartridge slots, we can then run any BASIC programs that we have on diskette.

While the other diskette commands have filename extensions of BAT, COM, or EXE, our BASIC programs all have extensions of BAS, short for BASIC. While the other commands are performed just by our keying in their filenames, for BASIC programs we type the word BASIC before we type the filename of the program.

For example, there's a BASIC program file named SAMPLES.BAS which comes with DOS; it's on the DOS Supplemental Programs diskette. To run the SAMPLES program, we key in the command BASIC, followed by the filename of the BASIC program, like this:

```
BASIC SAMPLES
```

This two-word command works in two steps: First, it instructs DOS to start running the BASIC language; second, it tells BASIC to find and run the program named SAMPLES.BAS. Although the mechanics are different, the effect is much the same as when we tell DOS to run a command program, such as FORMAT.

Two Kinds of Directory Listings

There are two ways we can use the DIR command to get a list of the commands and programs on a diskette. One is simply to use the command:

```
DIR
```

to get a list of all the files on a diskette. Although this method is simple and easy, it doesn't separate our commands and BASIC programs from the list of anything else, such as our own data files, that might be on the diskette. The other, more selective way to use the DIR command is to tell it to list only the files that have a certain filename or extension. For example, to see all the BASIC programs on a diskette, we enter the command:

```
DIR *.BAS
```

This tells the DIR command to list any filename (that's what the asterisk, *, means), but only if it has BAS as an extension. Since BASIC programs stored on diskette all have BAS as an extension, this command will show them all.

If we wanted to see all the commands on any diskette, and all the BASIC programs as well, we could enter four DIR commands, one right after the other, and get the complete list in four parts. We'd do it like this:

```
DIR *.BAT
DIR *.COM
DIR *.EXE
DIR *.BAS
```

After each command, DOS would list the files with that particular extension.

Chances are that you won't want to go to the trouble of keying in all four DIR commands. More likely you'll want to just look at the BAS files, or just the two command files, COM and EXE. Or, as you use DOS and diskettes more, you might want to see all the files you've created with the extension of, say, FUN or OLD or NEW. Conversely, you might want to see a list of all your files named LETTER, regardless of whether you had given them the extension MOM, DAD, or SIS. You could see the list easily by typing LETTER.*. Whatever you choose to do, the DIR command is ready to show you whatever is on your diskettes.

This, in brief, is what diskettes are about, and how we can find out what we have stored on them. As you use DOS and diskettes more, you'll come to understand more and more about how all this works. For now, you have the basics that you need.

LEARNING SOME MORE COMMANDS

So far, we've covered three of the commands that DOS puts at our fingertips: FORMAT, to prepare new diskettes; COPY, to copy files from one diskette to another; and DIR for directory listings. Now let's go over a few more DOS commands—some of the ones that you are most likely to use when you are just getting to know your computer. This will give me a chance to guide you through a little more hands-on experience. Then we'll finish with one more way we can create teamwork between DOS and our diskettes.

Using MODE for the Screen

As we've seen, there are several kinds of display screens, including TV sets and special computer monitors, that we can use with our PCjr. Usually, computers show 80 columns, or spaces, of information across the full width of the display screen. But, if we're using a TV set as our screen, this 80-column mode is difficult to read. So the IBM personal computers have another screen mode, the 40-column mode, which spreads 40 spaces across the screen width. In 40-column mode, the characters on the screen are fatter and easier to read.

Like its bigger brothers, our PCjr can work in either 40- or 80-column mode. There is one difference, though: Our Junior starts out in 40-column mode, while the other IBM personal computers start out in 80-column mode. That's because IBM wanted to make it easier for us to use a TV set with the Junior. But if we're using a computer monitor for our display screen, we'll probably want to switch to 80-column mode, to get as much information on the screen as possible. The MODE command does this for us.

To experiment with the MODE command, start your Junior with your main DOS diskette (which has a copy of the MODE command on it). You'll be in 40-column mode to begin with. Try switching to 80-column mode, like this:

```
MODE 80
```

Then switch back, like this:

```
MODE 40
```

If you experiment with both modes, you can see which works best for you and the display screen that you are using. If you want a quick and easy way to get a screenful of information to look at each time you switch modes, try this command:

```
DIR *.COM
```

There's one important fact that you need to know about the 40- and 80-column modes. Luckily, many programs can adapt themselves to use either mode, but

some programs require one or the other. Plenty of serious-minded programs simply have to use 80-column mode, because they have so much information to display. Be prepared to experiment with any programs you use, to see which mode is needed or works best for you.

COPY and DISKCOPY

Earlier in this chapter, we touched on the COPY command, which can copy a file from one diskette to another. We used it to copy COMMAND.COM, like this:

```
COPY A:COMMAND.COM B:
```

You ought to know more about the COPY command, and you ought to know about another command, DISKCOPY. Both can be used, though in different ways, to copy information.

Copying lots of files

If we want to copy a single file from one diskette to another, we can do it just as we did with COMMAND.COM. If X stands for the full file name, with extension, then a command like this one will copy that file from one diskette to another:

```
COPY A:X B:
```

In all these COPY operations, A: stands for the first diskette, and B: stands for the second. When the COPY operation is going on, DOS will tell us when to switch from one diskette to the other.

We don't have to copy just a single file. If we want to, we can ask COPY to copy lots of files. It's done similarly to the way we asked DIR to list all the files with the same extension. For example, if we wanted to copy all the BASIC programs from one diskette to another, we could do it like this:

```
COPY A:*.BAS B:
```

By using *.BAS for the list of files to copy, we've told the COPY command to copy all the files that have BAS as an extension. For example, there are twelve BASIC programs on the DOS Supplemental Programs diskette. Try using the preceding command to copy them to one of the diskettes that you formatted earlier. Then you'll have a copy of these programs to experiment with.

It's also possible to copy all the files on a diskette. That's done by telling DOS to copy *.*, which means any filename with any extension: That's every file.

We can practice this variation, by copying all the files on our main DOS diskette to a diskette that we've system formatted. Doing this will give us a spare

diskette that has all the DOS programs on it. This is the command we key in:

```
COPY A:*.* B:
```

If you have your computer ready, practice doing this, too.

By copying from our original DOS diskette to one of our system-formatted diskettes, we get a complete copy of our original, main DOS diskette. Whenever we're working with DOS, that's the diskette we should be using, so that we can safeguard our original copy. Then, if anything goes wrong with the diskette we're using, we will still have the original so we can make another copy.

Another way to copy

If you've been trying the COPY examples shown here, you've discovered two things that make the command inconvenient: You need to have a formatted diskette ready to copy onto, and you have to switch diskettes back and forth for each file being copied. There is another way to copy that's quicker and more convenient: the DISKCOPY command.

DISKCOPY will copy the entire contents of a diskette, in one operation. If the diskette you're copying to needs formatting, DISKCOPY will take care of that automatically. To speed things along, DISKCOPY copies as much of the contents as it can in each step, so we have to switch diskettes back and forth only a few times for an entire diskette, instead of once for each file. As an extra convenience, when it's done, DISKCOPY will ask you if you want to repeat the operation, just as the FORMAT command does. This makes it easier to use DISKCOPY with a batch of diskettes.

To try this command, take your main DOS diskette and make some copies of it like this:

```
DISKCOPY A: B:
```

At this point, DISKCOPY may look much better for the job of copying diskettes, but it has its disadvantages, too. We'll take a look at them as we compare COPY and DISKCOPY.

Comparing the two ways to copy

There are advantages and disadvantages to using either COPY or DISKCOPY. Here's a quick summary, so you can decide which you want to use each time you need to copy some information.

DISKCOPY is quicker when we're copying the entire contents of a diskette, and it can spare us the chore of formatting, as well. But DISKCOPY will make an exact copy from one diskette to another, and this has a couple of disadvantages. We can't use DISKCOPY to add files to a diskette that already contains information: Since DISKCOPY makes an exact copy of the original diskette, it will wipe away

any files on the target diskette. On the other hand, COPY is just dandy for adding files to a diskette that already has information stored on it.

Also, DISKCOPY duplicates the format from one diskette to another. Our PCjr has a double-sided diskette drive, but most program diskettes come to us in single-sided format (just in case our computer has only a single-sided drive). Our PCjr can work with either diskette format, but it's wasteful to use only one side of the diskette, as we would by copying with DISKCOPY. But, if we first format a blank diskette, it'll automatically be double sided; then we can use the COPY command to copy programs from a single-sided diskette, and have plenty of room left over. Obviously, that can be a real advantage in terms of storing other programs or data files on the same diskette.

Finally, the COPY command can work around the bad spots that occasionally crop up on a diskette, and COPY can improve the way that our files are laid out on the target diskette. DISKCOPY is more rigid: It can't do either of those things.

