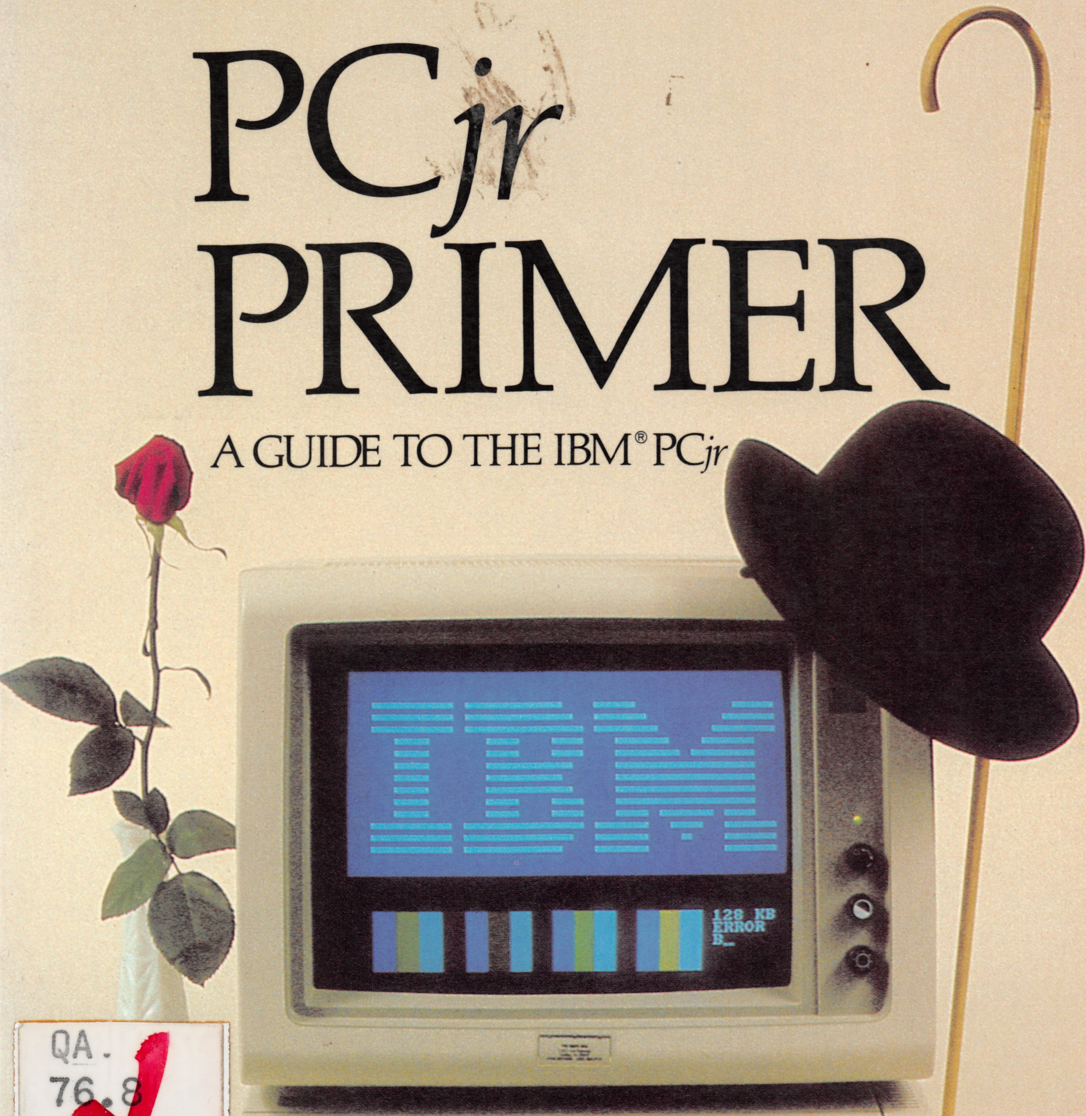
PCjr PRIMER

A GUIDE TO THE IBM® PCjr



STEVE STERN GREG YOUNG

ERRATA

Please note the following corrections:

Page 31, paragraph 4 (in parentheses), last sentence:

The command in the sentence beginning "On the other hand..." should read "copy MYfile B;" rather than containing a colon (:) after the B.

Page 39, third line:

"....Z will appear..." should be "...^Z will appear...".

Page 78, fifth line:

"When F4 is pressed..." should read "When ALT-F4...".

Page 88, def s-f2

Change " = plpage up]..." to " = [page up]...".

Page 112, middle of the page:

The word "Note" in the left-hand column of the chart should be deleted. The correct keystrokes to be entered at cell R1C14 are:
C F R4C2:R29C2 <ENT> <DOWN> <RIGHT>

Page 117, table entry for R8C15:

The formula is missing its close parentheses after [-7]; it should read:
+ R [-1] C + (AMT * RC [-7]) <ENT>

Page 133, the NOISE program:

The variable N should be an integer. Change all occurrences of it to N%, as follows:

```
10 SOUND ON
20 FOR N%=0 TO 7
30 NOISE N%, 15, 250
40 PLAY " ", " ", "VO"
50 FOR I = 1 TO 6
60 PLAY " ", " ", "V15; O = I; CDEF
70 NEXT I
80 NEXT N%
```

Page 140, line 2:

The phrase "produce a wider range of colors" should read "produce truer colors."

Page 141, top line of chart:

There should be a "no" in the Expansion column for the 160 × 200 mode.

Page 144, middle of page. The paragraph should read:

In fact, you can change the meaning of this bit to "Background Intensity," thus giving 16 background colors:

in 40-column width... OUT &H3D8,8

in 80-column width... OUT &H3D8,9

QA
76.8
.I2593
S74
1984

PCjr Primer
A Guide to the IBM® PCjr

by

Steve Stern and Greg Young

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

This book is dedicated to Charlotte and Martin Orem and to Diane Stern, without whom none of this would have been possible.

Library of Congress Cataloging in Publication Data

Stern, Steve
PCjr Primer.

"A Reston Computer Group book."
1. IBM PCjr (Computer) I. Young, Greg.
II. Title. III. Title: P.C. jr Primer.
QA76.8.I2593S74 1984 001.64 84-4766
ISBN 0-8359-4736-X

Editorial/production supervision and interior design
by Judy Elliston Coughlin

Photos by Jack Zibulsky

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

Contents

Acknowledgements

There are many who helped bring this project to fruition. The first in line, however, must be Greg Young, whom I have known as my closest friend for some 25 years. Without his knowledge of computers, his writing capabilities which are unique to the techie world, and his patience, this work would not have been completed. So—a monumental toast to Greg Young and our friendship.

Diane Stern and our children, David and Andy, are well-deserving of credit for their patience throughout this project, their input into the book, and, in general, their service which was above and beyond the normal call of duty.

At Reston Publishing, Ron Powers and Phil Menzies shepherded the project through to completion, providing not only moral but also tangible support in making available whatever tools were necessary to the project.

Susan Terry and her employer, Satellite Software International, deserve a great deal of thanks for their support and their word processing program, WordPerfect®, with which this book was written and which did not crash once during the entire ordeal.

Don Withrow and his company, Janadon, Inc., the producers of HostComm, the Personal Computer Host Communications System, deserve acknowledgement and thanks. It was through HostComm that much of this work was transmitted between the authors and the coasts.

Robert Volberg of the Van Ness Avenue ComputerLand of San Francisco, Bob Hebert of Sears Business System Center, Rockville, Maryland and ComputerLand in Bethesda, Maryland made the PCjr, its software, and their time available to the authors.

Others for whom thanks are in order include Harry Lipkind of A. J. National, Rockville, Maryland, who delivered the hard disk just in time and lent a hand whenever necessary; Office Furniture Mart of Washington, D.C., which delivered the ergonomic chair to Steve Stern just before his back was ready to give out; Harvey Perlstein of Bethesda Computers, Bethesda, Maryland; the IBM® PC, on which this entire project was produced; and all of those friends, associates, and family who lent their love and support throughout.

As is often said, last but not least, I thank Joe Guilfoyle and Bob Rettig of Dow Jones and Company, Inc., who have stood by me for the past 15 years with friendship and support and who taught me that English was not my second language.

Steve Stern

Introduction

In October, 1980, IBM burst on the microcomputer scene with its IBM® Personal Computer, a hot system destined to redefine the small computing industry. Fully configured by IBM, the PC comprised two single-sided disk drives, 256,000 bytes of Random Access Memory (RAM), Monochrome Display, printer, asynchronous communications adapter, graphics adapter, and game control adapter for a total of slightly over $5,000.00. There was little more than that hardware available and roughly a dozen software packages—all from IBM—for the PC. The software included a primitive word processor, VisiCalc®, a communications package, a couple of educational programs, and a game, among others.

In 12 months, hardware and software burgeoned and there were over 1,000 products for the system; at present, original equipment manufacturers (OEM's) have made more than 3,000 products, including both hardware and software, available for the PC.

There are now books on the PC, tea cozies for the system, seminars, a dozen magazines, and scores of IBM PC "users groups" to reel off just a few of the outside industries birthed from IBM's onslaught.

But the IBM PC did not actually become the answer to the "Home Computing" market that IBM had anticipated—sure, there are thousands of PCs sitting in homes, but for the most part they have become the standard in small business computing, replacing even the so-called mini-computer, or heretofore larger capacity systems in many cases.

Enter the IBM® *PCjr* to fill IBM's home computing void: a system at about half the price, when fully configured, of its big brother, that will operate most (as you will see throughout this book) of the software packages available for the PC.

It is an exciting, flashy system in the realm of Buck Rogers, with its infrared keyboard and cartridge drives. It is a system that will burgeon as did the PC and, in the IBM tradition, give you virtually any benefit you desire from a computer.

The purpose of this book is simply to give you an overview of *PCjr*. This is not to say that the overview will be cursory. It will explain the workings of the system, in what we hope will be a casual, informal, nontechnical and perhaps even entertaining manner.

We will teach you how to set up your *PCjr* system, running you through the "do's" and "don't's" and showing you the problems that might occur. And we'll teach you about the software that's available, commercially and noncommercially—and how to get that noncommercial or free software.

We will not teach you how to program, however; that field is better left to books dedicated to that purpose. After all, unless you have a very specific need, or a burning desire to learn BASIC, PASCAL, or whatever, there's no real reason to acquire the knowledge. Everything you can imagine is already on the market—even programs to write programs for you.

Here we go now; sit back and enjoy.

1

What Is the PCjr?

The *PCjr* currently comes in two models, Model 4 and Model 67. The essential difference in the two models consists of a disk drive and memory expansion facility.

The Model 4 arrives with 64,000 (64K) of RAM (Random Access Memory), or a storage capacity of approximately 64,000 characters and two cartridge slots; the Model 67 adds a disk drive, a facility for an additional 64K expansion of RAM, and 80-column text display support.

If you plan to start small, with Model 4, don't worry; everything that comes with the Model 67 is available for the 4 and you can add it later.

The initial 64K of memory available on Model 4 will be adequate to operate all of the programs available through IBM (as opposed to the OEMs) for *PCjr.* The additional 64K available for the Model 4 and indigenous to the Model 67 will open many new software and computing vistas for you. You may want to avail yourself of this initial option at the outset of your *PCjr* experience.

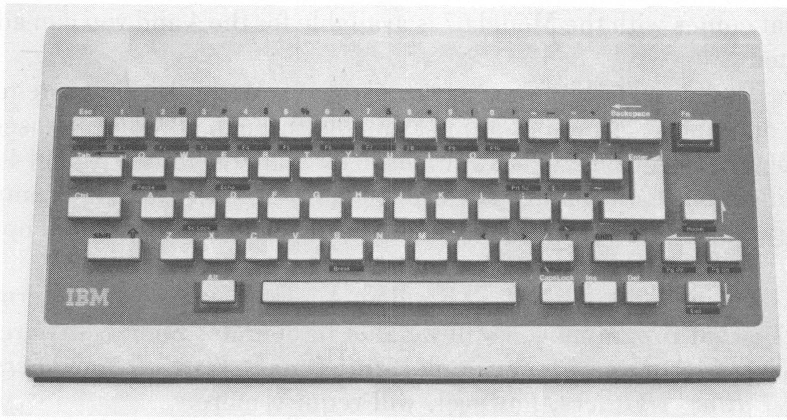
The amount of RAM, or Random Access Memory, will determine just what programs you will be able to operate. Some software requires 64K or less—for example, *MultiPlan* ™, *VisiCalc*®, and *Personal Editor* ™. Others, however, will require more.

Both units come with a 62-key cordless keyboard, which provides all functions of the IBM PC's 83-key keyboard. The keyboard is the real "gee whiz" component of *PCjr,* operating on an infrared beam that will communicate with the system unit from as far away as 20 feet. Four AA batteries are required.

The keyboard, incidentally, weighs less than two pounds with batteries and measures about 13.5 inches long, 6.6 inches deep and slightly more than an inch high. You won't have much trouble carrying the unit to your favorite armchair.

The keyboard can be connected to the system unit via the *PCjr* Keyboard Cord, which automatically disengages the batteries and disables the infrared link. If you happen to have more than one *PCjr* in your home or office, the cord option is a necessity to prevent "cross-communication"—not unlike cordless telephones or garage door openers.

You'll notice from the illustration that the keys are arranged in the basic typewriter layout, with a couple of additions: a function key and some cursor control keys (more on them later). The tops of the keys are not labeled; this allows for complete customization of the keyboard. In other words, you are able to make any key do whatever you desire . . . within reason. Keyboard overlays, made of heavy paper stock, fit over the keyboard to designate personal customization.



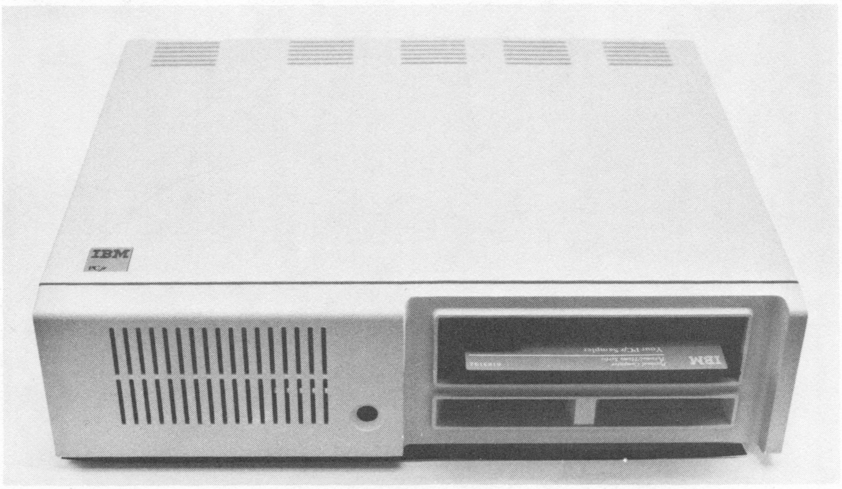
Again, these are an option which you can easily avoid by using a precomputer age pair of scissors.

After you have experimented with the infrared facility, you will find that when the unit is placed on a surface other than your lap, it has two tilt positions: the "normal" position at 5° and the extended leg position with a 12° slope.

The system unit in both models also includes cassette BASIC and cassette DOS, which are respectively the programming language BASIC and the computer's "manager" or operating system necessary to run programs. Again, we will deal more fully with these facets in subsequent chapters.

The disk drive, 64K memory, and display expansion are both included on the Model 67. They are, as mentioned earlier, applicable to the Model 4. In fact, without the disk drive on the Model 4, you won't be able to save any of the work you have done unless you have a printer. The drive is half the height of the standard IBM PC disk drive.

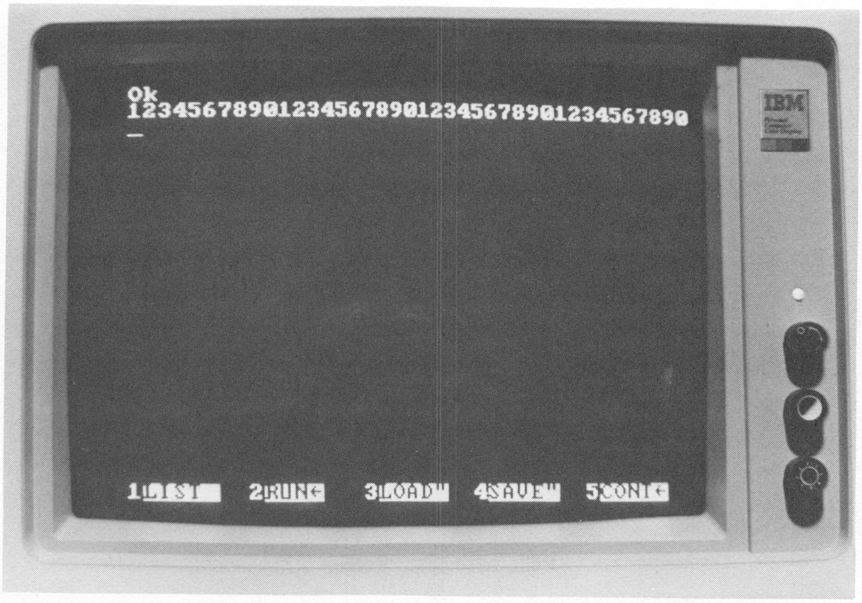
In spite of the size difference, it does precisely the same thing with the same storage capacity, although somewhat more slowly. With the disk drive you can store up to 360K of material. Without it, you can store absolutely nothing, except on paper.



PCjr with disk drive

The memory and display expansion has a couple of functions. First, it obviously increases Random Access Memory by 100%, to 128K. More importantly, it creates an increased density video mode allowing for 80-column text on your screen. In computerese, a column is simply a space, and generally there are 40 or 80 of those spaces (all of which may be filled with a character, letter, or number) on a screen; 8C is preferable for such things as word processing, as it is considerably easier to read.