So, we see a simple trade-off between the two ways of copying. For faster copying (when there are many files) and automatic formatting, use DISKCOPY. For more flexibility in combining files and changing from one diskette format to another, use COPY. Feel free to use either one, to suit your needs.

Getting Extra Teamwork

We've seen how several DOS commands, such as MODE, FORMAT, and DISKCOPY can be very useful when we're working with DOS. Since these commands are so useful, we may want to keep them handy at all times, by copying them onto our various working diskettes. We mentioned before that it's a good idea to copy COMMAND.COM onto all the diskettes that we work with. It's an equally good idea to copy any DOS commands we use frequently, so that they are ready to be used no matter what diskette we are working with.

To copy any of these commands to other diskettes, we just load our main DOS diskette, the one with all the programs on it, into the diskette drive. Then, one by one, we copy the command files that we want to our other diskettes. For example, we copy the MODE command like this:

```
COPY A:MODE.* B:
```

We give the filename as MODE.*, so that no matter what extension the command program file has, the program will get copied. These files usually have the extension of COM, but a few commands have the extension EXE; using .* will copy them, whichever extension they have.

For any other commands we want to transfer to other diskettes, we just substitute the command name for MODE in our example.

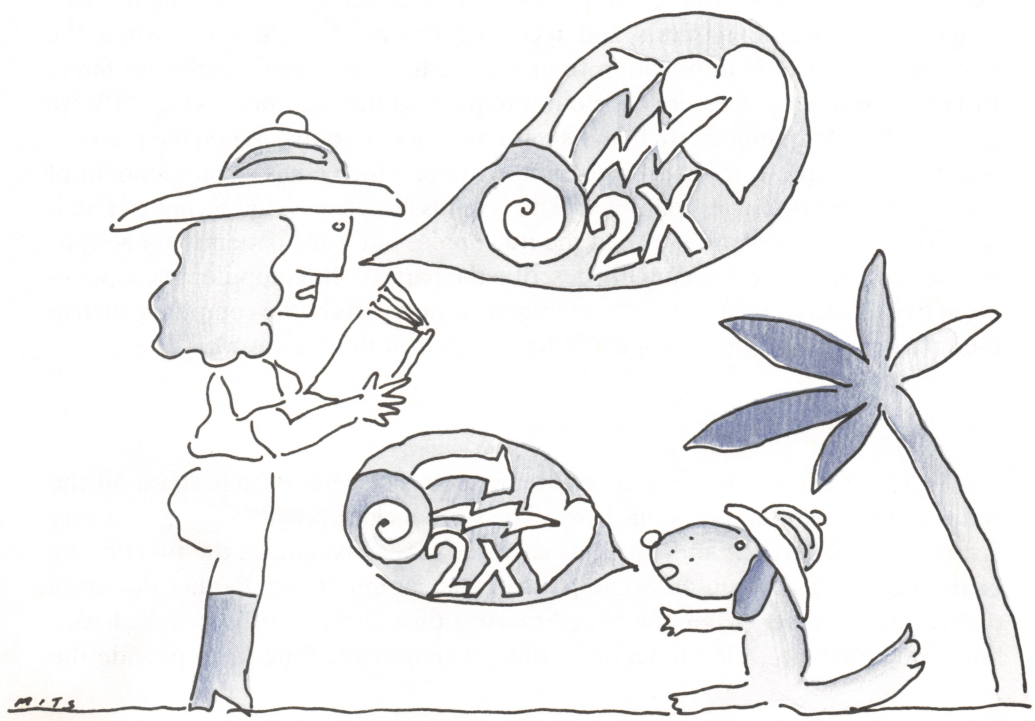
Not all command programs need copying, though. Among those that don't are two of the commands that we've discussed, DIR and COPY, and two others that

we mentioned earlier, DATE and TIME. These commands are called internal, because the programs that do their work are incorporated into DOS. Whenever DOS is in memory, they are in memory. That's handy. Any internal command can be used right away—it doesn't need to be read in from the diskette before it can be put to work. The commands that do have to be read from a diskette are called external, just as you might expect. Those are the ones we have to copy if we want to use them on other diskettes.

You can find out which commands are internal by looking them up in the DOS manual. As you grow more familiar with DOS, you'll quickly learn the internal commands by heart. For now, you already know the most useful ones.

11
UNDERSTANDING THE WORKINGS

In this chapter we'll take a nontechnical look at how the PCjr works. We won't be going into any real depth, but we will try to make the computer seem less of a mysterious "black box."



ELECTRONIC PARTS

The inside of any computer, including our PCjr, consists of a large number of electronic parts, all working together cooperatively. Most of these parts are more or less passive and dumb—they just perform some straightforward function; for example, the memory simply holds data, accepting new information and releasing old information.

But a surprising number of the computer's electronic components are both smart and active. The "brain" of the computer, the Intel 8088 microprocessor, is very smart and very active in controlling the actions of the rest of the computer. But it isn't the only part that fills this active role. There are several programmable electronic components—programmable in the sense that they vary their activity according to program instructions sent to them by other components. Among these programmable parts are the PCjr's timer (which you'll meet in a few moments) and its sound-producing chip, which is one of the extra features our Junior has that its bigger brothers don't. The sound chip accepts commands that tell it when to produce a sound and how loud to make it.

Teamwork and Talk

Not only do the parts of our computer work together cooperatively, they go far beyond the blind cooperation that the parts of ordinary machines could be said to exhibit. The various parts of our computer talk to each other over their wired connections, making requests and receiving replies. For example, when the microprocessor needs information from the memory, it doesn't use brute force. Instead, the microprocessor sends out a request to the memory saying, "Please give me the information in such-and-such a memory location." When the memory receives this request, it looks up the information, which takes a short amount of time (very short by our standards), and then sends back a message saying, "Here it is." While the memory is working, the microprocessor waits patiently for a reply. So you see, we can very accurately describe the parts of the computer as cooperating actively; and actually, even the seemingly dumb parts of the computer, such as the memory, don't play a completely passive part in this teamwork.

Our Junior's Heartbeat: The System Clock

An electronic metronome called the system clock helps orchestrate all the activity of the computer's various parts. The system clock provides a beat at a very fast tempo—nearly five million beats each second—that guides the functioning of the rest of the computer. Some parts of the computer use another device, a programmable timer, when they need to use a time signal that beats at a slower, but still regular rate. One function of this programmable timer is to provide the

microprocessor with a clock that is used to keep track of the time of day. Together, all the clock operations, fast and slow, serve as the computer's heartbeat.

Our Junior's Brain: The Microprocessor

The microprocessor controls nearly everything in our PCjr. It does two main kinds of work: One is executing the programs we ask the computer to perform; the other is computer housekeeping. This housekeeping involves such details as telling the other smart components what to do. It is the microprocessor, for example, that figures out just what signals the sound-generating chip needs to produce the right sounds.

That is a very rough idea of how the parts of the computer work together. Next, we'll take a look at just how our computer "thinks."

COMPUTER THOUGHT

In order to "think," a computer needs a program to tell it what to think about. Just how does a computer carry out a program? We'll use a short example to show roughly how the process goes. Let's suppose we have a BASIC program that includes this statement:

```
LET A = B - 17
```

In our programming language, BASIC, this statement, or instruction, means "take the value that is stored in the memory location for the variable we call B, then subtract 17 from that value, and finally store the result in the memory location for the variable we call A." But a computer doesn't actually carry out whole statements such as:

```
LET A = B - 17
```

That is, it doesn't look at the statement as we would and say to itself, "Hmm, B equals 20, and 20 minus 17 equals 3, so A equals 3." Instead, as you saw in Chapter 8, the computer can only perform machine-language instructions; so one way or another, our statement must be translated from a single instruction in BASIC into a series of small, do-able steps in the computer's own machine language. To see what these instructions are like, let's step through some assembly-language instructions that, as you'll recall, are very close to machine-language instructions, but in a form we can understand more easily. Since we've talked about assembly language, you might have been wondering what it looks like; this gives you a very short sample of it. The assembly-language instructions for our BASIC statement would go something like this:

```
MOV AX,[B]
SUB AX,17
MOV [A],AX
```

The first instruction, MOV AX,[B], takes the memory location (technically known as the address) for the variable B, reads the value out of that memory location, and MOVes the numeric value into a special kind of memory location, known as a register, called AX. Registers are commonly used by computer programs as temporary workplaces to hold information.

The second instruction, SUB AX,17, SUBtracts the numeric value 17 from the value B that's now held in register AX and replaces B in register AX with the new value, B − 17. The number 17 is actually a part of the SUB instruction; that's because some computer instructions, such as this subtract instruction, can use a number directly instead of having to fetch and use a number that's stored somewhere else in the computer's memory.

The third instruction, MOV [A],AX, then MOVes the result of the subtraction from the AX register into the memory location that has been set aside by the computer for the variable A.

What you've seen here is a brief sample of how the 8088 microprocessor actually carries out our programs. As you can see, the details of what the microprocessor has to do to accomplish anything we ask it to do are very lengthy and tedious. This is why almost everyone who writes programs for the PCjr writes them in high-level programming languages, such as BASIC. But no matter how our programs are written, they always end up being broken down into small, simple steps as in our preceding example.

EXCUSE ME: INTERRUPTS

One of the most important things that helps make a computer work so smoothly is called an interrupt. Even if we understand just a little about how interrupts work, we can get a very good idea of how a computer manages to function so cleverly and efficiently.

The Computer's Conflict

A computer has two conflicting needs to serve: First, it has some work to do, and it should spend as much time as possible doing it. Second, the computer needs to respond to such things as our pressing keys on the keyboard or pushing the trigger on a game joystick.

The computer could spend its time electronically watching the keyboard and other similar parts, checking to see whether anything had happened that needed attention. But if the computer worked this way, it would have little or no time left to get its work done. On the other hand, if the computer went about its business and just tried to inspect the keyboard and other parts periodically, it might leave us waiting too long before it got around to paying attention to us. That's the essence of

the computer's conflict: The computer should respond immediately to any events in its peripheral devices—such as the keyboard and diskette drive—yet the computer should also be free to carry out the work requested by our programs when there isn't anything needing attention.

The Interrupt Solution

The solution to this conflict is an interrupt. The 8088 microprocessor brain inside our Junior is designed, like all modern computer processors, to use interrupts. The idea of an interrupt is very simple, although lots of sophisticated effort goes into making it work. Basically, here's how an interrupt works: Our microprocessor works busily away until something happens, such as the pressing of a key, that needs the microprocessor's attention.

In effect, when we press a key, we're saying to the computer, "Excuse me." Our microprocessor immediately puts down the work it was doing and takes care of whatever the interrupt calls for. Then, when the interrupt has been serviced, as it's known in technical computer talk, the microprocessor goes back to the work that it was doing.

Each time we press a key, the microprocessor is interrupted. That may seem like a lot of interruptions, particularly if we're typing away at full speed. The remarkable thing is, though, that the computer is very efficient at handling interrupts, and dealing with our keystrokes hardly fazes it at all.

There are many kinds of interrupts that come to the microprocessor. One, called the clock tick, comes from the programmable timer we mentioned earlier and is used to enable the computer to keep track of the passing time of day. Clock-tick interrupts happen about 20 times a second, yet dealing with them puts hardly any load at all on the microprocessor's ability to get work done. In comparison to the ticking of the clock, any typing that we do on the keyboard is a delightfully rare distraction.

Explaining the Freeze

Knowing how the computer uses interrupts to manage its work is mostly a matter of satisfying intellectual curiosity, but an understanding of interrupts solves one mystery that puzzles many IBM personal computer users. Occasionally, when we are using our computer, it will seem to freeze up, and will not respond to anything we do.

How can this happen? As it turns out, programs sometimes have to lock out, or suspend, interrupts; when they do, they only need to suspend interrupts for a very short time, usually less than one-thousandth of a second. But, as we should all know, programs do have errors, or bugs, in them. The parts of programs that suspend interrupts are normally carefully checked and corrected—debugged—

so that they are completely error free. But, with just the right combination of bizarre circumstances, a program will sometimes unintentionally disable the interrupts or make them otherwise unusable. When this happens, the computer freezes up, and the only remedy is to turn it off and restart it.

VIDEO TRICKS

One of the most interesting things about the PCjr is how the display screen works. Much of this subject is quite technical, but here's a quick summary.

Information appears on the display screen and, like any other data, that information has to be kept somewhere. You might think it is kept inside the display unit, but that isn't the way the PCjr's display works. For one thing, the PCjr can use a TV set as its display, and a TV doesn't have any memory to hold display information. Instead, our Junior sets aside a section of its main memory to hold the information that appears on the screen.

Writing on the Screen

When our PCjr's programs want to "write" information onto the screen, they just place the information in the part of memory that is set aside for the display screen. Inside the PCjr's system unit is all the circuitry needed to make our display work. This video circuitry constantly reads the display information out of memory and converts it into the signals needed by the display screen, whether it's a TV set or a special computer monitor.

There is an interesting way we can demonstrate how this process works. Figure 11-1 shows a BASIC program that moves data directly into memory locations that are set aside for the display screen.

```
1000 ' Demonstrate how placing data into memory
1010 ' makes it appear on the display screen
1020 ' from Discovering the IBM PCjr Home Computer
1030 ' authored by Peter Norton, 1984
1040 '
1050 GOSUB 2000   ' initialize
1060 FOR INDEX = 1 TO LEN(MESSAGE$)
1070   GOSUB 3000 ' display the next character
1080 NEXT INDEX
1090 GOSUB 4000   ' finish up
1100 END
```

(continued)

Figure 11-1. Program listing

```
2000 '
2010 ' initialization
2020 '
2030 KEY OFF
2040 SCREEN 0, 1
2050 WIDTH 40
2060 COLOR 14, 1, 1
2070 CLS
2080 DEF SEG =  &HB800 ' the right memory area
2090 PRINT
2100 PRINT " This demonstrates how placing"
2110 PRINT " data into memory makes it appear"
2120 PRINT " on the screen."
2130 PRINT
2140 PRINT " Press the space bar to make each"
2150 PRINT " letter appear below"
2160 MESSAGE$ = "POKE puts data into memory."
2170 LOCATE 15, 1, 1
2180 RETURN
3000 '
3010 ' subroutine to display each character
3020 '
3025 ' the next line gets the next character
3030 CHARACTER = ASC(MID$(MESSAGE$,INDEX,1))
3035 ' next line finds where in memory
3040 OFFSET = 800 + INDEX * 2
3045 ' next line sets color, red on blue
3050 POKE OFFSET + 1, 28
3055 ' next line puts the character into memory
3060 POKE OFFSET, CHARACTER
3065 ' next wait for a keystroke, except at spaces
3070 IF CHARACTER = 32 THEN RETURN ' 32 is space
3080 IF LEN (INKEY$) = 0 GOTO 3080
3090 RETURN
4000 '
4010 ' finish-up subroutine
4020 '
4030 PRINT " Returning to BASIC"
4040 PRINT
4050 RETURN
```

Figure 11-1. Program listing (continued)

In the program, line 3060 uses the POKE statement of BASIC to place each character of the message in line 2160 directly into the display memory. If you key in and run this program, you'll see a message appear on the screen, even though

nothing in the program actually "writes" it; everything else that appears from the program is written on the screen in the ordinary way, with PRINT statements.

The memory that is used by the display screen controls not only what is shown, but how it is shown in color. While line 3060 POKEs the characters into memory, line 3050 POKEs the color that is to be used; we've made the color red on blue (color code 28 in memory). If you experiment with changing the number in line 3050, you'll see other colors; for example, 135 will make blinking white on a black background.

Two screen modes: Text and graphics

There are actually two completely different ways that the display memory can be used to control what is shown on the screen. One is called *text mode* and the other is called *graphics mode.* In BASIC, the mode part of the SCREEN statement controls this; mode setting 0 gives us text mode, and any other mode number, 1-6, gives us a graphics mode. (You can learn about all the details of the SCREEN statement in the manual that comes with BASIC.)

In text mode, numeric codes that correspond to characters are placed in memory, and the video circuitry performs the magic needed to make the right character shape appear on the display screen.