An internal modem is available for *PCjr.* A *modem* is a communications device that allows your computer to talk with other computers over the telephone line. At first glance you might wonder why letting your computer talk to another would be advantageous. Think about terms like high-speed data communications and telecommunications: with a modem your *PCjr* can talk with commercial data bases such as CompuServe® and hundreds of electronic bulletin boards throughout the country from which you can actually get "free" software.



```
Ok
1234567890123456789012345678901234567890
_



1LIST   2RUN←   3LOAD"   4SAVE"   5CONT←
```

Photo by Wayne Partlow

40-column screen

```
Ok
12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Photo by Wayne Partlow

80-column screen

Besides talking to data bases, *PCjr* could be used to communicate with your office, assuming, of course there's a computer there with which to communicate.

IBM has even given *PCjr* its own thermal printer that runs at 50 characters per second and can also be used on big brother, the Personal Computer.

Then there's the gamut of usual computer accessories including an RF modulator to connect the *PCjr* to a television, joy sticks for games, an accessory to attach a printer, a cable for the IBM color monitor, and even a carrying case with a combination lock.

2

A Little More Specific

Here's where we'll get into a bit of techno-jargon, just to give you a taste of it. Don't worry, it will be just a touch and fully explained as we go along.

As mentioned before, the most exciting feature of *PCjr* is the wireless keyboard. According to IBM, the infrared transmitter it uses will run for about three months on four AA alkaline batteries, with heavy use.

You can use the keyboard in its wireless state up to 20 feet from the system unit, but you must be directly in front of the unit, with no obstructions. If the keyboard is not communicating directly to the console, the software controlling the keyboard will warn you audibly whenever a "garbage" character is received. By the way, keep bright lights away from those Light Emitting Diodes (LEDs) on the front of the keyboard, for they too will cause characters to be incorrectly transmitted.

The *PCjr* keys are fairly easy to use. There are no indentations in them as in the PC to help keep the fingers centered, but the keyboard does use full travel keys. Moreover, the keys are standard size, thus allowing for touch typing. When you strike a key there is tactile feedback and a soft click, compared with the rather loud click emitted from the PC.

Looking at the box (the computer or system unit) from the front you'll see that half-height disk drive embedded just above the two cartridge slots on the Model 67. Of course, with the Model 4, there is just a covering plate, which can be removed later for installation of the drive.

Just to the left of the cartridge slots is the receiving lens for the infrared transmitter. That's what reads the input from the keyboard when in its wireless state.

At the rear of the system unit, you'll spot the cassette port, or jack—the spot into which you may plug a cassette recorder for storage or applications programs. Among the more fascinating features of that cassette port is the fact that it will allow the system to act as a sort of sound mixer while controlling the motor of the cassette recorder. There is also a built-in serial, or RS-232, or Asynchronous Communications port (any of the three will do for our purposes). This port allows for the use of an external modem, should you opt not to purchase the *PCjr's* internal communications device. Installing *PCjr's* internal modem, by the way, will allow for direct telephone connection through the rear of the system unit. Modems are more fully explained in Chapter 11.

Also in back are ports for monitors which make *PCjr* compatible with monitors other than the IBM-PC. You'll be able to connect your system directly to an RGB or composite monitor. However, should you want to hook up to a regular television set, you'll have to buy an RF modulator. Grouped with the video ports are a couple of input ports for joysticks and one for a light pen.

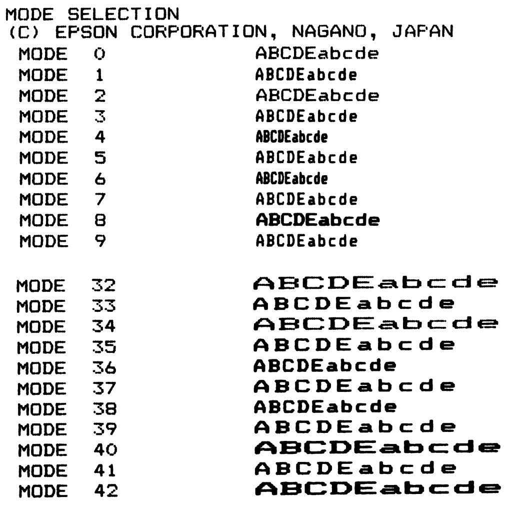
Just to give you a taste of things to come, look at the right rear of the system, where you'll see an expansion connector. Ultimate utilization of that module will allow that 128K memory maximum to be popped up to perhaps 640K. Again, a subsequent chapter will deal with *PCjr's* future.

3

Printers

In keeping with the *PCjr's* lower cost, IBM has introduced an inexpensive thermal printer. For less than $200.00 you can acquire the IBM Personal Computer Compact Printer 5181 and have a dot matrix printer that will print on standard size 8.5 by 11-inch paper.

Though the paper size is standard, the paper is not; it is chemically treated to use the printer's thermal technology. Basically, thermal printing is printing through a chemical and heat process.



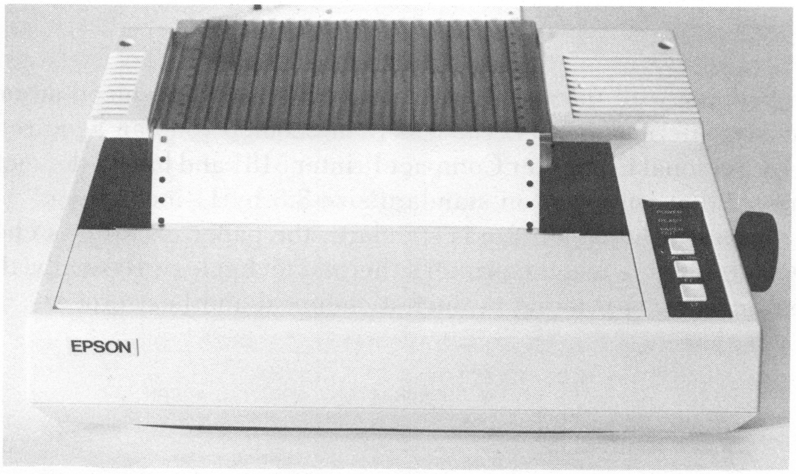
```
MODE SELECTION
(C) EPSON CORPORATION, NAGANO, JAPAN
 MODE  0            ABCDEabcde
 MODE  1            ABCDEabcde
 MODE  2            ABCDEabcde
 MODE  3            ABCDEabcde
 MODE  4            ABCDEabcde
 MODE  5            ABCDEabcde
 MODE  6            ABCDEabcde
 MODE  7            ABCDEabcde
 MODE  8            ABCDEabcde
 MODE  9            ABCDEabcde

 MODE  32           ABCDEabcde
 MODE  33           ABCDEabcde
 MODE  34           ABCDEabcde
 MODE  35           ABCDEabcde
 MODE  36           ABCDEabcde
 MODE  37           ABCDEabcde
 MODE  38           ABCDEabcde
 MODE  39           ABCDEabcde
 MODE  40           ABCDEabcde
 MODE  41           ABCDEabcde
 MODE  42           ABCDEabcde
```

Type fonts

The 5181 will print at 50 characters per second (cps), allowing for a variety of type fonts, including compressed and double width.

The *PCjr* user is not, however, limited to the 5181. A variety of dot matrix and letter-quality, IBM-compatible printers are on the market. Of the dot matrix printers, the most utilized with the IBM family is the Epson RX or FX 80, operating at 100 cps and 160 cps respectively. Epson, incidentally, manufactures the IBM Graphics Printer that usually accompanies the IBM-PC. It is a member of the older Epson MX series.

Using a nonthermal printer would probably be less of a hassle, as the paper *is* standard. Moreover, the print quality is generally superior to thermal printing and has greater staying power. The expense, however, is at least 100 percent greater.



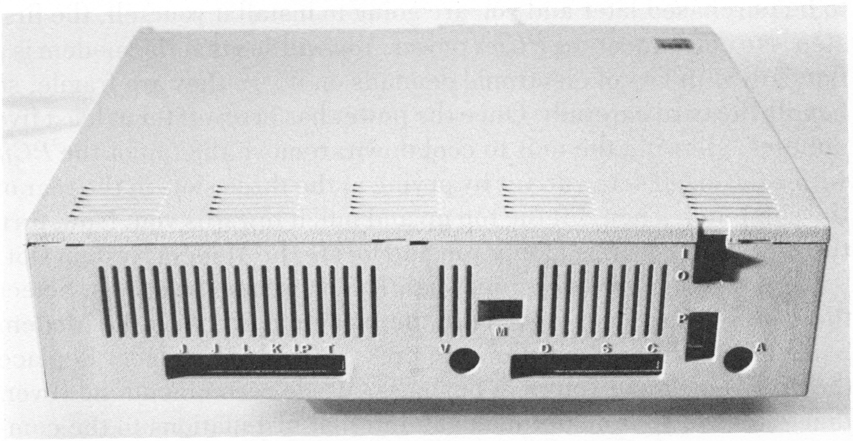
FX 80 printer

4

Getting Started

Setting up your *PCjr* is straightforward and not unlike setting up a stereo.

As you can see from the illustration, there are 11 sockets, each labeled with a letter. To begin, make sure the power switch on *PCjr* is in the *off* position (down). Your first step will be to plug in the power



supply, which entails simply plugging the appropriate plug into the "P" outlet in the back of the unit; the standard wall plug on the power supply goes into a wall outlet.

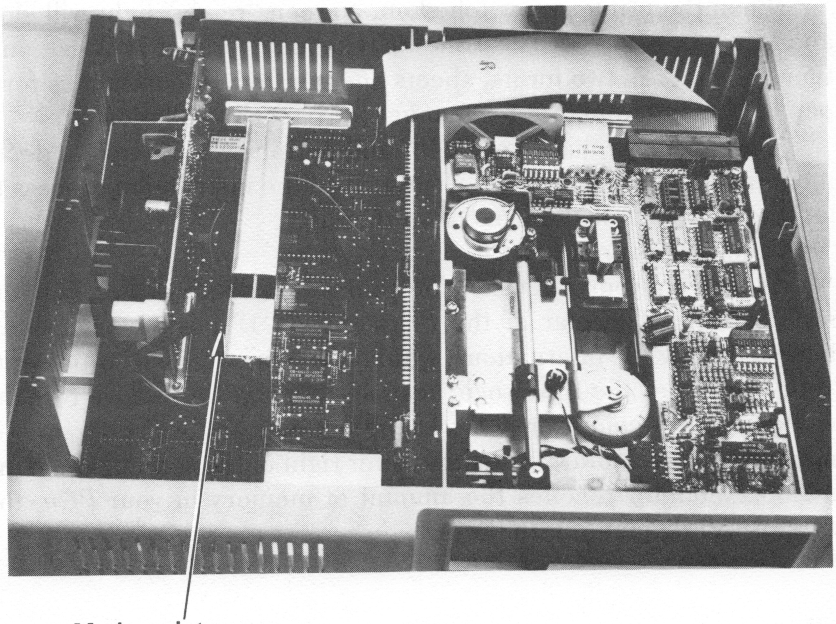
If the IBM Color Display is being used, you will need the *PCjr* Color Display Adapter Cable. After finding the end of the cable stamped with a *D*, plug that end into its matching outlet on the system and the other end into the IBM Color Display signal cable. Then plug the Color Display's power cord into a wall outlet. A similar procedure is followed with television set connection and the use of an RF modulator, although the outlet in the unit is designated *T* for television, or *C* for composite. Oddly enough, the only display (monitor or screen) that does not require a relatively expensive cable is a composite screen. All you need for a composite is a standard RCA stereo plug; connect it from the display to the *C* outlet in the system unit.

Before you switch the power on, be sure the *PCjr* system unit and the display are at least six inches apart, or the disk drive will not work properly. **Remember, at least six inches must separate the display and the system unit.** If this separation is not maintained, your DOS diskette, or any other, for that matter, will not boot. In other words, the disk drive will not use the data on the disk. Of course, this is true only for *PCjr's* with a disk drive. However, the separation is still recommended for machines without a drive.

If you acquired an internal modem for your *PCjr* and it is already installed, there is little left to do but plug your modular telephone cord into the outlet designated *M* in the system unit. If the modem is to be purchased later and you are going to install it yourself, the first step is to disconnect the *PCjr's* power. You will see that the modem is a flat card with lots of electronic doodads on it . . . they are fragile, so handle the card carefully. Once the power has been off for at least five minutes, allowing the unit to cool down, remove the top of the *PCjr* with a flat-head screwdriver by prying at the three slots in the rear of the computer. Then, lift the top up and pull it towards you. Now, turn the unit so the front is facing you and locate the Modem System slot.

You will see three slots, one slightly longer than the others. Select the middle short one. Now, take the modem card, with the Modem System Connector in position and press it firmly into place. Replace the unit's cover and you're in business. We do recommend, however, that you have your dealer make all internal installations to the computer, unless you have had previous experience.

If you plan to use the keyboard connector cord, again make sure the power is off, find the end of the cord marked *K*, and plug it into the



outlet marked *K* on the back of the system unit. Finally, insert the other end of the cord into the keyboard outlet that looks like a modular telephone outlet. If you plan to use the keyboard without the cord, you **must** unplug the connector cord from both the keyboard and the system unit. (The connector cord, incidentally, is an optional item.)

Using the infrared capabilities of the *PCjr* keyboard requires that the keyboard be directly in front of the system unit to allow the computer to receive the keyboard's output.

If you are using the IBM PC Compact Printer, installation, again, is fairly simple. You will require thermal paper to complete the installation. With all power for the computer off, plug the power cord into the appropriate outlet on the printer (a three-wire grounded outlet) and the other end into a wall outlet. The printer-to-computer cord is plugged into the outlet designated *P* on the computer.

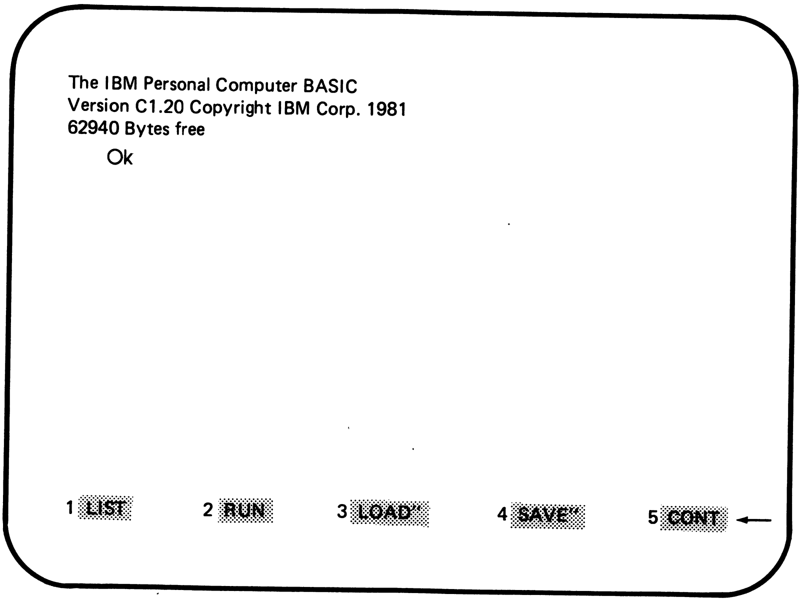
It should be growing obvious at this point that if you are running a display and a printer, you will require at least three wall outlets in close proximity to the computer.

When the printer is switched on, a green "ready" light will glow and it is set to print. The IBM PC Compact printer will use only thermal paper in two forms, sheets and fan fold, or continuous form paper. Consult your printer manual for paper installation.

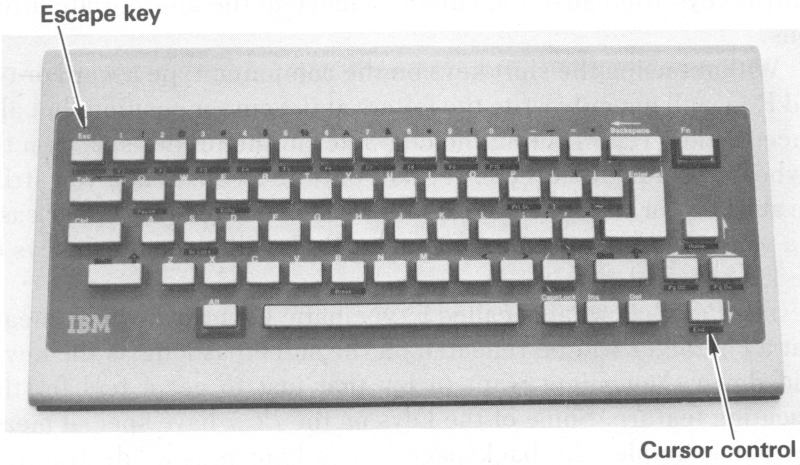
When you purchase your *PCjr* and peripherals, complete documentation is provided. We have described the installation process to give you an idea of the ease of setting up.

Now, on to running *PCjr.*

Switch on your television or your color display, then flip the switch at the left rear of the *PCjr* to the "1" position. The "1," incidentally, is the international symbol for "on" while "0" represents "off." You'll see the IBM Logo, against a blue background across the screen, with a color bar chart at the bottom. You will also see the memory being counted off on the lower right of the screen. Once the memory counter reaches the amount of memory in your *PCjr,* the screen will display the following:



```
The IBM Personal Computer BASIC
Version C1.20 Copyright IBM Corp. 1981
62940 Bytes free
Ok
```



Now, you are up and ready to run IBM's "Keyboard Adventure." Strike the "Esc" key at the upper left hand corner of your keyboard. Assuming you are using a color display or color television, a red rectangle will be drawn on your screen and a little cartoon person IBM has named P. C. will trot down the right-hand side of the display. Using the *cursor control keys* located on the right hand side of the keyboard, take P. C. for a walk. You'll notice that these keys have directional arrows and are also designated by small green banners. We'll get to those banners a bit later; for now, just move P. C. around the screen. When you rudely run him into a border, he'll beep, thus telling you he wants to move no further in that particular direction. But what of the rectangle P. C. hasn't entered? Send our friend up to the reverse *L* on the left hand side of the screen and cover the symbol with him.

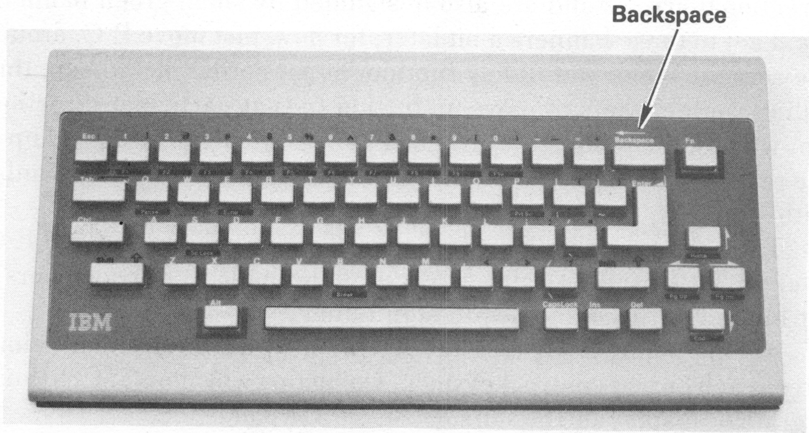
The secret door in the rectangle will open, inviting you and P. C. to enter. Do so and you will see P. C. fall through a variety of colors—let him go until he decides to stop falling.

At this point, P. C. is ready for his next adventure. This time, press each cursor control key once. Doing this will allow P. C. to build a graphic display of the cursor control keypad.

There is a small rectangle at the upper left of the screen that represents a cursor, which in computerese is a character that shows you where you are on the screen. Striking the left or right cursor control keys will cause the cursor to move in the appropriate directions.

Without using the shift keys on the computer, type a word or two and P. C. will not only write the letters at the cursor position, but also place the key representing the character in the proper space on the keyboard rectangle in the lower half of the screen. When you strike the shift key or the space bar, a letter will be moved into upper case, or a space will be inputted; of course, P. C. will place those keys on the keyboard also.

The *PCjr* has what is called a typematic keyboard, which means that a character will be repeated on the screen as long as the key is held down. You might want to try that just to get a feel for this repeating feature. Some of the keys on the *PCjr* have special meaning. For example, the backspace key is known as a "destructive" backspace, meaning that when it is struck, characters to the right of the cursor will be erased. If you just want to move the cursor to a character without erasing it, use the cursor control keys. The space bar, by the way, will erase characters to the right of the cursor when held down.

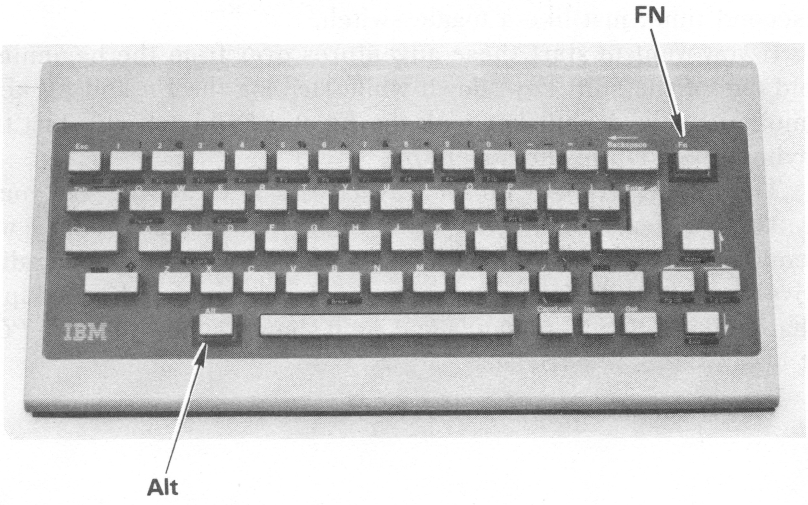


It is important to point out here that certain keys on the *PCjr* keyboard are used in conjunction with other keys. These keys are color coded in green, black, and blue. The green functions are always used with the *Fn* key, the blue with the *Alt* key, and the black with the *Shift* key.

Many of the programs we discuss in this book use the function keys. They must be activated by striking the <*Fn*> key first, then the *Function* key required. The function keys may be "toggled" on by striking *Shift*-<*Fn*>. That action allows you to operate them without hitting the <*Fn*> key each time you want a *Function* key operation. *Shift*-<*Fn*> struck a second time will put you back in the normal operational mode.

Let's begin with the *Alt* key. To use the functions with which it is associated, hold the *Alt* key down, while striking another key, such as the backslash. The backslash will appear on the screen. If you don't use the *Alt* key, striking the backslash key will yield a regular slash. **Remember: To activate the *Alt* keys, you must hold the *Alt* key down while striking the associated key.**

These function keys will be discussed in part here and in part in other chapters on *HOMEWORD* and *MultiPlan*. Give them a try by



striking the *Fn* key and then the *F6* key. Your screen should change color. Hitting the *Fn* key followed by the *F5* key will bring you back to the original color. You'll get similar results by using the *Fn* key in combination with the *F4* and *F3* key. If you strike the *Fn* key by accident, simply press the Shift key to cancel the Function mode. Again, you can lock the keyboard into the Function mode by pressing the Shift and *Fn* keys together and then striking the *Esc* key. When this "lock" is performed, all the green striped keys will operate in the Function mode without pressing the *Fn* key first. To unlock this, again press the Shift and *Fn* keys and press the *Esc* key.

If you have a display with an amplifier and speaker, the *Fn* key in conjunction with the *F2* key will yield music. To stop the music, use the *Fn* key with the "Break" key, which is also designated *B* on the keyboard. This combination may also be used to stop any program that is running.

PCjr, like all computers, has a wonderful memory. If there is nothing on the upper portion of your screen, type a sentence and hit the enter key. Your text will disappear—but strike the *Fn* key, followed by the *F10* key, and presto, your text is back again. The *Fn* and *F7* keys, by the way, will fill in the rest of P. C.'s keyboard, if you haven't used every key yet.

To get our friend P. C. moving again, use the *Fn* and *F8* keys, then go ahead and run our little buddy around with the cursor movement keys. The cursor may be reactivated by using the *Fn-F8* combination a second time, just like a toggle switch.

If you want to start these adventures over from the beginning, hold one of the shift keys down while striking the *Fn* and *F9* keys simultaneously. A shift key with the *Fn* and *F10* keys will start the keyboard portion of the adventure.

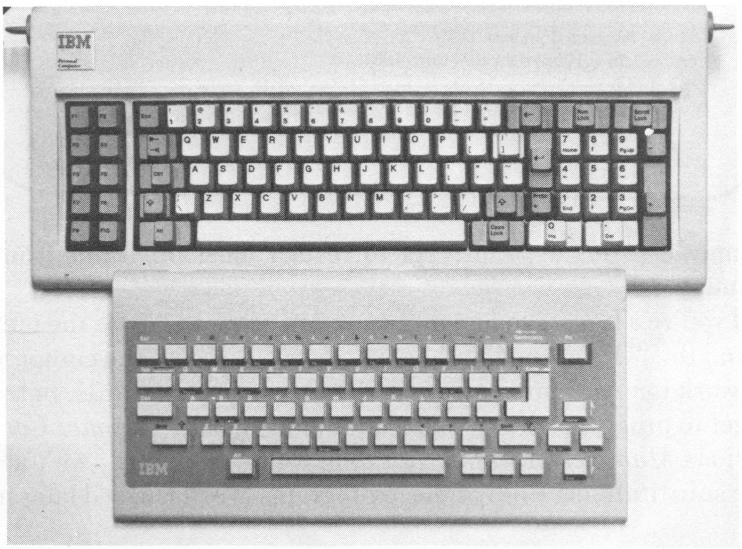
That about wraps it up for travels with P. C. It will give you a pretty good idea of the *PCjr* tutorial. In the following chapter, we will wend our way through the *PCjr* keyboard. We'll discuss using disk drive later in the book. If you want a quick start on that drive, skip to Chapter 9 on DOS Commands and we'll show you how to make *PCjr* sit up and do a few tricks.

5

What Are All These Keys?

IBM points out that *PCjr's* keyboard with 62 keys operates just like the PC's 83 keyboard.

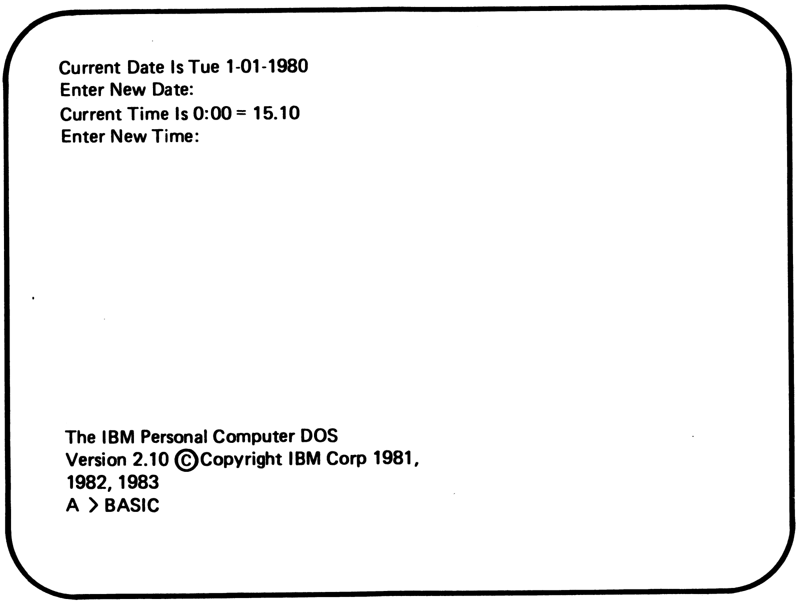
As we mentioned in the previous chapter, the major difference in use involves using combinations of keys, such as the *Fn* key, to



PC 83 keyboard (top) and PCjr keyboard

activate the Function keys or the *Alt* and Shift keys to operate others on the *PCjr* keyboard.

Here we'll go over some keyboard activity not discussed in the previous chapter. One of the most important key combinations to remember is *Ctrl-Alt-Del,* meaning Control, Alternate, and Delete. Using these three keys together performs what is known in computer talk as a *system reset.* To do this, hold the *Ctrl* and *Alt* keys down simultaneously, then strike the delete key. Assuming there is nothing in the disk drive or cartridge slots, your screen should show the following:



```
Current Date Is Tue 1-01-1980
Enter New Date:
Current Time Is 0:00 = 15.10
Enter New Time:











The IBM Personal Computer DOS
Version 2.10 ©Copyright IBM Corp 1981,
1982, 1983
A > BASIC
```

You may use this system reset to restart most programs from the beginning.

If you're a typist, you might be used to using an "l" for the number "1" and the "O" for the number "0." Don't do it with the computer. It may work for word processing or when you use only words, but when you get to programs that require numbers, such as *Personal Communications Manager* or *MultiPlan,* you'll find they just won't accept these substitutions. Both of these programs are discussed later in the book.

The *CapsLock* key is a toggle switch, meaning that it is turned either on or off every time you strike it. It operates a bit like the Shift

Lock key on a typewriter—when it is on, all letters typed will be in upper case. If *CapsLock* is on, you may type lower case by using the shift key. However, the *CapsLock* key will not allow you to use the shift characters such as !, @, #, $, %, etc. You must use the shift key, characterized on the keyboard in black letters, to access these characters and any other functions that are colored black on the keyboard.

If you don't like the silence of the *PCjr* keyboard and need to hear the click click of a typewriter, IBM has taken care of that requirement: Hold the *Ctrl* and *Alt* keys down and press the *CapsLock* key.

6

What Is a DOS?

DOS stands for "Disk Operating System." To explain the origin of DOS, we have to look at some computer history.

Long ago, when computers were first introduced to the business world (in the 1950s—that's long ago to this industry!), their internal speeds were measured in thousandths of a second, or "milliseconds." ("Internal speed" means the rate at which the machine performs its instructions, like adding two figures together, or moving data from one place in memory to another.)

The IBM *PCjr*, like all personal computers, has internal speeds measured in millionths of a second (microseconds); large "mainframe" computers operate at speeds quoted in billionths of a second (nanoseconds). A nanosecond, by the way, is to one second as one second is to 36 years.

At any rate, even when computers loafed along in the millisecond range, they were far faster than the devices which supplied their input and handled their output. Card readers, for example, could supply data at about 100 characters per second. Printers ran at about the same rate. These early computers simply stopped running momentarily when performing input or output (i/o) functions. When a line had to be printed, the computer sent the line to the printer, and then waited until it received a "ready" message from the printer, indicating that the line had been printed.

The central processing unit (CPU) would spend far more time waiting for data than it would spend in processing it. Yet the CPU was a lot more expensive than any of the i/o devices. Seemed like the tail was wagging the dog.

Somewhere along the line, somebody realized that if you attached another printer and another card reader to a computer, two programs could run more or less at the same time. While one program was waiting for input or output, the CPU could be working on a second program. When the second program stopped to wait for i/o, the CPU could service the processing needs of the first program. But how would the CPU know when to switch back and forth? In fact, *how* would it switch back and forth? Enter the operating system. An operating system (or "control program," as they were originally known) is simply another program. This program however, is very powerful. By making use of special CPU functions, the operating system can control other programs which run in the same CPU with it.

Originally designed simply to act as a traffic cop within the CPU, operating systems soon were given additional tasks. The operating system for the IBM *PCjr*, DOS, helps other programs perform i/o, updates the hardware clock and calendar, keeps track of where files are on the diskette, and much, much more.

DOS stands for Disk Operating System. Does this mean that there are non-disk operating systems? Well, no; not any more. Before disks came along, many large computers ran with tape drives, in addition to card readers and printers. When disks started to appear, they required special operating system support (that is, additional logic within the control program), because performing i/o on a disk is nothing like performing i/o on a tape. So there were Tape Operating Systems and Disk Operating Systems. Today of course, operating system support for disks is taken for granted. And that's enough history!

7

What Do DOS Do?

DOS, as we shall speak of it, comes in many parts. The real guts of DOS consists of two programs.

The first of these controls the computer hardware itself. The DOS functions include performing input/output operations for other, "application," programs (like word processing and spreadsheet programs), maintaining the system clock with the correct time, and so on.

The second program at the heart of DOS helps to manage diskette files and provides other basic services. For example, when a word processor saves a text file, it is DOS that finds space available on the diskette, writes the data to the diskette, and then updates the diskette directory with the file's name and location. The word processor does not have to know where on the diskette the file has been stored; it can retrieve the file simply by giving DOS the file name.

These parts of DOS may be critical to the proper functioning of the computer, but you do not have to get very involved with them unless you get into writing programs at the machine language (Assembly Language) level. What you do need to be concerned about are the *DOS commands*.

DOS commands are really just programs (or parts of programs) that run when you enter the command names on the keyboard. *DIR*, for example, is the DOS command which lists the contents of a diskette directory. The directory contains the names of all the files on the diskette. If you use the *DIR* command to list the contents of the DOS system diskette, you will see a lot of files which have the name of a DOS command, followed with .COM; for example, *FORMAT.COM* or *CHKDSK.COM*. You may notice that many DOS commands are not in the directory. These commands are those "other services" contained in the second of the basic DOS programs.

So, for example, if you ask DOS the time, it will immediately display the current clock value (and give you a chance to change it). But if you ask DOS to format a disk, it will first have to load the Format program *(FORMAT.COM)* from diskette into memory. The DOS manual contains a chart called "Summary of DOS Commands." This chart shows whether the command is built into DOS ("I" for internal), or if DOS has to read the program in from diskette ("E" for external).

The key to effective use of any personal computer is coming to terms with the operating system commands. Properly used, they can save you time and effort. They can help protect you from disaster and guide you through complex procedures. The chapters that follow are meant to start you off on the right foot.

8

Words About Words

Before we go any further, there are a few terms you should know.

A computer is a box containing a bunch of electronic circuits. The circuits are contained in solid-state devices called chips. Each type of chip is designed to perform a specific function. One chip, the microprocessor, does all the thinking. Thinking, in this case, means arithmetic and logic. (What else is there? For starters, there is creativity. Computers—even big ones—don't do that . . . yet. Creativity is your department.) The microprocessor is also called a Central Processing Unit, usually abbreviated as CPU.

The CPU is just a set of circuits. A program, running in those circuits, is needed to make the CPU useful. A program consists of a series of instructions. The instructions tell the CPU where to find data and what to do with it. Programs are also called software. (Isn't this easy?)

A computer must also contain some amount of memory. It takes different kinds of chips to provide memory. Some chips in the *PCjr* contain data and programs which you, as a computer user, are not able to change. (The programs include cassette BASIC. The data includes definitions which tell *PCjr* what its characters should look like.) Memory which is not changeable is called *Read-Only Memory*, or ROM. This means that the CPU can retrieve data from the circuits, but cannot alter their contents.

The *PC* also contains a kind of memory that can be altered. This is used to store the results of calculations, to hold programs while they are being run, and so on. This kind of memory is called RAM, for *Random Access Memory. Random Access* means that the CPU can address any location in memory instantly (as opposed to searching through it sequentially, like turning pages of a book to find something).

When you hear of the "memory size" of a computer, it usually refers to the amount of RAM that is available. Memory of any type is almost always measured in terms of bytes. The easy definition of a byte is that it is equivalent to a character of data.

(For those who wish to become more technical, the above, though correct, is not complete. A byte consists of eight bits. A bit is a single unit of memory and may be either a "one" or a "zero." A single byte can contain values from 0 to 255, which is to say, all bits set to zeros up to all bits set to ones. For a more detailed explanation of the binary system, please refer to the Binary Appendix. Don't get too bogged down here . . . most computer users don't need to get any more technical than this.)

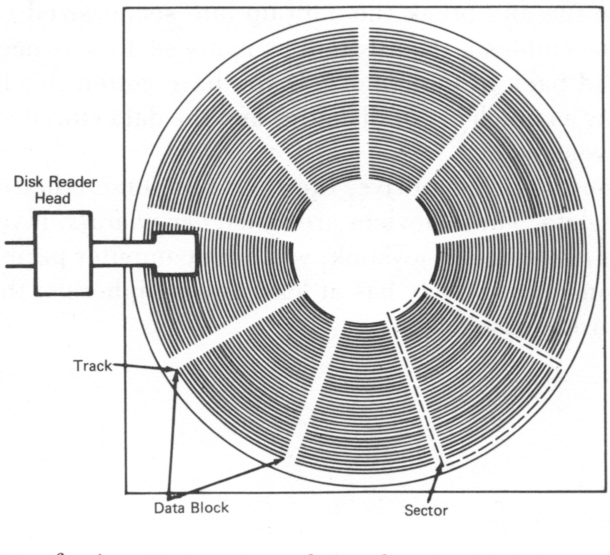
Since one byte doesn't represent a great deal of use, computer memories typically come in thousands of bytes. There is a complication, however. The closest "even" binary value to decimal 1000 is 1024 (two to the tenth power). Hence, a "thousand" bytes in a computer is really 1024 bytes. This number is referred to as *1K*.

Okay, now we have a computer, with its CPU and bytes of memory that can run software. All the software and data that you will ever need cannot fit into computer memory. To store programs and data outside the computer, we use floppy diskettes.

Diskettes (also called disks) are made of a flexible plastic called mylar, coated with a magnetic material (iron oxide, commonly called rust), and enclosed in a cardboard jacket. The jacket has an oval cutout in it to allow you to see the mylar surface. The exposed surface should *never* be touched; the microscopic bits of dirt and oils from the cleanest hands can damage the diskette so badly that it cannot be read. Data on diskette is read and written by the diskette drive in the *PCjr.* Once you get the hang of it, loading a diskette into the drive and removing it will become second nature—but always use caution; the diskette may be the most fragile part of your system.

(Stick with it, we are almost done!)

Standard ("double-sided, double density") diskettes are physically laid out in 40 concentric tracks. A track may be thought of as one groove around a record. Each track contains nine sectors; each sector holds 512 bytes. Now, if you will do a little arithmetic, you'll find that 512 times 9 times 40 equals 360K. (A "K" is 1024, remember.) and 360K bytes is what a DOS diskette will hold. (Older diskettes, formatted with DOS 1.1, have eight-sector tracks. This accounts for the fact that they can hold only 320K bytes: 512 times 8 times 40. Or, if you like, 8/9ths of the newer format. There are also single-sided diskettes, which will hold half the usual amounts of data.) Notice that the diskette capacity is a function of how the operating system formats it, not a function of the diskette drive hardware (as long as the hardware writes on both sides of the diskette, as the *PCjr* drive does). This is simply the physical layout; we'll describe how it relates to your data.



Forget, for just a moment, about the physical diskette structure, and let's turn to how data is organized. Data is normally broken down into *fields, records,* and *files.* A field is a unit of data used by a program function. Your social security number would normally be considered a field. Your last name might be one field, and your first name another. Your street address is usually broken down into fields which match the addressing lines on an envelope. Depending on how a program wants to use the data, your birthdate might be one field or be broken down into three (year, month, and day).

A group of related fields combine to form a record. The fields we have been using as examples are probably part of the employee record where you work. All similar records taken together make up a file. The terms field, record, and file have been around a while, and are pretty well agreed upon. The term *database*, on the other hand, is a little more vague. In some contexts, the term is used (incorrectly) in place of "file." A database might better be described as an integrated means of storing all data for an enterprise, so as to minimize redundancy and provide a standard means of access. (Whew!)

Now we get to the relationship between the physical layout of a diskette (512-byte sectors) and the structure of your data (any length). It's simple. Leave it up to DOS. DOS keeps track of which sectors are used as part of a file and which are available. When a program writes a record to a diskette file, DOS will allocate one or more available sectors to the file and break the data up into sector-sized chunks. DOS also reassembles a record from as many sectors as necessary when it is read back in to memory. If you have gotten this far, you probably know as much about diskettes and the data stored on them as you will ever need.

In addition to the disk drive, the *PCjr* has some other devices which handle data. These devices are called *peripherals*. If you have a printer, or a modem, or a joystick, you have computer peripherals. Congratulations! (Everybody has at least two peripherals: the diskette drive and the display.)

9

Using DOS Commands

DOS commands are used to create work disks for data file storage, set the time of day, list the files on a diskette, and a lot more. You may not be interested in many of the commands, but to make effective use of your *PCjr* you should at least have an idea of what DOS can do for you.

All DOS commands must be followed by pressing the *Enter* key.

The *Enter* key is the typewriter carriage-return key. On *PCjr*, you should reach it easily with your right pinkie, if you're a touch typist, and it is marked *Enter.*

DOS is *very* particular about the way commands are entered. Upper and lower case do not make any difference, but spaces, slashes, and the colon (:) are critical.

(For example, "COPY myfile b:" will copy the file named MYFILE from one diskette to another, with DOS telling you when to insert the destination diskette. On the other hand, "copy MYfile B:" copies MYFILE to the same diskette, and calls the copy "B.")

DOS indicates when it can accept commands by displaying a "prompt."

The standard DOS prompt is quite simple; it consists of the disk drive letter currently being used, and the greater-than sign (>). The prompt usually looks like: "A>", but the PROMPT command allows you to change the prompt to anything you like (see the PROMPT command description below). DOS will reissue (redisplay) its prompt after it completes a command, or after it gets control back from a program which had been running. It also displays the prompt after you first start up your computer.

When DOS is loaded into computer memory, the process is called booting the system. This term comes from "lifting yourself by your own bootstraps." When there is no DOS in the system to load a program, DOS does something like "bootstrapping" to load itself into memory.

Which brings up another point. A DOS command is actually the name of a program (or a part of the DOS program itself). When you enter something at the keyboard, you set in motion a chain of events. DOS first checks to see if it recognizes what was entered as one of its internal commands. If not, it checks the diskette for the name of any .COM file or .EXE file or .BAT file (in that order). When a .COM or .EXE file is found, DOS loads the contents of the file into memory, and passes control to the program. (How DOS handles BAT files will be covered in a separate chapter.) If DOS cannot find a file, it issues the "bad command or file name" message.

Not everyone who uses the *PCjr* will have to learn DOS commands in detail. (For example, see Chapter 10, "Making DOS Go to BAT for You.") It is a good idea, though, to become reasonably familiar with some of the commands. These DOS commands are as follows:

- COPY files: Needed to produce backup copies of programs or data files. Can also be used to print or display files, by copying to PRN or CON.

  Example 1: Copy a file on the same diskette and give it another name. If the second file already exists it will be replaced with the copy:

  ```
  COPY FILE1 FILE2
  ```

  Example 2: Copy a file to another diskette under the same name (note that the name need not be repeated):

  ```
  COPY FILE1 B:
  ```

  Example 3: Copy all files on one diskette to another. This has two distinct advantages over the DISKCOPY command:

1. The files will be copies to contiguous sectors. As files grow, DOS may assign sectors from different areas of the diskette. This will increase the amount of time required to access the file. The copy command shown here will correct that situation.
2. The files will be copied to the format of the destination diskette. If files must be transferred to a DOS 1.1 system, this is the way to do it: DISKCOPY will reformat the destination diskette to match the source diskette.

```
COPY *.* B:
```

- DIR: The Directory command lists the names of all the files on a diskette. Parameters can be used in the command for selectivity. For example, to list only BASIC program files:

```
DIR *.BAS
```

- DISKCOPY: Makes an exact duplicate of a diskette, usually for backup. For a single-diskette system such as the *PCjr*, DISKCOPY requires no designation of diskdrive. On a two-drive system you would type DISKCOPY A: B:.
- ERASE a file or files: Makes room on a diskette for new files by getting rid of old files which are no longer needed (identical to the DEL command). The same options in the DIR command apply to ERASE. Thus, to erase all BASIC program files:

```
DEL *.BAS
```

- FORMAT a diskette: Required before any program can write any data on a diskette. Optionally, will write a copy of the DOS system on the new diskette.

```
FORMAT A:/S
```

- RENAME a file: Change the name of a file:

```
REN FILE1 FILE1.OLD
```

In addition, the following commands may be of interest:

- CHKDSK: Checks the diskette directory and analyzes how the space is being used. Since CHKDSK does not catch all types of errors, its main value is in showing how much diskette space has been used and how much space is available.

The *F* (fix) option will cause CHKDSK to create files for sectors which DOS has "lost." (These are sectors which are not part of a file, yet are not shown as available. When this happens, there is a possibility that a file or files have been damaged. Please read Chapter 11 about Backup if that sounds scary—it is!) Usually you will want to "ERASE *.CHK" to return these reclaimed sectors to the available pool.

- COMP: Compares two files.
- DISKCOMP: Compares two entire diskettes. Note that a "Compare error" message can occur even when the individual files on the two diskettes being compared are identical. This is because the files may be in different locations on the diskettes.
- PROMPT: Changes system prompt. PROMPT has a number of parameters to indicate special functions. A few of these are
  $d . . . Date
  $g . . . the greater-than symbol, >
  $h . . . backspace
  $n . . . the current disk
  $t . . . Time
  $_ . . . new line

  Example 1: The standard system prompt. (This is silly, since the PROMPT command with no operands does the same thing.)

  ```
  PROMPT $n$g
  ```

  Example 2: Display date and time, but backspace over year and 100ths of seconds to save space, then prompt on next line:

  ```
  PROMPT $d$h$h$h$h$h $t$h$h$h $_Using disk $n$g
  ```

- TYPE: Displays the contents of a file on the screen. The same function may be accomplished with: "COPY filename CON".

DOS offers many more commands. As you become familiar with the commands listed above, you may want to try others. One word of caution: always try out new commands with a diskette which has been backed up, or with one you are willing to lose.

10

Making DOS Go to BAT for You

The BAT (batch) feature of DOS can turn your *PCjr* into a useful tool for the whole family. It can do this by permitting anyone to get right to work on the computer without having to learn DOS commands at all!

To illustrate what BAT files can do, pick a program that will be used by more than one member of the family—BASIC, for example. Now format a diskette and write the DOS system on it. With the DOS system disk in the disk drive, type the following:

```
FORMAT A:/S
```

(Please read about "Using DOS Commands" for information on how to enter commands if you skipped that chapter. The Enter key is not shown here, but it must be pressed after each command.)

Follow the instructions which the system will display. After the Format program is complete, the DOS prompt will be back on the screen ("A>"). With the system diskette in the disk drive enter

```
COPY BASICA.COM
```

The fun comes after the copy is complete: setting up the diskette for your family. First, we will set up subdirectories for each person. Subdirectories isolate programs and data from each other. This means that different people can save their BASIC programs on the same diskette, without having to worry about using a file name that someone else may have used already.

For each person, make a subdirectory on the new diskette:

```
MD\FATHER
MD\MOTHER
MD\JONATHAN
MD\DEBBIE
```

You can check your work by using the DIR command; it should list COMMAND.COM, BASICA.COM, and the four directories.

Now create a file that will run automatically when the computer is turned on (or rebooted). When DOS first starts it looks for a file called AUTOEXEC.BAT. If there is such a file, DOS will read each record from it and execute any commands it finds there. You can use an editor to create all the BAT files used here, or you can do the following:

```
COPY CON AUTOEXEC.BAT
```

This tells DOS to copy keyboard input to a file. After you enter this command you will not get the familiar A> prompt: DOS is waiting for a different kind of input.

```
ECHO OFF
```

This tells DOS not to display each line of the BAT file as it uses it.

```
TIME
```

```
DATE
```

This will produce the Time and Date questions that you normally get when you start DOS.

```
ECHO Please enter the code (and a BASIC program named
ECHO if desired) that matches your name:
ECHO   Dad . . .                      DAD
ECHO   Mom . . .                      MOM
ECHO   Jonathan . . .                 JON
ECHO   Debbie . . .                   DEB
```

The Echo command makes DOS display the rest of each line.

Now, press the *Fn* (Remember: To activate a Function key on the *PCjr,* you must strike the *Fn* key first) then the F6 key: Z will appear on the screen. Then press Enter. DOS will write to the diskette and report that it has copied one file. The familiar A> prompt should be back.

The DIR command will show you that, indeed, AUTOEXEC.BAT now exists.

You can test your work so far by typing AUTOEXEC. You should get Time and Date commands, and the messages that you entered.

Now for the BAT files that really do the work. Each will be pretty much alike:

```
COPY CON DAD.BAT
```

Make a file called DAD.BAT from console input. After this command is entered, the A> prompt will not appear.

```
CD\FATHER
```

This DOS command makes FATHER the current directory, so DOS will look there first for any commands. Note that the CD command is not executed now, but will be copied into the BAT file we are creating.

```
PATH\
```

This command tells DOS where to look if it cannot find a command in the current directory. In this case, we specify only the root directory.

```
BASICA %1
```

This will start BASICA for Dad. Furthermore, if Dad typed a BASIC program name after DAD, that program will be loaded and run by BASICA.

```
REM Have you backed up your work?!
```

The REMark command is used to issue a friendly reminder.

Now signal DOS that you are done by first pressing *F6*, then Enter. If you like, you can check your work by entering DAD. You should end up in BASIC if you have typed the commands correctly. Use the SYSTEM command to return to DOS. After you leave BASIC, you should see, "Have you backed up your work?"

Be sure to return to the root directory with the command CD\.

You can create similar BAT files to match each of the "codes" listed by AUTOEXEC.BAT. When you are all done, use the Diskcopy command to make a duplicate of the family BASIC diskette. (Read about "Backup" for more information.)

That was a pretty simple example, but it should help to give you an idea of how you can make DOS go to BAT for you!

11

Backup

This chapter will be as brief as we can make it. Not because the subject can really be covered thoroughly in a short space, but because a short piece is more likely to be read than a long one . . . and backup is something that you need to know about and to do.

The method chosen to back data up depends on what it is you wish to protect against. We offer the following scenarios:

Complete loss. A visiting alien takes your computer apart to make a space phone. You can get a new computer, but the diskette left in the computer has been shredded. (If that's too improbable, how about, "I just spilled coffee all over my data diskette," or, "My disk drive seems to have laid grooves across my spreadsheet program diskette: I guess all that red powder is bad news, huh?") For this kind of situation, VOLUME RECOVERY is required.

File loss. A file is accidentally erased, or mysteriously becomes unreadable. (Both have happened to us . . . more than once!) FILE RECOVERY is needed.

Elective recovery. A spreadsheet has just been recomputed and saved but you realize you've forgotten to print reports based on the old data. FILE RECOVERY again, but some of the implications are different.

Objective

Backup and recovery procedures must be fail-safe and must require a minimum of time, effort, and thought. You don't want to have to improvise when disaster strikes. In fact, standard backup procedures turn what might have been a disaster into a minor, temporary, sidetrack event. That's the goal: to make this kind of thing less exciting.

We have rarely had to perform a recovery process, but our work is backed up in one way or another every time the computer is used. In fact, backup procedures are used many times during each computer session. Therefore, we want a backup process that is simple enough to become second nature.

Backup for Volume Recovery

The DISKCOPY command makes it easy to make an exact duplicate of an entire diskette ("volume"). However, running DISKCOPY takes time, and means an extra step apart from the program in use. Therefore, DISKCOPY should be used only when a major change is made to a diskette. The change might be a restructuring of the subdirectories, or simply changes to several files on the diskette. Essentially, DISKCOPY should not be the major backup method, but it does provide a *starting point* for volume recovery.

File Backups

Every time a file is changed we produce two copies of it, on two different diskettes. There are two ways of doing this. The *safer* method, and the one we usually employ can be described as follows.

Save work twice at the same time, once to one diskette, and once to the other. This is generally done many times during each computer session at convenient points and should become second nature. The extra time involved in writing the file a second time is the price of such an insurance policy.

Using a different diskette for each program worked with should become a habit. We maintain separate diskettes for each spreadsheet, word processor, and database system used. While this may not be necessary, it seems to help keep things straight.

For example, from MultiPlan, the sequence: T S <Enter> will save a file. The next thing to do is to switch diskettes and enter the command again.

The alternative is to use the DOS COPY command after the program session ends. This has the dual drawbacks of not providing immediate backup, and of requiring extra steps aside from the program in use. It is preferable to use COPY only when making a *third* copy (for elective recovery or for transportation between machines).

Elective Backup

Backup for elective recovery may be merely a matter of convenience: a process that may save some time if you need to undo something. The easiest way to do this is to rename either the old file or the new, updated, version of it. It makes little difference which approach you use; we file the new version under a new name.

File Recovery

Individual file recovery is a matter of simply restoring from the backup, using DOS COPY. However, if a file has become unreadable for no apparent reason, make another copy of the suspect diskette, as it has probably become unreliable. Do this immediately: it is more important to feel secure with the data than to continue to work on it without proper backup.

Elective Recovery

One form of elective recovery consists essentially of renaming an old version with the name of the latest version of a file. The only question is, "What do I do with the latest version?". The answer depends on circumstances. If you are going back simply to rerun a piece of work (for example, printing an old version of a spreadsheet), you will probably want to keep the latest version intact. This can be done by renaming it before renaming the old version. On the other hand, if you are backing out a unit of work (say, you updated the sheet with the wrong figures), then the latest version is no good and might as well be erased.

Volume Recovery

Volume recovery assumes that DISKCOPY was used to create two identical diskettes, and that careful operating procedures have kept them identical ever since. (We suggest that after you have been using two diskettes this way for a while, you try the DISKCOMP command to check yourself. If DISKCOMP fails, try individual file comparisons (COMP), as the files may be identical but on different places on the diskette. If individual file comparisons fail, then you have not been careful enough.)

Replace the lost diskette by copying the backup. The message here is simply, "Two copies, always."

Conclusion

Backup/recovery not only saves data, it can save an even more precious commodity: time. Backup procedures should be a normal everyday part of computer activities. By the way, in the middle of "File Recovery" we had a real, honest-to-goodness power failure—it lasted for two hours, and we lost only half a paragraph. If disaster strikes you, we hope your damage-control is as successful!

12

Communications

This chapter, at the outset, assumes you have acquired the *PCjr* Model 67 with the internal modem. With these givens, you have one disk drive and some blank formatted diskettes.

IBM's Personal Computer *Personal Communications Manager* is the "supported" communications software for the *PCjr.*

It will allow you to "talk" with other computers over the telephone lines. This program is highlighted by a number of sophisticated features including an electronic mail management system that will actually schedule the sending of mail through the computer. It will act as a demon dialer, review messages your computer has received, and support two transmission speeds: 300 and 1200 *baud.* Those baud rates mean simply 300 or 1200 bits transmitted every second. Exactly how many characters per second this translates to depends on some of the transmission options you have chosen. The character rates are usually around 30 and 120 characters per second respectively. (Details about computer communications will be found in the Communications Appendix.) The *PCjr* internal modem does not operate at 1200 baud. You will need an external modem to take advantage of higher speed communications.

You should be aware of several selectable, somewhat technical features incorporated in *Personal Communications Manager.* It's a good idea to know what they do and when to utilize them.

Parity is one of these features. When accessing (that's computerese for telephoning) an electronic bulletin board system, it is necessary to set the correct parity so that *PCjr* and the host computer (the one you are accessing) are in sync. For most of the IBM Bulletin Boards, parity is set at E/N/1, meaning, paradoxically, eight data bits/*no* parity/one stop bit. Other bulletin boards, along with Compuserve^{tm} and The Source^{TM}, operate at 7/E/1, meaning seven data bits/*Even* parity/one stop bit. Sometimes, when calling a public bulletin board, you won't have to worry about parity, as the system will adjust to your transmission.

Full and half duplex are simply fancy words for the facility of seeing the characters you send from your computer on the screen. In the half duplex mode, *PCjr* is actually echoing the characters you key in on the screen. The host computer, however, must know that you are operating in the mode, or you will see double characters for each one you type. In the full duplex mode, the host is relied upon to "echo" characters to the *PCjr*. Full duplex is most often used in communications networks because you can see what the host is actually receiving, rather than what you are sending. This is important, because interference on the telephone line may cause distortion of transmission. If you see distorted or "garbage" characters (you'll know "garbage" when you see it), you will know you must retransmit. Most such networks, such as Compuserve^{TM} and the bulletin boards, assume you are in full duplex, so in most cases you won't have to be concerned with switching.

Incidentally, if you see nothing at all once you have connected to another computer, there is generally no need to worry. Just switch the duplex setting with a keystroke and you'll be in business.

With *Personal Communications Manager* you will also be able to filter out certain transmitted characters and control the flow of transmission. These features are somewhat advanced and should be used with care. Please consult your documentation for further information.

Personal Communications Manager will allow you to define the *PCjr's* Function Keys, so that entire strings of data can be executed with a single keystroke. For example, you might define the *F1* key to dial the telephone number of your favorite bulletin board, transmit a *logon* command, and enter your password. Once set up, simply pressing the *F1* key will do the work of dozens of individual keystrokes. Although you can define the function keys to do anything, this feature is especially useful when signing on to bulletin boards or data bases that require your name, user identification, and password. You can define other keys to sign off (or *logoff,* whichever command is used), to list messages waiting for you (if that is a feature of the service you are using), and so on.

Many of the public domain electronic bulletin boards listed in this chapter offer free software. Much of this public domain software is of a high quality and it is free. This book will deal with a few of these programs, but for now, suffice it to say that they exist and with *Personal Communications Manager* you can easily download them to your computer. *Download* means simply to transfer data from the host computer to your *PCjr.*

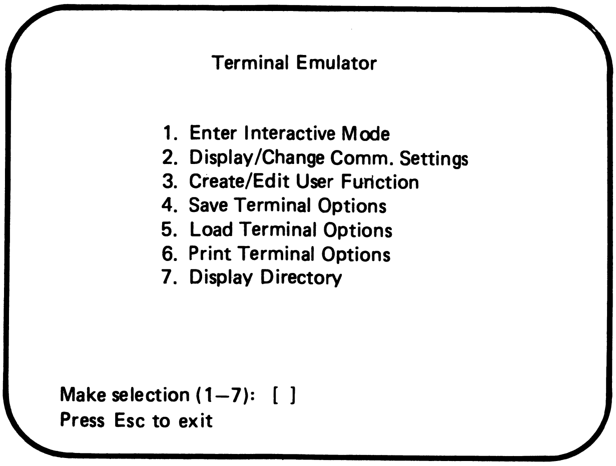
In order to accomplish this marvelous feat, simply dial up the bulletin board with *Personal Communications Manager.* There are three ways to dial:

1. Use the Dial command (Alt-D followed by the telephone number).
2. Define a function key in an existing terminal options file to generate the dial command and telephone number.
3. Create a new terminal options file with an entirely new set of functions.

The documentation for *Personal Communications Manager* is quite extensive and detailed. If you're looking for a quick start to a bulletin board, we'll work you through the dialing procedures here. However, to use all the features of this piece of software fully, you will need to read the documentation carefully.

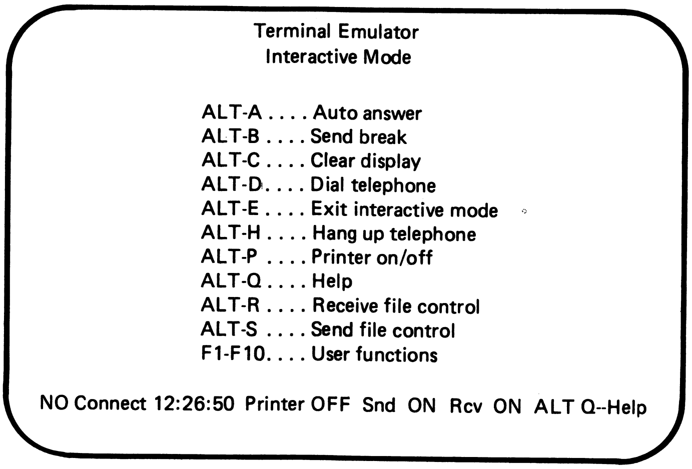
With *Personal Communications Manager,* connecting to a bulletin board system is rather simple. From the Main Menu, select the Terminal Emulation Mode, number 1. At this point, you will see another menu.

Here you will strike *Alt + D* for Dial Telephone. Do this by holding the *ALT* key and striking the *D* key. A new screen will appear and you will be prompted to enter the telephone number. Hit *enter* after the phone number; your modem, if it is of an auto dial nature, will dial up the bulletin board. From that point, you are home free. At the bottom of the screen is a "status line," that will indicate connection to the "host" you have dialed.