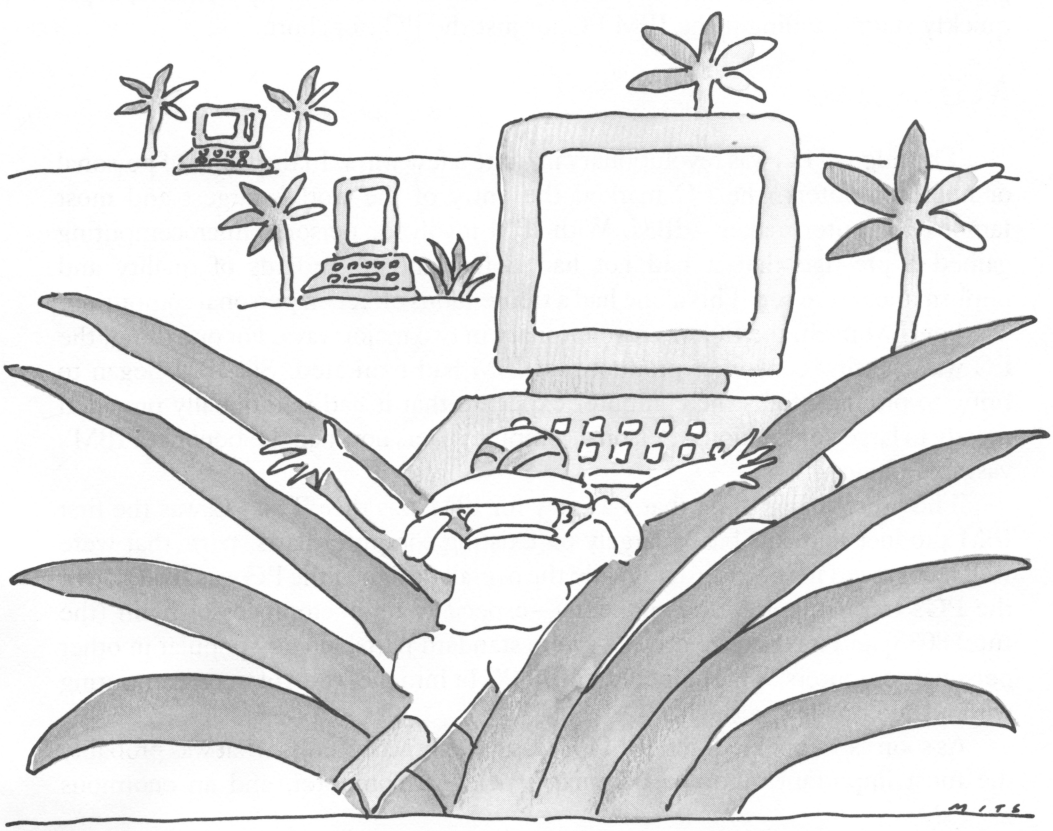
In graphics mode, the screen is set up for drawing pictures. In this case, the screen is divided into a great many little dots; the computer draws pictures by lighting up the right dots needed for each picture. When we see text characters appearing on the screen in graphics mode, they appear because programs have actually created drawings of the characters in memory. The method used to do this is technically fascinating and quite clever; but to us using the computer, characters in graphics mode appear on the screen just as easily as they do in text mode.

There is, of course, a whole lot more to learn about how computers and display screens work, but what we've covered here should be enough for a basic understanding of the computer and how it shows us what we see.

The goal of this chapter has been partly to explain some of the fundamentals of how the computer works, and partly to whet your appetite for more. If you want to go further in understanding the Junior's inner workings, see my companion book, *Exploring the IBM PCjr Home Computer.*

12
OUR JUNIOR AND THE REST OF THE FAMILY

One of the most important things about the PCjr is that it is one member of a family of computers, the IBM personal computers.



FAMILY CONNECTIONS

We've been seeing throughout this book how important the PCjr's family relationship is—in terms of available software for our computer and our ability to move on, if we want, to a more powerful, yet very similar, computer. Since our Junior is just one member of a compatible family of computers, most of the knowledge and skills that we gain on one model can be used on another. Now, it's time to learn the history of the family and to compare our Junior with its bigger brothers—to see what it has that they don't, and vice versa.

We'll begin with the IBM personal computer family history. I'll also stir in the story of my own involvement with these computers. It will tell you how this book came to be written, and it will show you one way a personal computer hobby can become a personal computer career.

FAMILY HISTORY

The public story began on August 12, 1981, when IBM announced its very first personal computer. IBM officially called it the IBM Personal Computer, but people quickly started calling it the IBM PC, or just the PC, for short.

A PC Revolution

The original PC was revolutionary in quite a few ways. In the world of personal or microcomputers, the PC marked the entry of the world's largest and most famous computer maker—IBM. With IBM involved, personal microcomputing gained a prestige that it had not had before; new standards of quality and performance were set. This alone had a tremendous effect on personal computing.

For IBM itself, the PC was revolutionary in two major ways. For one thing, the PC was the first consumer product that IBM had marketed. So, IBM began to bring to ordinary folks the computer expertise that it had traditionally provided mostly to large corporations. Personal computer users now had the benefit of IBM's vast experience.

The other major thing that was new for IBM was this: The PC was the first IBM product that was based largely on existing computer parts, parts that were well recognized in the industry. While the overall design of the PC was IBM's own, the PC's most important components—especially its microprocessor brain (the Intel 8088) and its diskette drives—were standard parts, already popular in other personal computers. This helped the IBM PC fit into the world of microcomputing without too much of a shock.

As soon as it appeared, the IBM PC was quickly accepted as what was probably the most important, and the dominant, personal computer, and an enormous

number of computer programs were written for the PC. The focus of attention in the personal computer marketplace shifted to the PC. Many computer enthusiasts felt that the IBM PC was "where the action was," and so a great many programs and hardware enhancements were developed for the PC. It was the beginning of an exciting new world.

I, personally, was caught up in that excitement, and I bought my PC in the first few weeks after its announcement, even though I had no idea what I would do with it. Over the space of a few months, I developed some programs to help me explore and discover more about my PC and I began writing a few articles to share the knowledge that I'd gained in playing with my wonderful new computer.

The programs grew into a set I called the Norton Utilities™. Because they helped people explore little-known parts of the computer, and because they helped people recover data they would otherwise have lost from their diskettes, my programs became popular and sold well. This is how it has often been with personal computer programs: Someone has an idea for an interesting program; if the idea is new, and the purpose is useful or fun, and the program is well-written, then the program can become a success. Perhaps you, too, as you tinker with your PCjr, will have a wonderful idea for a program that no one has had before.

I also contributed the articles I wrote about the IBM PC to the newsletters of computer clubs and users' groups. Some of them I published in tip sheets and gave out by the thousands, to help PC beginners get started with their computers. Soon, I was contributing articles to several PC-related magazines, and then I was invited to write a book telling all I know about the PC.

Adding the XT

In March 1983, IBM introduced the XT model of personal computer. While the original PC filled the needs of many personal computer users, people who used it heavily found that it lacked one important feature: high-capacity disk storage, where lots and lots of information and programs could be stored.

The Personal Computer XT, as it was officially called, incorporated a built-in, high-capacity storage disk, that could hold over ten million characters of data; in technical talk it had a ten megabyte (that is, ten million-byte) disk. IBM's term for this new storage device was fixed disk, since it can't be removed and replaced the way diskettes can. We usually call it a hard disk, since it isn't flexible, like a floppy diskette, or a Winchester disk, since the technology used for this disk was nicknamed Winchester.

The importance of the XT

The XT, as we all quickly came to call it, had two very important features. One, of course, was the storage capacity of its fixed disk. The other was its

compatibility with the original IBM PC, which meant that all the programs and skills people developed and used on the PC could be transferred to the XT.

The XT, then, became the PC's big brother, and IBM now didn't just have a single personal computer; it had a family of two. This family relationship was extremely important, because, up till then, most personal computers had not been compatible with any other models. Too often skills learned, programs written, and data collected could be used with one computer, but not with others. For the IBM family of personal computers, there was no such problem.

Helping the pioneer

To make sure that existing PCs weren't cut off from the benefits of a ten megabyte fixed disk like the XT's, IBM produced an expansion unit that could be attached to a PC. The expansion unit included the XT's fixed disk, so that an original PC, like mine, could be brought up to the same capabilities as the XT. The expansion unit could also be used to add a second ten megabyte disk, or any other standard hardware options, to either a PC or an XT.

The expansion unit was a welcome addition for me. With all the program development for my Norton Utilities, with dozens of articles and two books written, I really needed storage space. So, I upgraded my PC to the equivalent of an XT with an expansion unit. That helped a lot. The expansion unit enabled me to keep all my programs in one place, and made it easier to keep track of everything that I had written. The high capacity of the fixed disk even made it possible for me to make program improvements that actually wouldn't have been possible if I'd had to store the programs on diskettes. Many other personal computer users had similar experiences, finding that a fixed disk made some work easier, and made other work possible for the first time.

Most of this book was written on my PC, and if you were to inspect the contents of the fixed disk in my PC's expansion unit, you would find the very words you are reading now.

The PCjr's Appearance

While the PC and XT filled the needs of many personal computer users, these two models were too expensive for many who wanted a personal computer, because they were oriented toward professional, rather than home use, and they offered more computing power than many people needed. In addition, the PC and the XT don't have certain features, such as very sophisticated sound and graphics, that are needed in computers designed for home and educational use.