```
                    Terminal Emulator


              1. Enter Interactive Mode
              2. Display/Change Comm. Settings
              3. Create/Edit User Function
              4. Save Terminal Options
              5. Load Terminal Options
              6. Print Terminal Options
              7. Display Directory




   Make selection (1—7):  [ ]
   Press Esc to exit
```

To make your call, select number 1, "Enter Interactive Mode," and the following will be displayed:



```
                    Terminal Emulator
                     Interactive Mode

              ALT-A . . . . Auto answer
              ALT-B . . . . Send break
              ALT-C . . . . Clear display
              ALT-D . . . . Dial telephone
              ALT-E . . . . Exit interactive mode
              ALT-H . . . . Hang up telephone
              ALT-P . . . . Printer on/off
              ALT-Q . . . . Help
              ALT-R . . . . Receive file control
              ALT-S . . . . Send file control
              F1-F10. . . . User functions

 NO Connect 12:26:50 Printer OFF Snd ON Rcv ON ALT Q--Help
```

Once connected, follow the instructions displayed on your monitor by the host system. On a Hostcomm™ (Hostcomm is the name of the software used to run the bulletin board) you will be required to enter your first name, then your last name and the city and state in which you live. Hostcomm™ will also ask you if you need linefeeds or nulls. Answer "no" to both queries. You'll generally have to enter a password; in this system the password will most often be IBMPC. Now you might get a few messages and prompts, telling you what to do next. Ultimately, you will reach a menu with various selections: select "Download a file." At this point a list of files (or programs) will be displayed.

One of the primary reasons for accessing the bulletin boards is to obtain "free" software. This can be accomplished with *Personal Communications Manager.* Again, follow the host's prompting for downloading procedures, after you have selected a file. As to *Personal Communications Manager,* in order to open a file to receive the transfer, hold the *ALT* key down and strike the *R* key. You will be prompted for a filename, which may be any valid DOS (see Chapter 7) filename. Hit return and you're on your way. When the transfer is completed (you'll know from your screen when it is), hit *ALT+R* again, and *E* for "end" at the "Pause or end (p/e)?[ ]" *Personal Communications Manager* prompt. Don't forget to exit the Hostcomm™ system, or any other bulletin board system, properly. Work your way through the menus until you reach a selection that offers a *Bye, Quit,* or *Terminate* choice. You could cause your host and yourself some problems if you leave abruptly by switching off your modem or computer.

Personal Communications Manager will also allow you to handle electronic mail. The electronic mail function is based on setting up a number of "mailboxes" on your system (on diskette). You can send mail to these mailboxes all day long, then connect via telephone to transmit the accumulated mail. When connected to another *PC* or *PCjr* which also has *Personal Communication Manager,* any mail waiting for you will be sent to you. For even more power, you can set up the program to initiate phone calls automatically at a time of day you choose (in the middle of the night, for example, when rates are lowest).

You can use the electronic mail feature just as you would the U. S. Postal Service. However your letters will be delivered almost instantaneously. This assumes, of course, that you are sending "mail" to another computer with a similar system, or to MCI Mail™ or the Postal Services E-Com™. That's right, the *PCjr* will allow you to avail yourself of those services.

With *Personal Communications Manager,* you can send and receive any computer files based in DOS. These might include spreadsheets from MultiPlan™, various charts and graphs, or reports and other computer programs.

If you are expanding your *PCjr* system, or if you decide not to purchase IBM's internal modem and *Personal Communications Manager,* there are other ways of establishing computer communications.

External modems are readily available, with the preeminent brand for the IBM being the Hayes Smartmodem™, which is available in both 300 baud and 300/1200 baud. The latter is the more expensive of the two (but well worth it, we feel, as it is apparently becoming the standard). An external modem will require a special cable to plug into your *PCjr.* You should pick up the communications cable when you get your modem. The modem usually comes with a standard telephone wire, so that it can be connected to your telephone. If you do not have so-called "modular" telephone equipment, you may have to get an adapter. Check with the salesperson when you buy your modem and communications cable.

Once hooked up, simply switch the modem on and begin communicating . . . if you have a software package with which to do so. Fear not—if you didn't purchase the IBM package (even if you did, you might want to try this)—there is FREEWARE™. FREEWARE publishes a program called *PC-TALK III™*, which has become the most used and among the most sophisticated communications programs available for the IBM *PC;* it works on *PCjr.* It would be appropriate to point out here that the authors are in no way connected with FREEWARE, but believe in its concept. Much of this book was transferred between our computers using *PC-TALK III;* generally, it is the only communications program we use.

From the PC-TALK III documentation (instructions) comes the following:

> FREEWARE user-supported software is an experiment in distributing computer programs based on three principles:
>
> First, that the value and utility of the software is best assessed by the user on his/her own system. Only after using a program can one really determine whether it serves personal applications, needs, and tastes;
>
> Second, that the creation of independent personal computer software can and should be supported by the computing community.

> Finally, that copying and networking of programs should be encouraged, rather than restricted. The ease with which software can be distributed outside traditional commercial channels reflects the strength, rather than the weakness, of electronic information.

From the same documentation, how to obtain PC-TALK III:

> Anyone may request a copy of a user-supported program by sending a blank, formatted diskette to the author of the program. An addressed postage-paid mailer must accompany the disk (no exceptions, please).

A copy of the program, with documentation, will be sent by return mail. The program carries a notice suggesting a contribution to the program's author. Making a contribution is completely voluntary on the part of the user.

Regardless of whether a contribution is made, the user is encouraged to copy and share the program with others. Payment for use is discretionary on the part of each subsequent user.

Now, you can go ahead and send a blank, formatted diskette to Freeware, The Headlands Press, Inc., Post Office Box 862, Tiburon, CA 94920. We feel that a better way, notwithstanding the "free" part of this deal, would be to send a check for $35.00 to that address. This saves you the hassle of finding mailers and sending disks . . . and might even get you quicker service. However you handle it, *PC-TALK III* is a superb software package, coming with some 70 pages of documentation that prints out right to your printer (which may be a problem, of course, if you don't own one). You can always share your *PC-TALK III* with a friend who has a *PCjr* and a printer or a *PC* with printer.

With *PC-TALK III* you will have few, if any problems. The documentation, though long, is excellent. Read it and you'll be up and running in no time.

Although *PC-TALK III* does not have *Personal Communications Manager's* built-in electronic mail feature or operate in color (on a color monitor or television set you will see white text on a black background), it does encompass everything else in the commercial program. On the other hand, it does have the XModem file transfer protocol, which *Personal Communications Manager* doesn't have. The XModem protocol is a method of transferring files between two computers in a virtually error-free manner. It is widely used in personal computer communications and necessary for the transfer of certain files. This sort of "protocol" transfer is discussed at some length in the Communications Appendix.

Generally, because of its ease of use, wide variety of functions, and reliability, *PC-TALK* is thought to be the best and most substantive communications program for the IBM *PC;* it will probably be considered so with *PCjr.*

PC-TALK actually comes in two versions: One may be run if you have 128K or more of memory, the other if you have 64K. What this means is that if you acquired the *PCjr* Model 4, with 64K, you will have to run the 64K version of *PC-TALK*. To do this, cartridge BASIC is required. With *PCjr* Model 67, the 128K version of the program may be run *without* cartridge BASIC. The primary difference in the two versions is the speed of operations. Performance on the 128K version is definitely faster, although both versions are precisely the same in function.

To actually run the program, start *PCjr* with DOS, then swap diskettes and type either TALK64 or TALK128. In Chapter 7, "What Do DOS Do?" you learned to create what is known as a "batch" file. Using this function of the computer will allow you simply to use the *PC-TALK* diskette.

The authors feel that unless you require the electronic mail feature of *Personal Communications Manager* or find color an absolute necessity, *PC-TALK* is more than adequate for communications . . . and you really can't beat the price.

There are two other commercial communications programs worth mentioning here, TELIOS (Genesys Corporation, Rockville, MD) and CROSSTALK XVI, Ver. 3.4 (Microstuf Corporation, Atlanta, GA). Again, neither has the electronic mail facility of *Personal Communications Manager;* however, both will operate in color and support the XModem protocol discussed earlier. Both operate on *PCjr* with facility, but both are somewhat more complex. We mention them only to illustrate the variety of communications programs available for *PCjr*, and these in particular because they are well tried and have decent track records.

The following is a list of public domain bulletin board systems running for the IBM system. The authors assume no responsibility for your experience with them. Those marked with an asterisk represent the most popular. This list was generated December 31, 1983, when all of the following were known to be operating.

IBM PC Bulletin Boards

(201) 531-7268 Ocean, NJ SYSOP: Wilbur Streett
(24 hrs, 300/E/7/1)

(201) 678-6670 NYC, NY SYSOP: Donald David HOSTCOMM
(24 hrs, User Group software exch, Password = IBMPC)

(203) 966-8869 New Canaan, CT SYSOP: Whit Wyant RBBS-PC
300/1200 (24 hrs, messages, upload/download, no password)

(203) 521-1991 CT SYSOP: John O'Boyle BBS
300/1200 (24 hrs, upload/download, conference, messages)

(214) 223-0983 Desoto, TX SYSOP: Mark Collar RBBS-PC
300/1200 (24 hrs, XMODEM xfer, messages, bulletins)

*(215) 439-5696 Allentown, PA SYSOP: Glenn Wesley RBBS-PC
300/1200 (4:30 p.m.–8:00 a.m., upload/download, messages)

(219) 255-8803 South Bend, IN SYSOP: Terry Alley RBBS-PC
300/1200 (24 hrs, XMODEM upload/download, messages)

(303) 690-4566 Denver, CO SYSOP of PC sect—Chris Carson
(24 hrs, Tips, download/upload, no password)

(303) 223-8342 Fort Collins, CO SYSOP: Leroy Casterline
RBBS-PC
300/450/1200 (24 hrs, upload, download, Msgs, 'C', XMODEM)

(304) 344-8088 Charleston, WV SYSOP: Bob Ketcham
RBBS-PC
300/1200 (24 hrs, download, upload, messages, no password)

*(312) 882-4227 Chicago, IL SYSOP: Gene Plantz RBBS
300/1200 (24 hrs, upload, download, messages, no password)

(313) 761-1399 Ann Arbor, MI SYSOP: RBBS-PC
300/1200 (24 hrs, messages, XMODEM)

(314) 741-8655 St. Louis, MO SYSOP: Karl Krummenacher
RBBS-PC
300/450/1200 (24 hrs, upload, download, Msgs, XMODEM)

(318) 688-7078 Shreveport, LA SYSOP: Sam Holoviak
RBBS-PC
300/450 (24 hrs, messages, XMODEM files xfer)

(319) 332-7648 Dubuque, IA SYSOP: Jeff MacHusak RBBS
300/1200 (24 hrs, messages, upload/download)

(319) 363-3314 IA SYSOP: Ben Blackstock RBBS-PC
300/1200 (24 hrs, messages, upload/download, no password)

(404) 252-9438 Atlanta, GA SYSOP: Rod Roark MLBBS
300/1200 (24 hrs, messages, download/upload, no password)

(404) 926-8411 Woodstock, GA SYSOP: Ken Shackelford
RBBS-PC
300/1200 (24 hrs, download, upload, messages, no password)

(415) 845-9462 Berkeley, CA SYSOP: John Carmichael
(24 hrs, password = GUEST)

(415) 861-5733 San Francisco, CA SYSOP: Harry Logan
RBBS-PC
300/1200 (24 hrs, XMODEM, messages, no password)

(415) 689-2090 Concord, CA SYSOP: Jon Martin
ML+RBBS-PC
300/450/1200 (24 hrs, upload, download, Msgs, XMODEM)

(415) 481-0252 San Lorenzo, CA SYSOP: Terry Taylor RBBS-PC
300 (24 hrs, XMODEM upload/download, messages)

(416) 499-7023 Toronto, Canada SYSOP: Doug Peel
HOSTCOMM
(24 hrs, PCanada/E-Mail/Ads ID# PC250 PW = IBMPC)

(507) 281-0970 Rochester, MN SYSOP: Alfred Anderson
RBBS-PC
300/1200 (24 hrs, 20 Mb hard disk full of goodies)

(608) 262-4939 Madison, WI SYSOP: Read Gilgen RBBS-PC
300 (evenings & weekends, messages, downloads, tips)

(701) 293-5973 Fargo, ND SYSOP: Loren Jones RBBS-PC
300/1200 (24 hrs, messages, upload/download)

(703) 680-5220 Dale City, VA SYSOP: Tim Mullins
(24 hrs, news, new product review—all PC's)

(704) 873-5140 Statesville, NC SYSOP: Mac Wiley RBBS-PC
(9 p.m.–8 a.m. M–TH, 24 hrs: 9 p.m. Fri–8 a.m. Mon)

(704) 364-4311 Mathews, NC SYSOP: Charles McCurry/Scott Totaro
300/1200 (24 hrs, download, General Information)

(806) 353-7484 Amarillo, TX SYSOP: Dorn Stickle RBBS-PC
300/1200 (6 p.m.–8 a.m., messages, XMODEM, no password)

(913) 841-6424 Lawrence, KS SYSOP: Bruce Anderson RBBS-PC
300 (10 p.m.–6 a.m., XMODEM upload/download, messages)

(914) 221-0774 Hopewell Junction, NY SYSOP: John Giberson RBBS
300 (24 hrs, XMODEM upload/download, sports, messages)

(918) 749-0718 Tulsa, OK SYSOP: Lynn Long RBBS-PC
300/1200 (24 hrs, c language, download, upload, etc.)

(919) 847-4625 Raleigh, NC SYSOP: Randy Ray HOSTCOMM
300/1200 (24 hrs, Public domain software, Password = IBMPC)

*(703) 560-0979 Annandale, VA SYSOP: Wes Merchant CONNECTION-80
300 baud (24 hrs, Capital PC message center)

(703) 321-7003 Vienna, VA SYSOP: Richard Cunningham RBBS-PC
300/1200 (24 hrs, Capital PC C/UNIX SIG)

*(703) 978-9592 Fairfax, VA SYSOP: Don Withrow HOSTCOMM
300/1200 (24 hrs, Capital PC BASIC SIG, Password = IBMPC)

(703) 759-5049 Great Falls, VA SYSOP: Tom Mack RBBS-PC
300/1200 (24 hrs, Philosophy, Religion, XMODEM xfer, etc.)

(703) 425-7229 Springfield, VA SYSOP: Bob Blackwell HOSTCOMM
300/1200 (24 hrs, upload/download assembly, Password = IBMPC)

(703) 237-4322 McLean, VA SYSOP: Bob Jueneman RBBS-PC
300/1200 (6:30 p.m.–7:30 a.m., weekends, cryptography)

(703) 522-4513 Vienna, VA SYSOP: Paul McKnight HOSTCOMM
300 only (24 hrs, upload, download, tips, Password = IBMPC)

(703) 370-8893 Oakton, VA SYSOP: Greg Gallagher HOSTCOMM
300/1200 (Irregular hours, topic discussions)

(703) 560-7803 Vienna, VA ABBS with IBM PC Conference ABBS
300 (24 hrs, messages, download, upload, no password)

(301) 424-5817 Potomac, MD SYSOP: Doug Thompson HOSTCOMM
300/1200 (24 hrs, Capital PC Monitor article upload only)

*(301) 949-8848 Rockville, MD SYSOP: Rich Schinnell HOSTCOMM
300/1200 (24 hrs, CPC Software Exchange SIG, Password = IBMPC)

*(301) 251-6293 Gaithersburg, MD SYSOP: Larry Jordan RBBS-PC
300/1200 (24 hrs, Capital PC & communications info)

(301) 460-0538 Bethesda, MD SYSOP: Ramona Landberg RBBS-PC
300/1200 (24 hrs, Capital PC Federal/Medical SIG)

(301) 924-5323 Rockville, MD SYSOP: Eileen Rodgers RBBS-PC
300/1200 (evenings & weekends, Capital PC CEAM SIG)

(301) 596-3569 SYSOP: Jorge Del Pinal RBBS-PC
300/1200 (24 hrs, Capital PC Statistics SIG)

(301) 267-4930 Annapolis, MD SYSOP: Vince Castelli RBBS-PC
300/1200 (24 hrs, Capital PC XT-SIG, messages, XMODEM)

*(301) 948-9143 Gaithersburg, MD SYSOP: Jim Fry RBBS-PC
300/1200 (24 hrs, Capital PC best local buys info)

(301) 972-2456 Germantown, MD SYSOP: John Mac Evoy RBBS-PC
300/1200 (24 hrs, RINGBACK [see below], Scuba diving, etc.)

(301) 946-6790 Wheaton, MD SYSOP: Dan Cameron RBBS-PC 300/1200 (24 hrs, electronics buying & info, download)

RBBS's require three carriage returns to speed and parity detect after CONNECTION before they will respond.

HOSTCOMM's will respond with a connect message after switching baud rate to match your system; they will detect and switch parity to match your system automatically.

RINGBACK means you call the number and let the telephone ring only *once*. You then call back within 50 seconds, and the bulletin board will answer on the first ring. If you let the phone ring more than once the first time, you will get a very annoyed SYSOP on the phone talking to your modem!

PARAMETERS: The above systems operate with NO PARITY, and 8 DATA bits. Even parity can be used only with RBBS-PC systems, but is not recommended.

If that list isn't enough for you, again, don't worry—electronic bulletin boards proliferate like the proverbial rabbit family. Many of those listed will themselves have listings of virtually hundreds more throughout the country. If *you* want to set up a bulletin board system—at this writing, you can't with the *PCjr.* By the time you read this book, second disk drives for the *PCjr* may well be available, but we'll talk about that in Chapter 24, "The Future of *PCjr.*" There are a couple of ways to set up a bulletin board system, incidentally; one of them is user-supported software. That bulletin board system is called RBBS, Remote Bulletin Board System, and can be downloaded from many of those listed. Hostcomm™, which is much more than a bulletin board system, must be purchased. You'll find a listing for it in the Appendix. As of this writing, it has not been tested on *PCjr.*

A number of commercial data bases are available to you and your *PCjr,* notably Compuserve®. The Source™ and Dow Jones News/Retrieval®. You must sign up for each of these and receive a personal password. For the privilege of using them, you will be paying a minimum of $5.00 per hour.

These commercial data bases will give you an extremely broad range of selection. Compuserve, for example, has at least 40 games you can play while connected, an air travel service, brokerage and banking facilities, financial data on the stock and bond markets, an on-line encyclopedia, various newspapers from around the country, magazines, real estate ads, special interest groups involving computers, working at home and other subjects, personal finance programs, health reports, and major wire service news, to name a few of the hundreds of selections. The Source is similar, while Dow Jones offers primarily financial news.

13

Homeword®

If the phrase "word processing" frightens you, trust us: *HOMEWORD* may be just what you need to get your feet wet. One of your authors spent the better part of his working life slamming his fingers into an old Royal 440; after trying several word processing systems, he's finally comfortable with *HOMEWORD.*

Though *HOMEWORD* isn't what one would term a "professional" or "business" word processor, it has a relatively high degree of sophistication and is extremely easy to use. In the parlance of computers, it's more than "user-friendly"; it's downright cuddly.

In case you are not quite sure what word processing is, think of it as simply typing on a computer and being able to accomplish all you can do at a typewriter with greater dispatch and ease than you ever dreamed of. For example, you don't need to "x" out a word in word processing; you simply type over it, leaving no trace of the incorrect version on the screen.

To start with, everything in *HOMEWORD* is operated from menus. This is quite a plus for the beginner as all you need to do is select a function from a given menu, such as deleting a block of copy or moving a paragraph. Moreover, this selection can be done with a joystick.

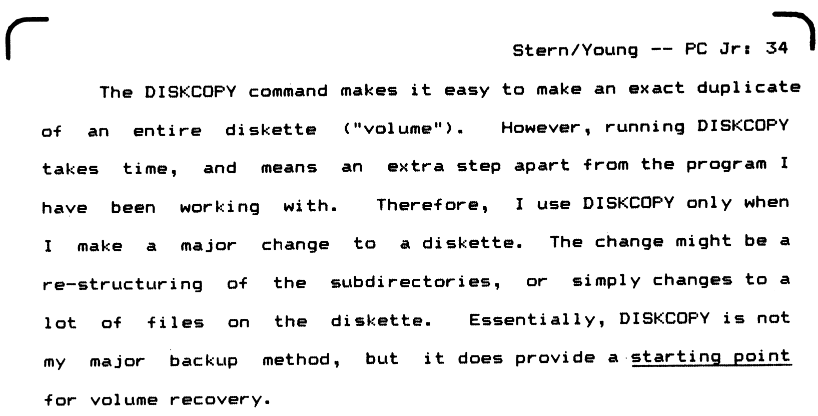
Although menus tend to slow you down, once you are an experienced *HOMEWORD* user, you can bypass the menus and go directly to any function from the *text entry* mode. *Text entry* is the word processing term for typing. In word processing, generally, and specifically in *HOMEWORD*, you need not hit the return (or enter) key unless you are ready to designate a new paragraph. *HOMEWORD*, and most word processors, use what is known as *wordwrap*—you just keep typing to the end of the line and the line "wraps" automatically to the left margin.

With *HOMEWORD*, what you see on the screen is precisely what will be printed out on the printer. If your first three paragraphs are double-spaced, the next two single-spaced, and the sixth an indented single-spaced quote on the screen, that's what you'll get in print. This screen layout is continually updated as you insert or delete or move paragraphs around.

When you insert a character, word, sentence or entire block of text within existing text, you will see subsequent characters moved to the right or down if the line continues past the established margins.

HOMEWORD allows the user to move blocks of text around the document with just a few keystrokes. For example, if you are working on a 20-page report, and find that you wish to move a paragraph from page 8 to page 19, with *HOMEWORD* you simply select the Find and Replace icon menu and follow the prompts. More on the specifics a bit later.

The global and selective search and replace facilities of *HOMEWORD* are impressive features. They allow the user literally to replace a word, phrase, or character with a couple of keystrokes. In our mythical 20-page document assume we have the name C. F.



Stern/Young -- PC Jr: 34

The DISKCOPY command makes it easy to make an exact duplicate of an entire diskette ("volume"). However, running DISKCOPY takes time, and means an extra step apart from the program I have been working with. Therefore, I use DISKCOPY only when I make a major change to a diskette. The change might be a re-structuring of the subdirectories, or simply changes to a lot of files on the diskette. Essentially, DISKCOPY is not my major backup method, but it does provide a starting point for volume recovery.

Header

Martin repeated throughout, perhaps 15 times. C. F. Martin, however, has been replaced by R. J. Fruitstand. Using the global search and replace function, R. J. is substituted for C. F. in a matter of seconds. But suppose that C. F. is still around and we'd just like to replace him in certain instances. *HOMEWORD*'s search and replace function may also be used selectively. This, of course, allows us to choose the occurrences of replacement.

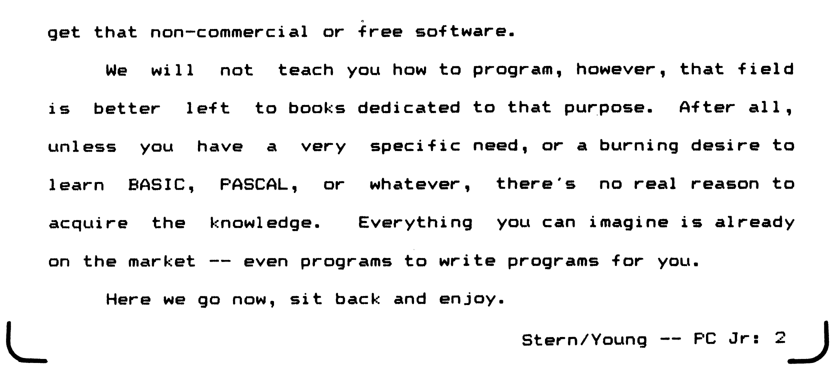
In addition, *HOMEWORD* has all the functions of a typewriter, such as adjustable margin settings and tab stops.

The right margin may be *justified* (straight) or ragged right (like a typewriter margin) by using the formatting functions of *HOMEWORD*.

Additional functions include the ability to create a new page anywhere in your text, line centering, and the printing of headers and footers. *Headers* and *footers* are terms meaning the software will print a phrase at the top (header) and bottom (footer) of each page. This function might be used in producing manuscripts, such as this one—the manuscript header was *Stern/Young:PCjr.* Headers and footers may be printed with *HOMEWORD* in conjunction with automatic page numbering.

To use *HOMEWORD*, a disk drive and a printer are required. Additionally, it requires 128K of memory. Any kind of monitor may be used, even a television set; *HOMEWORD* does display in color.

HOMEWORD requires the *PCjr*—in its first release it will not run on the IBM PC, primarily because of differences between the two machines in the graphics.



```
get that non-commercial or free software.

     We  will  not  teach you how to program, however, that field

is  better  left  to books dedicated to that purpose.  After all,

unless  you  have  a  very  specific need, or a burning desire to

learn  BASIC,  PASCAL,  or  whatever,  there's  no real reason to

acquire  the  knowledge.  Everything  you can imagine is already

on the market -- even programs to write programs for you.

     Here we go now, sit back and enjoy.

                                        Stern/Young -- PC Jr: 2
```

Footer

As you learned to do in Chapter 10, "Making DOS Go to BAT for You," format a blank diskette and set it aside for a moment. We'll be using it with *HOMEWORD.*

To get started, simply boot the program; that is, slip the *HOMEWORD* diskette in the disk drive and turn on (or "power on") the machine. You'll be prompted for the month, day, year, and the time, using a 24-hour clock. Once you have entered the dates and time, *HOMEWORD* will ask you to replace the program diskette with a formatted document disk. Notwithstanding the documentation for *HOMEWORD,* it is definitely not recommended that you use the program diskette for your data.

You will note that a "Backup Diskette Certificate" comes with your *HOMEWORD* package. The program diskette you receive with the package is "copy protected"; that is, the publishers of the software configured the diskette so that you cannot use the DOS "diskcopy" or "copy" commands to make a backup. Therefore, if you destroy or wipe out your original—not an unheard of occurrence—you're out of luck until you send $20.00 to IBM and wait six weeks for another copy.

Back to *HOMEWORD:* Following your initial time and date entries, you'll see some fancy graphics and then a prompt on your screen. You'll find the screen divided into three parts: the typing area, the icon area, and an outline of the document with which you are working. That outline will show you the page you are currently on along with its format—margins, spacing, etc. Pressing the ESC key now will remove that outline and show all six icons, representing different functions of *HOMEWORD.*

The icons include *print, edit, file, layout, preset values,* and *Exit to DOS.* The *Print* icon allows you to print your document on a printer, setting the page number for the first page and displaying on the screen a facsimile of your document as it appears on paper.

Selecting the *edit* icon will allow you to move, copy and erase text. In addition, using this icon will allow you to search for and replace certain words or phrases.

The *layout* icon is used for defining various formats, such as margins, tabs, spacing, and straight or uneven margins. Moreover, this selection will allow for choice of type fonts, or print styles, and for page numbering.

The *dial* icon, which represents preset values, is used for setting the "default" or regular values you desire whenever you start *HOMEWORD*. In other words, you may set the margins, line spacing, and tabs only once and they will remain "written" to the diskette, coming up each time it is used. These "defaults" of course, may be changed at any time.

Finally, the *exit to DOS* icon is just that: it exits to the Disk Operating System which, of course, you may use to format disks or copy your data.

We will here take you briefly through *HOMEWORD*'s operations. If you don't own the program, the following will give you a pretty good idea of what the package is all about; if you do own it, this text will give you a quick start. Please consult your documentation for specifics.

First, press the ESC key and you'll see the blinking cursor in the typing area. Remember, the only time you can type is when that cursor is winking at you from the typing area. If the cursor is a small box, rather than a line, you are in the insert mode. It is the insert mode you are in now. This means anything you type will be inserted, rather than replaced.

Type a line or two, then move the cursor to any space on the line and begin to type again—you'll see your new text being inserted, pushing the existing text to the right.

Pressing the *INS* key will put you in the replace mode; the cursor will change to a short line. Now, repeat the previous exercise and you will see your existing text being erased and replaced by the new text. Strike the *INS* key again to return to the insert mode.

Now, clear the screen of icons by pressing the *Fn* key, then the *F1* key. The same procedure is used, incidentally, to bring the icons back to the screen. We just want a clear screen to begin to type a letter.

First, please type the following:

Dear Marvin:
Our recent acquisition of your Gardens has benefited our monopolistic tendencies.

We intend to continue our purchases until all of the Broadway and Park Place properties along with those of a purple color have been acquired.

Again, thank you for your performance.

Sincerely,

The Robber Barons

If you have made any errors, fix them using the replace function by toggling the *INS* key. Let's get those icons back on the screen by pressing *Fn* and then *F1*. Remember, you could have kept the icons on the screen while typing; however, without them, more text will be displayed. At the lower right you will see an outline of the letter on a formatted (with margins, tabs, spacing, etc.) 8½ by 11-inch sheet of paper.

It is important to back up your document at this point; as we discussed earlier you could lose it through a power failure or just human error. In order for *HOMEWORD* to save the letter, press *ESC* to move the cursor to the icon area of the screen. With your joystick or cursor movement keys, place the cursor at the File icon and press Enter.

With the file menu on the screen, move the cursor to the Save Document icon and press Enter. *HOMEWORD* will now ask you for a filename. Remembering what you learned in the chapters on DOS, give your document a filename. If you can't think of one, call it *sillylet.txt* and type the same. Strike the Enter key and the prompt to assure you that your letter was saved.

Turn on your printer, assuming you have one, and we'll step through printing our *sillylet.txt*. With *HOMEWORD* still up, of course, press the ESC key to get to the Main Menu. Hitting the left arrow cursor key twice will move you to the Print icon; then press enter. As in the Save command, use either the left or right arrow to display the icon names and choose the Print document icon. Again, press Enter. You will now see the following on your screen:

```
Press ENTER to start printing.
Press the SPACEBAR to pause.
Press ENTER to start again.
Press ESC to cancel.
Pause at the end of each page.
Type Y (for Yes) or N (for No).
Current choice: DON'T PAUSE.
```

If you were feeding single sheets through your printer, you would select *Y* so that the printer would pause between pages. Carry on by striking the return or enter key; your letter will be printed. Hit Enter again to return to the typing area.

Esc will clear the screen and get you back to the main menu.

It would be appropriate here to list some of the keystrokes used in *HOMEWORD*'s editing functions.

Ctrl + A	INSERT DOCUMENT
Ctrl + B	BOLDFACE
Ctrl + C	COPY TEXT
Ctrl + D	NEW PAGE
Ctrl + E	ERASE TEXT
Ctrl + F	FIND
Ctrl + O	CENTER NEXT LINE
Ctrl + R	FIND AND REPLACE
Ctrl + Y	MOVE TEXT
Fn + F1	TOGGLES ICON DISPLAY ON/OFF
Fn + HOME	CURSOR TO TOP OF DOCUMENT
Fn + END	CURSOR TO END OF DOCUMENT

You will find many more *Ctrl* + and *Fn* key functions in the *HOMEWORD* documentation, but these should give you a taste of the program's editing capabilities. Here we'll try an editing move that will demonstrate the power of *HOMEWORD.*

With a clear typing area on your screen, type the following:

This will be a test of *HOMEWORD*'s editing capabilities. We will test a text move command, along with several others.

Typing these short paragraphs really isn't much fun, but bear with the authors, as the hour grows late and they grow weary.

Now, press the *ESC* key to go to the menu; from the menu, select the Edit icon, again using Fn and F1 keys to see the icon's names. We are going to switch those two paragraphs around, so choose the Move text icon.

After striking the Enter key, move the cursor (in the typing area) to the "This" in the first paragraph and hit the Enter key again. You will be asked to "paint" the text you wish to move by moving the cursor keys. Paint the entire first paragraph using the arrows. Incidentally, you can paint an entire line by hitting the down arrow. Press Enter again and you will be asked where you want to move the text. Move the typing area cursor to the line below the second paragraph and press Enter. Presto—your paragraphs have been moved. You can use the *DEL* key to erase extra spaces or line-ending symbols.

HOMEWORD will also find and replace letters, words, or phrases in your text through its search and replace functions. Search and replace is also accessed through the Edit icon menu and the selection of Find and Replace. You'll be prompted to move your cursor to the portion of the text from which you want to start looking for a word. Move it to the top of the document.

Now, hit the *Fn* key and HOME, then Enter. Again, you'll be prompted for the text you wish to find and you can enter up to 39 characters. Enter "the" and again you will be prompted for the replacement word. Be whimsical and enter "*PCjr*" and press Enter. Here *HOMEWORD* will allow you to make some choices as to whether you want to replace every "the" with "*PCjr.*" Throw caution to the winds and replace them all by pressing the spacebar and then pressing Enter.

Obviously, *HOMEWORD* has many more powerful features, as described earlier. The program is certainly not a full-service office word processor; however, it will be more than adequate to handle needs ranging from school work to the requirements of a small retail store. *HOMEWORD*'s documentation is clear, and its color, displayed on *PCjr*, spectacular. The use of menus and icons will allow the novice quick and easy word processing.

There are, of course, many other word processors on the market. The manuscript for this book was typed (word processed?) with *WordPerfect*™ from Satellite Software International of Orem, Utah. It comes in two versions, *WordPerfect*, a full service professional word processor that will footnote automatically, allow various keys to be defined in any manner you desire (like *Personal Communications Manager* or *PC-TALK*), do mathematical operations, and type mailing labels, to name just a few of the functions. The second version is called *Personal WordPerfect*™, a stripped down version, more akin to *HOMEWORD*. Both operate on *PCjr*. Among the other word processing packages that will work with *PCjr* are *Volkswriter*™ and *WordPlus PC*™.

It's a pretty good bet that most word processing software that requires only one disk drive and 128K will operate on *PCjr;* however, consult your dealer prior to making a purchase.

Oh, one other matter: Freeware. Yes, there is a Freeware-type word processing package available that will run on *PCjr.* It's called *PC-WRITE,* a hefty package that prints out some 100 pages of documentation. *PC-WRITE* supports many of the functions of commercial word processors, and though it's a relatively young program, appears to operate quite well. If you're cruising around the bulletin boards, you'll probably see it available for downloading (although at 300 baud, it will take quite a while; in fact, since most BBS's have time limits, you'll probably have to connect more than once to get it all). An easier route, however, would be to write to Quicksoft, 219 First N. #224, Seattle, WA 98109 (Telephone: 206-282-0452) to determine the best way to obtain the package.

14

The IBM® Personal Editor

The IBM Personal Editor (PE) is a general-purpose editor. Its use requires a bit more thought than *HOMEWORD* or other programs released specifically for *PCjr.* However, as you become more proficient with *PCjr,* you may find yourself in need of the PE—it can be customized for a specific use, such as word processing or program editing. If you have tried to do any major amount of work with the DOS line editor, EDLIN, the full-screen power of the PE should be a welcome change. We have included this chapter to give you an idea of the PE's capabilities and of what you can do with the program and a little imagination.

In addition, we have included clarifications and corrections to the Personal Editor manual; suggestions for effective use of the Personal Editor; how to recover data with [undo] the PE and the .UNNAMED file; and how to extend the power of the Personal Editor with the Define and Macro commands.

The PE has a wide range of commands and functions. (The difference between commands and functions is explained below.) The PE allows you to define the keyboard to generate entire sequences of these commands and functions by pressing just one or two keys. Because use of this feature can provide big gains in productivity, considerable attention will be given to it in this chapter. Before going any further, read at least the Tutorial section of the PE manual.

Once the PE creates the spill file, it will return to it each time it needs another piece of a file (like when you Page Down). It continues to do this even after the memory-full condition is alleviated. Since the continual reading and writing of the spill file takes so much time, we recommend that you try to avoid the memory-full situation which triggers creation of the spill file. To keep track of available memory, the PE provides the *? memory* command. When this figure gets low, it's time to think about freeing up memory by "quitting" or "filing" one of the files being edited.

Another approach to conserving memory is to break up a large file into a number of smaller files, so that only a part of the data is in memory at one time. The files can be recombined later on if necessary. One way to do this is with the DOS COPY (with concatenation) command. For example:

```
COPY PART1+PART2+PART3 ALLPARTS
```

combines three files into the output file called ALLPARTS.

Editing Multiple Files

The PE can load up to 20 files into memory—or a combination of memory and the spill file—at one time. However, only one file can be viewed and edited at a time. This file is called the *active file*. The PE provides two ways to switch active files.

To make a specific file active, use the *Edit* command with the file name. If the file is not already in memory, the PE will load it from diskette. If the file cannot be found in memory or on diskette, the PE will start a new file. To step through the files already in memory, use the *Edit* command without a file name. (F8 is set up to do this.)

The PE will remember margin and tab settings for each file. When a new file is created, the last margin and tab settings that were in effect will be used.

Data may be moved and copied between files. This can be used, for example, to copy standard text or figures from a sample document, or to break up a finished document into page-length pieces for printing. (We will explain how to do this later, after a few more concepts are introduced.)

Setting Up Work Diskettes

Before you start work with the PE, you should make at least two work diskettes. (They will back each other up, as discussed in Chapter 11.) First, format the diskettes with the DOS FORMAT command. Next, copy the file PE.HLP to the work diskettes. This will allow the PE to access the Help-screen file without making you re-insert the PE program diskette each time you use Help (F1). You may also want to copy PE.PRO and any other key-definition files you have created.

The Screen Layout

The PE divides the screen into four areas. The first 22 lines of the screen make up the data (or "text") area. The data area is a window into a file. Since the window is only 22 lines high and 80 columns wide, the file may extend beyond the window frame in any direction.

Line 23 of the screen is highlighted; it is called the *command line*. All commands are entered here. The *Esc* key moves the active cursor back and forth between the data area and command line.

Line 24, below the command line, is the *status line*. The status line shows the name of the file being edited, the cursor position in the data area (line and column), and whether the PE is in Insert or Replace mode (controlled by the Ins key).

Line 25, at the bottom of the screen, is reserved for error messages from the PE.

Memory Management and the Spill File

Whenever the size of a file (or the sum of all files) being edited gets too big to fit into the computer's memory, the PE will automatically create a diskette file named PE.TMP to hold the overflow. This is called the *spill file*. The spill file can take up to 128K of space on the diskette, if the PE needs it and if there is room available for it on the diskette. If there is no room on the diskette for the spill file (or if the spill file gets full), the PE will display the message

MEMORY IS FULL—REMOVE FILES.

Once this message appears, the PE will not allow you to do any more editing functions until a file is removed from memory (by either the Quit or File commands).

Internal Files

The PE makes use of three *internal files*. This term refers to a file which cannot be saved or filed on diskette. However, internal files may be renamed (with the *Name* command), so that they can be used like any other file. Additionally, contents of these files may be copied into "real" files. The internal files are: .KEYDEFS, .UNNAMED, and .DIR. The .DIR file holds the results of the last *Dir* command. The other two internal files will be covered in separate sections.

Commands and Functions

Briefly stated, the difference between a *command* and a *function* is that a command can be entered only on the command line, while most functions can only be executed in the data area. Since the PE assumes that anything you type in the data area is data, functions must always be assigned to special keys. The keys which may be used are:

- Any cursor-movement key.
- The *Ctrl* key in combination with any cursor-movement key.
- The function keys *F1* through *F10*.
- The *Ctrl* key in combination with a function key.
- The shift (upper-case) key in combination with a function key or the tab key.
- The *Alt* key in combination with an alphabetic, numeric, or function key.

Defining Keys

A function is assigned to a key with the *Define* command. A complete list of the available keys and the "default" definitions given to them is provided in Appendix B of the PE manual. As you can see, one key can generate a mixture of any number of commands and functions (as long as the definition—including functions, commands, and text—does not exceed 254 characters, or contain more than 120 characters of text).

Each time the PE is started, it reads a diskette file called PE.PRO which contains a series of *Define* commands. (It does this after the title screen asks you to "Press Enter to continue.") The PE stores these key definitions in an internal file called .KEYDEFS. To display the key definitions being used by the PE, edit .KEYDEFS. The *Edit* command may be used to look at the contents of .KEYDEFS, but changing a key definition in the .KEYDEFS data area will not have any effect. Instead, functions are assigned to a key by the Define command. If this command is used and .KEYDEFS edited again, the new definition will appear. However, the new assignment is just temporary. The next time the PE is run, the original definitions will be read in again from PE.PRO.

Key definitions may be changed permanently either by editing the PE.PRO file, or by changing the name of .KEYDEFS and editing it. If the new key definitions are filed under the name PE.PRO, the PE will automatically load them into .KEYDEFS when it starts up. If the key definitions are given some other name, the PE will have to be told to use that file instead of PE.PRO. This is done with the Macro command.

(The PE will still use PE.PRO when it starts up. The Macro command may be issued at any time to have the PE load and use a new set of definitions or execute any set of commands.)

Making Your Mark

A lot of editing functions call for areas to be "marked" first. The PE provides three types of marking functions.

The [mark line] function (normally Alt-L) tags the entire line where the cursor is by highlighting the line (using reverse video). If a second line is marked, the tagged area extends from the first marked line to the second. The PE indicates this by highlighting the entire area. Entire portions of files can be moved around intact with the [mark line] function.

The [mark block] (Alt-B) function permits you to define a rectangle of any size and shape. The rectangle is set up by marking any two diagonally-opposite corners. That is, use [mark block] in the upper left and lower right corners, or in the upper right and lower left. The [mark block] function provides a cut-and-paste feature. It can be used, for example, to move a column of figures from one side of a chart to the other, or to move the entire chart around on a page.

The [mark character] (Alt-C) function allows you to tag a single character or group of characters of any length. When you mark the two ends of the characters to be tagged, you will see a highlighted area which follows the contours of the data, always starting in column one, but extending to the right only as far as the data (including typed spaces) does. Blank lines will not be highlighted, but are included in any marked-character operation. The [mark character] function is useful for editing of all types.

The PE allows only one area to be marked at a time. The [delete mark], [move mark], and [unmark] functions all remove existing marks. All the other mark functions—[begin mark], [copy mark], [end mark], [fill mark], [overlay block], and [reflow]—leave the tag in place.

Using [UNDO] and the .UNNAMED File

There are two ways to restore deleted or changed text: by using the [undo] function, or by editing the .UNNAMED file. Each method has its limitations.

The [undo] function (normally assigned to Shift-F4) can be used only before the cursor is moved off the line you wish to recover. Furthermore, [undo] will not restore data deleted by the [delete line] function (Ctrl-Backspace), or if the change resulted from a series of functions generated by a single key.

The .UNNAMED file is a sort of "holding area" for changes. When you make a change, the PE saves the original data in this internal file. The .UNNAMED file will hold only five changes. When you make a sixth change, the PE deletes the oldest of the saved original texts from the .UNNAMED file. Note that the size of each change is not limited. For example, the entire block transferred by the block-move function is stored in .UNNAMED by the PE.

To recover something from the .UNNAMED file, do the following:

1. Move the cursor to the command line (if necessary) by pressing the Esc key.
2. Type the command "E .UNNAMED" to Edit the .UNNAMED file.
3. Mark the text you wish to recover. You can use line, block, or character marking.
4. Now make the file you started from active again by typing an Edit command with the proper file name. (You can also use Edit with no file name (F8) to step through all the files in memory.) DO NOT USE THE QUIT COMMAND to leave .UNNAMED, because that will wipe out the entire .UNNAMED file.
5. When you have switched back to your original file, position the cursor as desired, and use Alt-M to move the original data from the .UNNAMED file.

You can also use Alt-Z to copy from .UNNAMED. This leaves the original data in .UNNAMED, still marked.

Move and Copy Between Files

The technique described above for retrieving data from the .UNNAMED file may be used to move and copy data between any files being edited with the PE. (Before doing this, you might want to Save both the source and destination files, in case of error.) For example, if you wish to transfer some data to another file:

1. Mark the data to be moved, copied, or used as an overlay.
2. Switch to the file which will get the data: use either the *Edit* command or *F8*. DO NOT USE THE QUIT COMMAND to leave the current file, as that will remove the file from memory and cancel the marked area.
3. Position the cursor in the destination file and use the appropriate command to transfer the source data.

It is a good idea to Save again, after the successful transfer.

Advanced Formatting Techniques

This brief discussion of a few formatting techniques may help you to be more inventive in fitting your own needs.

A "paragraph" of text (any block of data beginning and ending with a blank line) may be formatted within set margins by the sequence of functions normally generated by Alt-P. ("Normally" means the Alt-P functions as defined in PE.PRO.) Each paragraph can be formatted with different margin settings, as the text below illustrates:

1. The text was originally entered using the default margin settings.

2. After all text was entered, the margins were set to the narrow width desired for the columns you see here.

3. The *Alt-P* functions were used on each block of text to reformat it within the new margin settings.

4. Points 3 and 4 were then block-marked, and the [move mark] function used to transfer the text, producing the double columns.

After all that, the margins were reset to their original values and the empty lines produced by moving the text block were deleted.

The Notabs Option

The PE tries to save space on a diskette when it writes a file by substituting a series of Tab characters for the equivalent number of blanks which were entered with the space bar. Unfortunately, since only the PE knows what the Tab settings were when the file was last edited, only the PE can recreate the right number of spaces from the Tab characters. If any other program will be using the output of the PE—or if you wish to display or print your file directly with DOS commands—then the PE must be told to write the file in "uncompacted" form. The *Save* and *File* commands have the notabs option to do this.

In the customized key definitions below, a specific key is set up to generate the Save command with the notabs option.

Manual Labor

The First Edition (November, 1982) of the PE manual contains a couple of errors. The third example of the CHANGE command (pages 5–8) will not capitalize "xyz," but does put the space on either side. The last example of the LOCATE command (pages 5–19) is incorrect as written. If you wish to search backwards for "xyz," the command is: "/xyz/-." As written, the command will search forward for "-xyz—."

Creating Your Own Editor

Now it's time to get creative, and personalize the Personal Editor. The following sections will present two different examples of customized key definitions. They are included here to give you an idea of what can be done; you can build your own Editor as your needs dictate.

The Memo Writer

The *memo writer* key definitions are used to generate a memo form. The process includes pauses so that the user can supply heading data.

Because of the limitation that a key definition can contain no more than 120 characters of text, separate keys must be used to generate each different line of the form. To create a form, start with an empty file. Then, press the *Alt* key and the function keys in numerical order (1 to 9). When *F4* is pressed, and with each key thereafter, the cursor will appear at a location in the heading where the user is expected to add information. A vertical line is left in the SUBJECT field, but this should be over-typed when the subject is filled in. *Alt-F10* has been defined to generate a *Save* command with the notabs option. Since a file name must be supplied when the notabs option is used, the definition of *Alt-F10* places the cursor where the user must insert the file name. If only one file is being edited, then the key can be temporarily defined to include the file name in the *Save* command, and to execute the command. On the command line type:

```
def a-f10 = [cursor command] [begin line] [erase end line]
    'save filename notabs' [execute]
```

The memo form and the key definitions which produced it are shown below. The file which contains these definitions was created by editing PE.PRO. Care was taken to define the *Alt-function* keys in such a way that the correct form will be produced even if the keys are pressed in the wrong sequence. The new definitions were filed under the name MEMOKEYS.

Other key definitions have been added to make the PE more suitable for general word processing:

The *Enter* key has been redefined to provide a new line. This gives it a function identical to that of *F9*, and makes it more natural for text entry (that is, more like a typewriter carriage-return key).

(The *Enter* key is the only key definition which has been changed: All the rest make use of keys which are not defined by default. This approach avoids confusion when switching from one set of key definitions to another. Keeping most keys the same in all definitions makes it easier to remember them. Also, the PE manual description for those key definitions will still be valid, leaving fewer keys which must be separately documented.)

Alt-A has been defined to produce a standard "page footing," but its use is a bit tricky. When *Alt-A* is pressed, it generates three blank lines followed by a line with the word *Page* centered in it, and a printer control code to skip to a new page. Then the cursor is positioned so that the user can insert a page number. This standard set of functions is meant to insure that the line will be the last on the page, and that the next line will be at the top of the next page.

However, it is up to the user to figure out when the end of the page has arrived. (That is, when *Alt-A* should be used.) The standard page holds a maximum of 66 lines, so *Alt-A* should be employed no later than line 60. The status line may be used to keep track of the current line within the current page. (A full-function word processor can keep track of when page-breaks are required, and can automatically generate a footer with the correct page number. The PE is not a full-function word processor.)

To make printer control easier, the combination of *Ctrl* and the function keys has been defined to generate some printer control codes. For example, pressing *Ctrl-F1* generates two characters which tell the IBM Graphics printer (or any Epson printer) to turn on double-strike (bold face) mode.

Finally, we created a [delete word] function by defining a series of functions for the *Alt-W* key.

To implement the commands and definitions in MEMOKEYS, the command "MACRO MEMOKEYS" must be issued, since the PE will use PE.PRO when it is first started. (See "Defining Keys," above.)

(If you have an IBM graphics printer, the appearance of the memo form can be improved by using the extended character set to make the box: see Appendix G of the BASIC manual for the characters.)

Memo Writer Form and Key Definitions

```
+-----------------------------+-----------------------+
!                             !                       !
! TO:                         ! FROM:                 !
!                             !                       !
! LOCATION:                   ! LOCATION:             !
!                             !                       !
!                             ! DATE:                 !
!                             !                       !
+-----------------------------+-----------------------+
!  SUBJECT:                   !                       !
+-----------------------------------------------------+
```

In the definitions below, the graphics lines have been shortened to fit within the margins of this page.

SET MARGINS 5 75
SET TABS 5 10 15 20 25 30 35 40 45 50 55 60 65 70

Make sure that the cursor starts in the right location:

```
def  a-f1  =  [cursor  data]  [top]  [begin line] [erase end
line] [replace mode]
'              +-----------------------------+---------------------+'
[insert line]
```

```
def a-f2 =
'      ¦                          ¦                      ¦'
[mark  line] [copy mark] [copy mark] [copy mark] [copy mark]
[copy mark] [copy mark] [bottom] [insert line]
```

Add the line which separates the routing information from the Subject, and copy another marked line.

```
def a-f3 =
'              +-----------------------------+---------------------+'
[copy mark] [bottom] [insert line]
```

The empty form will be finished and the cursor positioned for the first user-supplied data:

```
def a-f4 =

     +---------------------------------------------------+'

[insert  line] [unmark] [top] [begin  line] [down] [down]

[first nonblank] [right] [right] 'TO:
```

Since the user may move the cursor after typing, the key definitions from here on will begin with functions which ensure correct cursor position. The added logic also means that the *Alt-function* keys can be pressed out of their "proper" (numerical) sequence and still generate the correct data in the correct place. This involves some extra work when defining the keys, but it is well worth the effort; the results make the PE more powerful and more forgiving of human error.

Note the sequence of functions which are used to place the Locate ("/") command in the command line and then execute it:

```
def  a-f5  =  [cursor data] [top] [begin line] [down] [down]

[first  nonblank]  [cursor  command] [begin line] [erase end

line]  '/|' [execute] [cursor data] [right] [right] 'FROM:
```

```
def  a-f6  =  [cursor  data]  [top]  [cursor command] [begin

line] [erase end line] '/TO:' [execute] [cursor data] [down]

[down] 'LOCATION: '
```

```
def  a-f7  =  [cursor  data]  [top]  [cursor command] [begin
line]  [erase  end  line]  '/FROM:'  [execute] [cursor data]
[down] [down] 'LOCATION: '
```

```
def  a-f8  =  [cursor  data]  [top]  [cursor command] [begin
line]  [erase  end  line]  '/FROM:'  [execute] [cursor data]
[down] [down] [down] [down] 'DATE: '
```

```
def  a-f9  =  [cursor  data]  [top]  [cursor command] [begin
line]  [erase  end  line]  '/LOCAT'  [execute] [cursor data]
[down] [down] [down] [down] [down] 'SUBJECT: '
```

This completes the key definitions used to create the memo form.

The next key generates a *Save* command with the notabs option, then places the cursor so that the user can insert a file name:

```
def  a-f10  = [cursor command] [begin line] [erase end line]
'save  notabs' [begin line] [right] [right] [right] [right]
[right] [insert mode]
```

The following definitions generate printer control codes for the Epson and IBM graphics printers. The characters in the quote marks are produced by holding down the *Alt* key while entering the three-digit code on the numeric key pad. For each three-digit code a single character should appear when the *Alt* key is released.