And so, IBM introduced the PCjr on November 1, 1983, giving us an inexpensive computer with PC-compatibility and lots of extra features as well. PCjr is the official name of this new computer, but many of us just like to call it Junior. By

the way, it's interesting to note that the very first PC was titled, in full, the IBM Personal Computer, and that IBM at first very much resisted its popular name, PC. But by the time our Junior was announced, IBM had decided to join the rest of us in using informal names for these wonderful computers.

Because it is so much more affordable than the PC and the XT, the PCjr has made it possible for a lot of people to have an IBM personal computer for the first time. Now, many schools and homes can afford IBM personal computers; more people who need computers in their work can get them, and PC and XT users can expand their use of IBM computers to more places, including their homes.

On the day the PCjr was announced, I paid for two of them—and impatiently waited for the day I could get them in hand. One was for my office, for help in writing this book and for fun, and the other was for my home, for even more fun.

In this book, I've told you the main things to know about the PCjr, and shortly I'll outline the differences between our Junior and the PC and XT models. But before I do that, let's take a look at the PCjr's distant relatives in the IBM personal computer family.

Great Aunts and Uncles

The main part of our PCjr's family tree consists of the original PC, the XT, and our own Junior. But there are other members in the family that you might be interested in knowing about. These other family members are like great aunts and uncles in two ways: They are more distantly related to the PCjr than are the PC and XT, and they are bigger, more powerful computers, so they are "great" in that sense of the word, too.

We'll mention three of these distant relatives. These family members are far removed from the world of our PCjr, because they are all oriented toward the top of the computing scale—bringing mainframe and personal computing together. Yet, each member of our IBM personal computer family, including these great aunts and uncles, can run practically all the programs that have been written for the entire family.

The 3270 Personal Computer adapter

The first of these great aunts and uncles appeared in March 1983, at the same time as the XT; it is the IBM 3270™ Personal Computer adapter. Most of IBM's large mainframe computers use computer display terminals known as the IBM 3270-series. Hundreds of thousands of these 3270 terminals are in use, connected to large IBM computers. The IBM 3270 Personal Computer adapter attaches to one of these 3270 terminals and allows it to act both as a PC and as an ordinary 3270 terminal. The IBM 3270 Personal Computer adapter, then, brings the flexibility of PC-style computing power to people who already have 3270 computer terminals.

The 3270-PC

While the IBM 3270 Personal Computer adapter converts an existing 3270 terminal into a part-time PC, another member of the IBM personal computer family takes the same idea much further. This computer is called the 3270-PC™. Although the name 3270-PC is very easy to confuse with the name of the personal computer adapter, the 3270-PC is a complete computer in itself, while the adapter attaches to a separate 3270 terminal. Like the adapter, the 3270-PC can be used as a terminal, allowing the user to interact with a mainframe computer. But a 3270-PC also does many fancy things that no other computer or terminal does. Among the 3270-PC's tricks is the ability to keep seven different operations going at one time; some of these operations are PC programs and others are 3270 display work.

The XT/370

The most exotic member of the IBM personal computer family is the XT/370™. The XT/370 starts out with the ordinary abilities of an XT personal

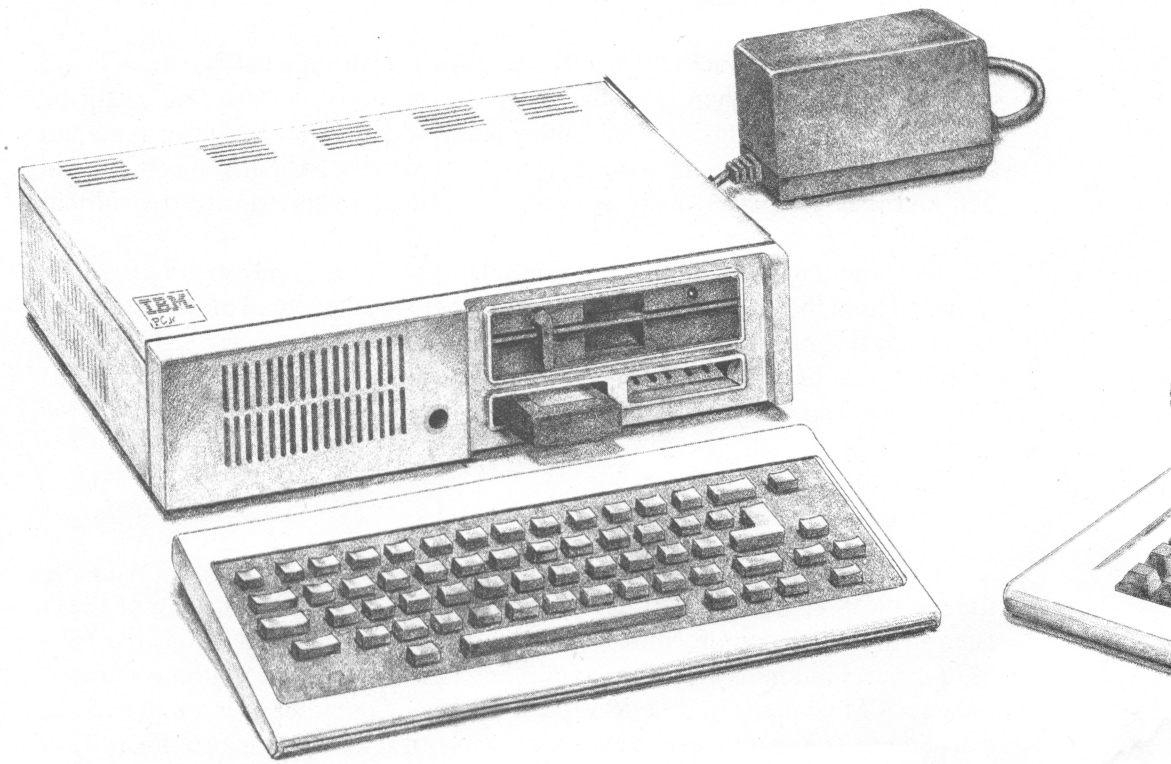


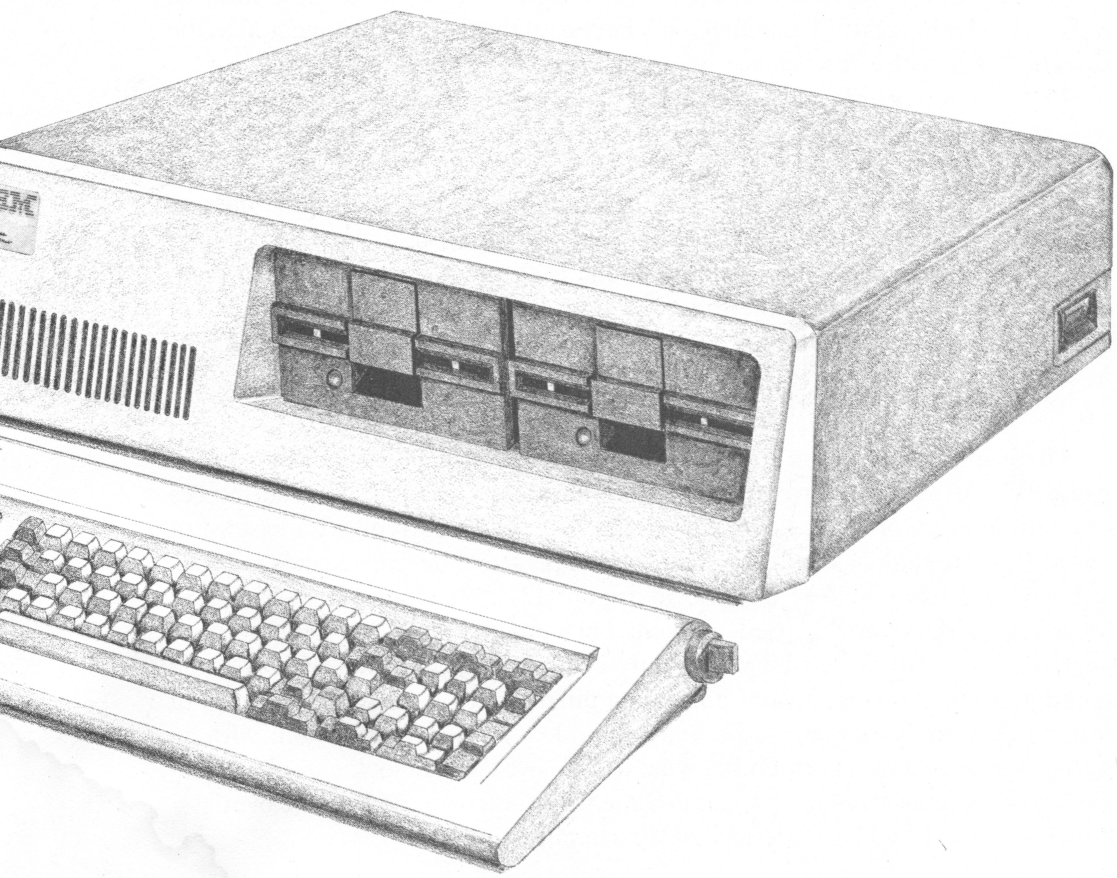
Figure 12-1. The PCjr and its bigger brother, the PC

computer, and then adds the ability to carry out programs copied from IBM's 370 line of mainframe computers. The XT/370 actually puts mainframe computing inside an XT. While the 3270 Personal Computer adapter and the 3270-PC can connect to a mainframe, to tap into the big computer's work, this XT/370 can actually do the mainframe's work once it has borrowed a copy of one of the mainframe programs.

After this short look at the exotic members of the family, let's now compare our PCjr's features with the features of the main family members, the PC and the XT. You can see the PCjr and a PC in Figure 12-1.

WHAT OUR JUNIOR HAS THAT THEY DON'T

As we've said over and over again, one of the most important things about the PCjr is its relationship to the other IBM personal computers, and its ability to do most of the things the others can do. We've gone over the whys and wherefores in



various places throughout these chapters, but it would be good to bring them all together here, in one place. So in the next part of this chapter, we'll summarize the similarities and differences and explain again why our Junior's family relationship is so important.

Let's begin with what the PCjr has that the other IBM personal computers don't have, or don't normally have.

Cartridges

One of the most obvious things we have with the PCjr is the ability to use software cartridges. The other models of IBM personal computer, including both the original PC and the XT, can't use cartridges unless they are modified with a special cartridge adapter; they rely almost completely on diskettes for the programs they use.

A Different Keyboard

Another thing we have, which is quite different, is our 62-key, infrared keyboard. The PC and XT use an 83-key keyboard that is always connected to the computer by a cable. You can see both keyboards in Figure 12-2.

The PCjr keyboard can be connected to our Junior's system unit by a cable if we wish or, as we have seen, it can use the infrared light link to send its signals to the computer.

Besides the infrared light link, our keyboard also has fewer keys; the missing keys are replaced by combinations of keystrokes on the PCjr keyboard, and these combinations serve the same functions as the PC's and XT's extra keys. Essentially, then, we can do the same things on both keyboards—we just do some of them differently with our Junior.

Keyboard overlays

There is also one more thing that is different about our Junior's keyboard layout. The distances between our Junior's keys are the same as the distances between the keys on a PC keyboard (so that we can touch-type with ease), but Junior's keys are slightly smaller than the PC's. This size difference makes room for keyboard overlays, such as the one in Figure 12-3, that fit around our keys. Keyboard overlays can be used to point out certain special keys that various programs use, so they can be very handy in reminding us when or how to do something. For example, a word processing program might be set up to save our work on diskette whenever we press a certain key or key combination; with an overlay, we don't have to remember what to do—we can just look at the explanations on the keyboard. Some programs come with their own ready-made overlays, but we can also buy blank overlays so we can tailor our own.

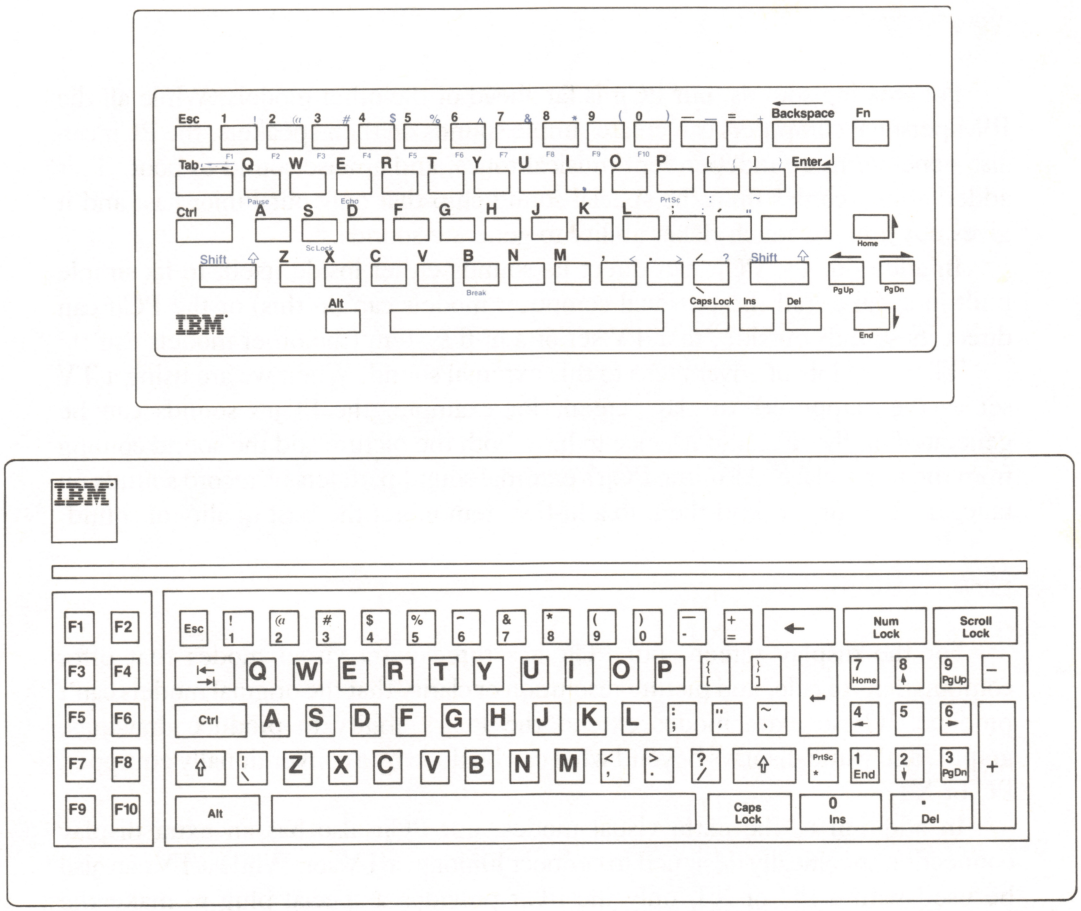


Figure 12-2. The PCjr and the PC/XT keyboards

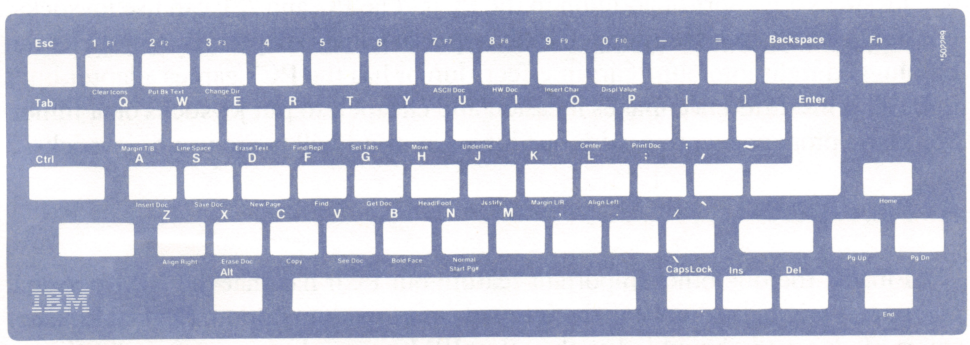


Figure 12-3. An example of a keyboard overlay

New Sounds

For creating sounds, our PCjr is far ahead of the other models. While all the IBM personal computers can create simple sounds on their speakers, the PCjr can also generate three independent musical tones, and a noise sound to boot. This added ability comes from the special sound chip that only our Junior has, and it gives our Junior a much richer ability to generate sounds.

In addition, the PCjr can direct its sounds either inside itself, to its simple built-in speaker (all the personal computer models can do this) or the PCjr can direct its sounds outside, to a TV set or a hi-fi system (the other models can't).

There are lots of advantages to this external sound. When we are using a TV set as the computer's display screen, for example, the PCjr's sounds can be generated on the TV; and it's nice to have both the picture and the sound coming from the same place. Also, the PCjr's external sound path lets us record sounds on tape, if we want, or send them to a hi-fi system to get the best quality of sound.

New Pictures

For the display screen, our PCjr has three extra visual modes that offer combinations of color and picture resolution or clarity that the original models can't produce. These extra modes extend the PCjr's ability to produce drawings, graphs, and game displays beyond what can be done by a conventionally equipped PC or XT.

In addition to the extra visual modes, our PCjr also has an extra display connection specifically designed to connect Junior to a TV set. While a TV can also be used with a PC or XT, only the PCjr provides a special plug to make the connection easier.

More Joy

Joysticks are a sort-of addition to the PCjr. The PC and XT can use joysticks, but only if they are also equipped with a special games adapter. But joysticks can be plugged right into our PCjr; in effect, Junior has the PC's games adapter built right in. This difference makes it easier, and cheaper, to put joysticks on a Junior. So, many programs written with our Junior in mind will make use of joysticks.

A Smart Modem

Finally, the one other important feature our PCjr has that the IBM personal computers originally didn't, is the internal smart modem option. Several kinds of smart modems can be added to the other IBM personal computers, but only our PCjr has one specially designed to fit inside it.

These are the main items our PCjr has that the other models either can't have, or usually don't have. Next, let's look at what they have that our Junior doesn't.

WHAT THEY HAVE THAT OUR JUNIOR DOESN'T

There are two main things that our PCjr has given up in order to be as inexpensive as it is: memory capacity and disk capacity.

Memory Capacity

Our PCjr has a general limit of 128K of working memory space, or RAM; more memory can be added, but our Junior isn't really designed to have more. The other models of IBM personal computer can be expanded to have up to 640K of memory, or five times what our PCjr can accommodate. Does this difference place a serious limitation on what we can use our PCjr for? Not really.

Some programs do require more memory than our PCjr can provide, and there are special uses for extra memory—besides its ordinary use for running programs—that our PCjr must forgo. But the mainstream of programs for the IBM personal computers can fit into the amount of memory that our PCjr provides. In plain words, then, our Junior has plenty of memory for most purposes.

Disk Capacity

Our PCjr is limited to using only a single diskette at a time, because it has only one diskette drive. The other models commonly have two diskette drives, and can have as many as four, plus the huge storage capacity of a fixed, hard disk system.

What have we given up, in being able to use only one diskette at a time? We've given up two things. First, some convenience. Many things we use our computer for are much easier and more convenient to do when the computer has two diskette drives (or a high-capacity hard disk). When a computer has only a single diskette drive, we have to move diskettes in and out of the drive more often. If we are accustomed to a larger computer, this diskette "swapping" is not only a nuisance, but also can lead to our making more mistakes. Overall, though, our computer's limitation in this respect is a small price to pay for all the advantages of the PCjr, including the advantage of its low price.

There is one other consideration we need to bear in mind about our diskette storage capacity: Some uses for a computer simply require more disk storage space than the PCjr can provide. While most things we'll do with the computer can be accommodated by changing diskettes, some tasks require large sets of records or data and can't be satisfied this way—they have to have more space, and all at once. The most common examples of this storage requirement include the development

of large programs, and the needs of complex business accounting (lots of accounting tasks can work well with a single diskette, but at some point, some jobs just need more space).

Everyone knows that if you need a moving van, you don't use a compact car for the job. So no one should be surprised that some computer tasks just exceed the disk storage capacity that our PCjr can provide. For heavy business and professional use, it's likely that the PCjr just won't be big enough to handle the load. There is nothing remarkable in that. What is remarkable is how large a proportion of the computer work for a PC or an XT can be done by our modest little PCjr.

Expansion Capacity

There is one more important difference between our PCjr and the other IBM personal computers: How they can be expanded. The other models have space inside for five or more options to be plugged into a general-purpose connection called an expansion bus. This expansion bus is designed to accommodate all sorts of options and new equipment, including memory expansion units and lots more.

Our PCjr doesn't have the five expansion slots, although it does have the general-purpose bus. On the PCjr, the expansion bus takes the form of the I/O channel connector, located on the right-hand side of the system unit. This connection can be used more or less like the PC's expansion bus; more than one option can be plugged into it in a kind of "chain"—as long as each option has both an "in" and an "out" plug to continue the original connection on down the line.

In short, the PC and XT provide an electronic accommodation for adding options to the computer. They also provide a physical place to put options; the options plug into the computers' internal expansion slots. Our PCjr also makes an electronic accommodation for options with its I/O connector; the form differs, but the purpose is the same. Our Junior, however, differs from the PC and XT because it doesn't provide a physical place to house any options other than the standard memory, diskette drive, and modem options.

What this difference means in practical terms is twofold. First, none of the expansion options for the PC and XT can be plugged into our PCjr, because the design of the expansion connection is different. Second, any options that are added to the Junior need to have their own case or housing; other than the memory, diskette drive, and internal smart modem, our Junior's options are external to the system unit.

Keys

There are also some minor things the other IBM personal computers have that our Junior doesn't. One we've already mentioned is 21 more keys on the keyboard. The standard keyboard used by the PC and XT has 83 keys to our PCjr's 62.

These extra keys make it more convenient for us to control what we are doing with programs and the display. Functionally, however, our PCjr can duplicate the action of any of the PC's keys by using combinations of keys; for example, where we might press the special key labeled F1 on the PC keyboard, we would press two keys, Fn followed by 1, on our PCjr's keyboard. (F or Fn, by the way, stands for Function and indicates a special key a program can use to tell our Junior, "When this key is pressed, do such and so"—for example, underline a word, go to the end of a file, or whatever.)

In many ways the keyboard differences are a lot like our having to shuffle diskettes in and out of the PCjr's diskette drive: The PCjr is a little less convenient than the the PC, but it will still do everything the PC does.

Monochrome Display

Another, and very minor thing that the other IBM personal computers have that our PCjr doesn't is the ability to use the IBM Monochrome Display™, which is unusually clear and sharp and nice to work with. However, we can get nearly the same result by using a non-IBM monochrome composite monitor with our PCjr. A monochrome composite monitor can produce a picture very much like IBM's Monochrome Display does, although the characters displayed on the screen won't be quite as nicely shaped. That's hardly anything to worry about.

LOOKING TO THE FUTURE

One of the most important things about the IBM PCjr, and one of the main things setting it apart from the usual run of home computers, is the fact that it is a machine with a pedigree and a future. The pedigree is its family connection with the other IBM personal computers. Its future is based on its family connection, because it is part of a series of computers that have come to dominate the world of personal computing. While other home computers—especially game computers—may wither simply because they stand in isolation, without the support of a computer family, our IBM PCjr has a long life ahead of it.

Our Junior's future has some important practical meanings for us. In the past, many home computers were like (in fact, they often were) Christmas toys that were used for awhile, and then put away, never to be seen again. The limited abilities of many home computers encouraged us to treat them as toys. And even if a home computer performed well, when it was outgrown—when we needed a bigger computer—we had to throw away the programs and skills we acquired, along with the computer we had outgrown.

The PCjr, though, is so rich a home computer, we can plan to use it for some time to come. Even more important, if our computing needs outgrow what the

PCjr can do for us, we can carry our programs and our skills over to other IBM models, like the PC and the XT.

Because of this open future, as wise PCjr owners we should approach the computer in a special way. First, we ought to think of the things we need to learn in order to use our PCjr, and the skills that we acquire in using it, as a true, valuable education—a permanent addition to our knowledge and our abilities. With this in mind, we have good reason to treat what we learn with and on our PCjr seriously, and not think of it all as a passing fancy. After all, becoming proficient with the PCjr is an investment in our continuing education.

There is another thing to consider when we think about our future with the PCjr and other computers. It is quite possible that we will outgrow the PCjr itself, but it isn't likely that we will outgrow the abilities of the full range of IBM personal computers. With this consideration in mind, we would be very wise to make sure that we aren't limiting the value of our investment in the PCjr. When we buy or develop our own programs, we should make sure that they aren't tied to the specific features of the PCjr; we should make sure that our programs can also be used on the other IBM personal computers. After all, if we spend a few hundred dollars on some fancy program, we shouldn't have to throw it away when we get our next computer. In addition, when we buy expensive accessories, such as printers, we should make sure that we'll be able to use them with other computers.

The message here is clear. We all know that computers in general are important to our future. In our Junior, we have a modestly priced home computer that needn't be treated as a throw-away amusement. With this in mind, let's equip our computer and ourselves for this exciting and expanding future—a future that can begin, for us, with the IBM PCjr.

INDEX

A

A-audio socket, 20
accounting programs, 66
Ada, 86
application software, 4, 13
ASCII, 8
assembler, 11
assembly language, 11, 99, 145

B

BASIC, 40, 86, 95, 97, 104
BASIC language cartridge, 48, 104
batteries for the keyboard, 35
baud, 75, 79
BIOS, 9
bit, 7
buffer, 26
bulletin board, 81
Bumble Games program, 55
byte, 7

C

C programming language, 86, 101
cartridge, 6, 40, 48
cartridge slots, 20
cartridges, 158
character, 7
character strings, 87
CHASM, 100
clock, system, 144
COBOL, 86
color composite display, 29
command, 6, 122
command interpreter, 131
COMMAND.COM, 131
communications program, 76
Compact Printer, 24, 64
compiler, 12, 95
composite video, 29, 35
CompuServe, 82
COPY command, 129, 138
Ctrl-Alt-Del key combination, 38, 41, 120
Ctrl-Alt-Ins key combination, 39

D

D-direct drive socket, 21
daisy-wheel printer, 26
data, 5, 87
database program, 71
data bits, 80
dBase II, 72
density, 15
diagnostics, 10, 39
DIR command, 133
direct mode, 105
Disk Operating System, 10, 119
DISKCOPY command, 138
diskette, 13, 49, 120, 125, 126
diskette drive, 14
display, 28, 35
  composite video, 29
  RGB, 29
  TV, 28
DOS, 10, 41, 48, 119
dot-matrix printer, 26
double precision, 88, 107
double-sided diskette, 15
Dow Jones News/Retrieval, 83
drive, diskette, 14

E

editor program, 62
EDIX, 95
emulator, terminal, 76
Exploring the IBM PCjr, 150
extension, 132

F

file, 16
file name, 16, 131
floating-point numbers, 88
format, 16
FORMAT command, 125, 127
FORTH, 86, 101
FORTRAN, 86

G

games, 43
graphics mode, 150
Graphics Printer, 25
Guide to Operations, 10, 19, 36, 37, 39

H

hardware, 3
high-level language, 12
Home Budget Jr, 66
HomeWord, 64

I

I/O, 91
I/O channel connector, 22
income tax program, 69
indirect mode, 105
input/output, 90, 91
integers, 87, 107
interpreter, 12, 95
interrupt, 146

J

J-joystick socket, 22
joystick, 27, 46
Juggles' Butterfly, 56

K

K (kilobyte), 8
K-keyboard socket, 22
keyboard, 18, 158, 162
Keyboard Adventure, 38
keyboard overlay, 158

L

L-later socket, 22
language, programming, 11
letter-quality printer, 26
light pen, 22
logic, 90
Logo, 86, 101
loops, 92, 108
low-level language, 12
LP-light pen socket, 22

M

M-modem socket, 21
machine language, 11
Macro Assembler, 100
memory, 4-9, 144, 161
microprocessor, 4, 144
MODE command, 137
modem, 27, 74
monochrome composite display, 29
Monster Math, 57
Mouser game cartridge, 50
Multiplan, 68

N

Norton Utilities, 153

O

operating system, 10, 120
operations, 90
overlay for keyboard, 158

P

P-power socket, 20, 34
parallel printer, 24
parity, 79
Pascal, 86, 97
PC model, 152
PC-Talk, 76
PCjr
  entry-level model, 22
  enhanced model, 23
Personal Communications Manager, 76
Personal Editor, 95
PL/I, 86
PMATE, 95
POST, 37
power-on self test, 37
precision, 88
printers, 24
  daisy-wheel, 26
  dot-matrix, 26
processor, 4

program, 2, 86
programming language, 11, 86
prompt, 122

R

RAM, 8
random access memory (RAM), 8
read-only memory (ROM), 8
real numbers, 88
reset operation, 37
RF modulator, 28, 35
RGB display, 21, 29, 36
ROM, 8
RS-232, 78

S

S-serial socket, 21
SBB-Pascal, 98
sectors, 15
serial communications, 78
serial printer, 24
sided, single and double, 15
single precision, 88, 107
single-sided diskette, 15
smart modem, 27, 75
software, 3
software cartridge, 6
software cartridges, 158
sound, 160
SOURCE, 82
spelling checker program, 62
spreadsheet program, 68
stand-alone programs, 40
stop bits, 80
string, 107
strings, 87
subroutine, 109
system clock, 144
system-formatted diskette, 126
system software, 4
system unit, 19, 34

T

T-television socket, 21, 35
template program, 70
TERM, 77
terminal emulator, 76
text, 7
text mode, 150
THE SOURCE, 82
tracks, 15
tractor feed, 27
transformer, 19

V

V-video socket, 21
variable, 88, 106
VEDIT, 95
video, 148
VisiCalc, 68
VolksWriter, 64

W

word, 8
word processing, 60
word processor program, 64
Word Proof, 65

X

XT model, 153
XT/370 model, 156

3270 PC model, 155
3270 Personal Computer Adapter, 155
8088 microprocessor, 144, 152

PETER NORTON

Peter Norton was raised in Seattle, Washington and educated at Reed College in Portland, Oregon. Before discovering microcomputers, Peter spent a dozen years working on mainframes and minicomputers for companies including Boeing and the Jet Propulsion Laboratories. After the debut of the IBM PC, Peter was among the first to buy one. It was then that his exceptional talents for explaining this new machine came to the fore. Now recognized as the principal authority on the IBM PC, Peter is the author of the best-selling *Inside the IBM PC* and creator of the popular Norton Utilities programs which allow the user to manipulate the "insides" of the IBM PC. Peter is currently a featured columnist for both *PC* and *PCjr.* magazines.

The manuscript for this book was prepared on an IBM Personal Computer. Submitted to Microsoft Press in electronic form, text files were processed and formatted using Microsoft Word.

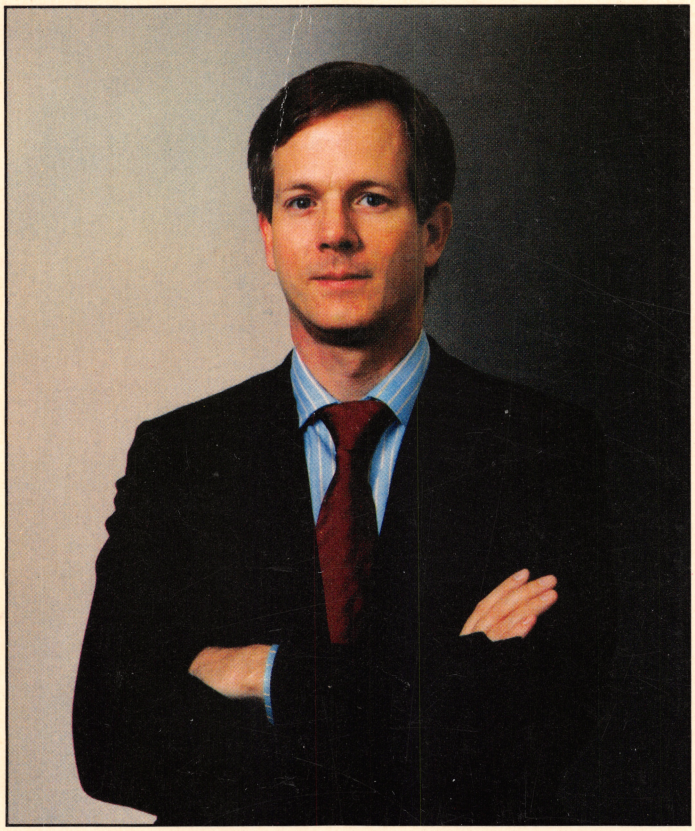
Cover and text design by Ted Mader and Associates; illustrations by Mits Katayama; figures by Melanie Milkie.

Text composition in Caslon 540, with display in Eurostile condensed, using CCI Book and the Mergenthaler Linotron 202 digital phototypesetter.

Cover art separated by Color Masters, Phoenix, Arizona. Text stock, 60 lb. Glatfelter Offset, supplied by Carpenter/Offutt; cover 12 pt. Permalin. Printed and bound by R.R. Donnelley and Sons, Crawfordsville, Indiana.

MICROSOFT® PRESS

Peter Norton



Heard about the new IBM® PCjr™, the home computer that's here to stay? Let Peter Norton ease your entry into the fascinating world of home computing with this beginner's guide to the IBM PCjr. In his clear, non-technical style, Peter leads you through the first steps to getting your PCjr up and running. Then, the mysteries and excitement of computing unfold before you as you learn to create programs and apply the PCjr for home, educational and business uses. In *Discovering the IBM PCjr Home Computer,* an adventure in computing, including sound, graphics and telecommunications, awaits you and your entire family! Peter Norton, author of the best-selling *Inside the IBM PC,* is a featured columnist for *PCjr* magazine and creator of *PC Magazine*'s popular *Norton Chronicles.* Be sure to look for the other books in the Microsoft Press/Peter Norton IBM PCjr Library.

\$15.95
0484-1545
S&S No. 0-671-30828-9

IBM® is a registered trademark and PCjr™ is a trademark of International Business Machines Corporation. Microsoft® is a registered trademark and MS™ is a trademark of Microsoft Corporation.

ISBN 0-914845-01-2