```
def c-f1 = 'Alt-155 Alt-071'  Bold (double strike) on

def c-f2 = 'Alt-155 Alt-072'  Bold off

def c-f3 = 'Alt-155 Alt-052'  Italics on

def c-f4 = 'Alt-155 Alt-053'  Italics off

def c-f5 = 'Alt-143'          Compressed print on

def c-f6 = 'Alt-146'          Compressed print off

def c-f7 =

def c-f8 =

def c-f9 = 'Alt-142'          Double width on current line

def c-f10 = 'Alt-140'         Page  eject (skip to new page)
```

Finally, we come to the "standard footer" and "delete word" definitions, and the revised definition of the Enter key. (The *Alt-140* combination produces a single character; hence, the three [left] functions in the definition of *Alt-A* position the cursor after "Page ".)

```
def  a-a = [insert line] [insert line] [insert line] [insert

line]  [right40]  [left8]  'Page      Alt-140' [left] [left]

[left] [insert mode]
```

```
def  a-w  =  [unmark]  [mark  char]  [tab word] [left] [mark

char] [begin mark] [delete mark]
```

```
def enter = [insert line]
```

The Basic Editor

This example of a customized editor takes a different approach from that of the Memo Writer. Instead of trying to add functions to the standard PE, these key definitions are intended to mimic the BASIC editor as much as possible. After the BASIC keys are defined, additional PE editing functions are assigned to the remaining keys.

Since the built-in editor of BASIC makes use of the *Alt-letter* keys to generate certain BASIC words, all the usual PE "mark" functions must be reassigned. This could make things quite confusing to someone who is familiar with the default definitions used by the PE. (Of course, the intent is to make things less confusing for the BASIC programmer.) To help matters a bit, the standard PE "help" function *(F1)* is redefined to edit our own help file, BASKEYS.HLP. (Remember that function keys on the *PCjr* are activated by pressing the FN key first, then the appropriate number key.)

The contents of BASKEYS.HLP are listed below. The file was produced by renaming PE.HLP to BASKEYS.HLP, editing it as shown, and saving the new key definition file to diskette.

The command MACRO BASKEYS must be entered to use these definitions.

Help Screens for Basic Editor

```
+----------------------------------------+
!       B A S I C   H e l p       p.1    !
+----------------------------------------+
!                                        !
!  Function Keys                         !
!                                        !
!  F1   --   Help Menu                   !
!  F2   --                               !
!  F3   --   Edit                        !
!  F4   --   Save                        !
!  F5   --                               !
!  F6   --   Print                       !
!  F7   --                               !
!  F8   --                               !
!  F9   --                               !
!  F10  --   Quit                        !
!                                        !
!            s => Shift key              !
+----------------------------------------+
!       s-F1 = dn      F10 = Exit        !
+----------------------------------------+
+----------------------------------------+
|      B A S I C   H e l p      p.2      |
+----------------------------------------+
|                                        |
|  Alt key and a letter...               |
|      See the BASIC manual.             |
|                                        |
|                                        |
| F4 (SAVE command) saves file to        |
|      diskette.                         |
|                                        |
| F10 (QUIT command) removes text        |
|      from memory without saving.       |
|                                        |
|  If the QUIT message occurs,           |
|  Typing N stops QUIT command           |
|  typing Y removes any change           |
|  since the last SAVE or FILE.          |
|                                        |
+----------------------------------------+
| s-F1 = up   s-F2 = dn   F10 = Exit     |
+----------------------------------------+
```

```
+----------------------------------------+
|      B A S I C   H e l p      p.3      |
+----------------------------------------+
|                                        |
|  Eight Cursor Movement Keys            |
|                                        |
|      Home        Cursor Up    PgUp     |
|                                        |
|      Cursor Left  Cursor Right         |
|                                        |
|      End Cursor Down   PgDn            |
|                                        |
|                                        |
| Cursor Up      up one line             |
| Cursor Down    down one line           |
|                                        |
| Cursor Left    left one space          |
| Cursor Right   right one space         |
|                                        |
+----------------------------------------+
| s-F1 = up  s-F2 = dn   F10 = Exit      |
+----------------------------------------+
+--------------------------------------+
|      B A S I C   H e l p     p.4    |
+--------------------------------------+
|                                      |
|  Cursor Movements  (cont.)           |
|                                      |
|                                      |
| Home      to top of screen           |
| End       after last character       |
| PgUp      shows previous page        |
| PgDn      shows next page            |
|                                      |
| Ctrl-Home   to Top of file           |
| Ctrl-End    erase to end of line     |
| Ctrl-PgUp   to top of screen         |
| Ctrl-PgDn   to bottom of screen      |
| Ctrl-Left   to beginning of word     |
| Ctrl-Right  to end of word           |
|                                      |
+--------------------------------------+
| s-F1 = up  s-F2 = dn   F10 = Exit   |
+--------------------------------------+
```

```
+--------------------------------------+
|      B A S I C   H e l p     p.5    |
+--------------------------------------+
|  Text Markers    a => Alt key        |
|                                      |
|  a-F1 -- block mark for              |
|          rectangles, vertical        |
|          and horizontal lines        |
|  a-F2 -- character mark for          |
|          sentences, phrases          |
|          and characters              |
|  a-F3 -- line mark for one           |
|          line or paragraph           |
|  copy mark...       a-F5             |
|  delete mark...     a-F6             |
|  move mark...       a-F7             |
|  overlay block...   a-F8             |
|  unmark...          a-F9             |
+--------------------------------------+
|       s-F2 = up     F10 = Exit       |
+--------------------------------------+
```

The BASKEYS Definitions

```
set margins 1 254 1
set tabs 1 9 17 25 33 41 49 57 65 73 81 89 97 105 113 121 129 137 145
  153
def up = [up]
def down = [down]
def left = =[left]
def right = [right]
def pgup = [page up]
def pgdn = [page down]
def home = [top edge]
def end = [end line]
def ins = [insert toggle]
def del = [delete char]
def enter = [insert line]
def backspace = [rubout]
def esc = [command toggle]
def tab = [tab]
def f1 = [cursor command] [begin line] [erase end line] 'e bas-
  keys.hlp' [execute]
def f2 =
def f3 = [cursor command] [begin line] [erase end line] 'edit'
def f4 = [cursor command] [begin line] [erase end line] 'save'
def f5 =
def f6 = [cursor command] [begin line] [erase end line] 'print'
def f7 =
def f8 =
def f9 =
def f10 = [cursor command] [begin line] [erase end line] 'set
  display color 80'
def c-left = [backtab word]
def c-right = [tab word]
def c-pgup = [top edge]
def c-pgdn = [bottom edge]
def c-home = [top]
def c-end = [erase end line]
def c-enter = [execute]
def c-backspace = [delete line]
def c-f1 =
def c-f2 =
def c-f3 =
def c-f4 =
def c-f5 =
def c-f6 =
def c-f7 =
def c-f8 =
def c-f9 =
def c-f10 =
def s-tab = [backtab]
def s-f1 = [page down] [bottom edge] [down] [down] [cursor
  command]
def s-f2 = plpage up] [top edge] [up] [up] [cursor command]
def s-f3 = [reflow]
def s-f4 = [undo]
def s-f5 = [confirm change]
def s-f6 =
def s-f7 = [shift left]
def s-f8 = [shift right]
def s-f9 = [cursor command] [begin line] [erase end line] ‘dir a:’
  [execute]
def s-f10 = [cursor command] [begin line] [erase end line] ‘dir b:’
  [execute]
def a-a = ‘AUTO’
def a-b = ‘BSAVE’
def a-c = ‘COLOR’
def a-d = 'DELETE'
def a-e = 'ELSE'
def a-f = 'FOR'
def a-g = 'GOTO'
def a-h = 'HEX$'
def a-i = 'INPUT'
def a-j =
def a-k = 'KEY'
def a-l = 'LOCATE'
def a-m = 'MOTOR'
def a-n = 'NEXT'
def a-o = 'OPEN'
def a-p = 'PRINT'
def a-q =
def a-r = 'RUN'
def a-s = 'SCREEN'
def a-t = 'THEN'
def a-u = 'USING'
def a-v = 'VAL'
def a-w = 'WIDTH'
def a-x =
def a-y =
def a-z =
def a-f1 = [mark block]
def a-f2 = [mark char]
def a-f3 = [mark line]
def a-f4 = [fill mark]
def a-f5 = [copy mark]
def a-f6 = [delete mark]
def a-f7 = [move mark]
def a-f8 = [overlay block]
def a-f9 = [unmark]
def a-f10 =
def a-0 =
def a-1 =
def a-2 =
def a-3 =
def a-4 =
def a-5 =
def a-6 =
def a-7 =
def a-8 =
def a-9 =
```

15

Home Budget, Jr

HOME BUDGET, Jr is just what the name implies: A program that will allow you to keep track of the way you handle your finances. If you are accustomed to a budget, you will adapt easily to this program. There will be changes, of course, in the way you keep track of your budget items, because you'll be putting them on the computer. But overall, your computerization of home budgeting should come fairly easily. If you have never budgeted, but have had the desire to do so, this package will help you along—and give you some incentive to continue keeping your accounts straight. That incentive is the money you spent on the program.

Keeping track of every expense may be somewhat counterproductive, so we suggest that you first develop a budgeting plan. In other words, determine what your budget goals are: more vacation money, more spending or savings funds, or simply having a reasonable idea of where it's all going.

Budgeting—at least on paper—isn't for everyone because it does involve tracking expenses and income. You will have to write down what you spend and where, then transfer it to the computer. However, once you develop some simple disciplines, the time spent doing this will be minimal and the time spent entering the data on *PCjr* shouldn't be more than an hour a month.

The requirements for *HOME BUDGET, Jr* on the *PCjr* include one disk drive, the BASIC cartridge and one blank diskette. You'll also need DOS and, of course the *HOME BUDGET, Jr* program diskette.

(By the way, *HOME BUDGET, Jr* also operates on the IBM PC and the PC XT; however, on these machines the BASIC cartridge is not required.)

To get started, insert the BASIC cartridge in the appropriate slot and DOS in the disk drive, then switch on the computer. After entering the time and date, as prompted on the screen, remove DOS and insert your *HOME BUDGET, Jr* diskette. Then type the letter *G* and hit return. If you are using *HOME BUDGET, Jr* for the first time, the program will prompt for the kind of screen you are using—color or monochrome. Once this is out of the way, you are just about ready to begin.

HOME BUDGET, Jr will ask you for the date, even though you entered when you booted DOS. You must enter the date just as requested: MM/DD. That means two digits in each place. For January 9, the input would be 01/09. Whatever date you enter will be the date *HOME BUDGET, Jr* remembers as the creation date for the budget.

Before you actually start entering data into the program, make a list of expense categories. *HOME BUDGET, Jr* will allow you up to 48 different "accounts" for expenses. Your list might include the following:

Rent/Mortgage
Utilities
Telephone
Water
Car
Maintenance
Gasoline
Recreation
Gifts
Clothing
Food
Installment Loans

Of course, these are just suggestions and memory joggers. You can, as mentioned before, have as many as 48 different expense categories. However, the more inclusive each category, the easier it will be to keep your budget and mind in order. At the outset, very narrow categories may be too detailed for your needs.

Now, assuming you have configured your *HOME BUDGET, Jr* diskette to the proper screen and it is up on *PCjr*, you'll have a menu with six selections. Choose number five: *Create or Change Budget.* If, by the way, you want to set everything aside for a while and come back to the creation of the budget, press *ESC* to end the program. Striking this key while you're using one of the options on the menu will return you to the Main Menu.

Once you select the *Create* option, you'll be tossed into a submenu, from which you will select number four: *Create a New Budget. HOME BUDGET, Jr* will prompt you through this. You will now see the following message:

```
ENTER THE ACCOUNT NAME [        ]
```

The space within the brackets shows just how long an account name may be. It is important to note here that *HOME BUDGET, Jr* will *not* accept spaces in account names. This is easily overcome by utilizing a (−) or just by not spacing. For example, if you have two car payments you might want to designate them:

```
CAR1
CAR2
```

Because of *HOME BUDGET's* space constraints on account names, eight letters, you will probably find yourself abbreviating from time to time. You don't have to worry about an INCOME category because *HOME BUDGET, Jr* creates that one for you automatically.

After you enter an account name, you will be prompted for a monthly amount. *HOME BUDGET, Jr* will ask:

```
ENTER THE MONTHLY AMOUNT [        ]
```

The amount *must* be entered in whole numbers and without dollar ($) signs or decimals. In other words, if you are allotting $500.00 per month for rent, enter 500.

Charge accounts will be handled a bit differently. The space constraint remains eight letters, but three are required as an extension. Using Sears as an example, you would enter:

```
SEARS-CA
```

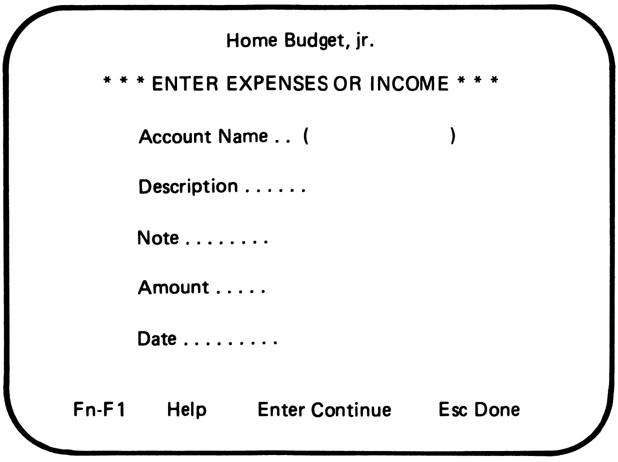
The -CA tells *HOME BUDGET* that this is a charge account; with such an account, there is no amount entered as the program sets the amount to zero. You will see this on the screen in the form of the following message:

```
CHARGE ACCOUNT . . . AMOUNT SET TO ZERO
```

If you happen to enter a duplicate account name, *HOME BUDGET, Jr* will alert you to this error and ask you to enter another name. Once you have entered all of your accounts and budget allotments, strike the *ESC* key to return to the Main Menu. At this point it would be a good idea to review your entries, so from this menu, select option 5 to *Display Budget.*

If you are satisfied, we can continue. If you need to make changes, *HOME BUDGET, Jr* will allow for the addition of new accounts or changes in account amounts or names. However, if you want to delete an account you must redo the entire budget. You can get around this problem by changing the *amount* to 0 and adding a new account (using the appropriate selections on the Main Menu), spelled properly. Again, if you just want to change an account name, correct a typo, or change the amount, select the appropriate Main Menu selection and follow the prompts.

Let's start entering some income and expense information at this juncture. If you are not at the Main Menu, get there by striking the *ESC* key until you see it, then select the first option, *Enter Expenses or Income.* The following illustrates your screen display.



```
                 Home Budget, jr.
        * * * ENTER EXPENSES OR INCOME * * *

          Account Name . . (              )

          Description . . . . . .

          Note . . . . . . . .

          Amount . . . . .

          Date . . . . . . . . .

Fn-F1    Help    Enter Continue    Esc Done
```

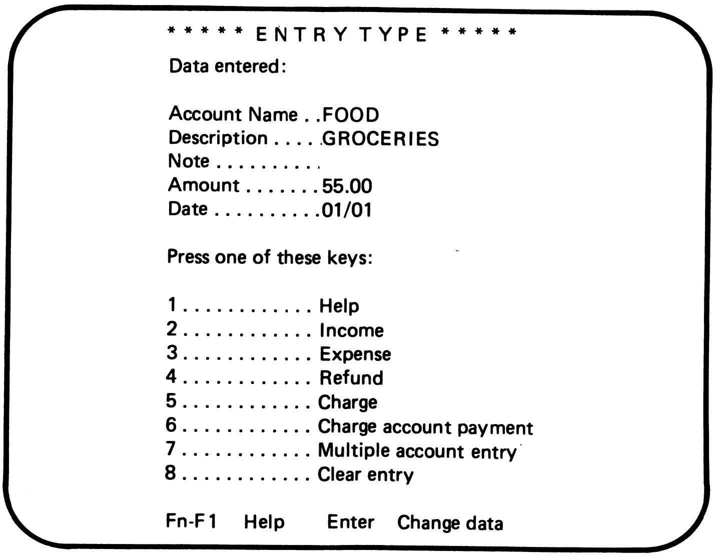
It is through this screen that you will enter your transactions, or entries. The only things other than account names and amounts that might be confusing here are the *Description* and *Note* entry areas. The *Description* field is for an optional explanation of up to 20 characters of the transaction, while the *Note* field is an identification area. In the *Note* field you might want to create your own code of up to three characters, to indicate tax deductible items. You might use *TX* or *TAX*, for example. *HOME BUDGET, Jr* lets you keep track of such groupings.

Once you have completed the entries on the screen, strike the *Enter* key and *HOME BUDGET, Jr* will move you to a second entry screen.

Here, select the appropriate number and move on to your next entry.

A number of entries for selection appear on the second entry screen. With the exception of the *Multiple Entry* option, they are all self-explanatory. A *Multiple Entry* selection would reflect, for example, a credit card purchase. Using this option would allow you to charge a clothing purchase at Sears to both the clothing account in your budget and to your Sears charge card budget account.

Upon completion of your expense or income entries, strike the *ESC* key and *HOME BUDGET, Jr* will move to a checking mode called DATA CHECK, through which you may check all of your entries. Again, the program self-prompts you through this procedure.



```
***** ENTRY TYPE *****
Data entered:

Account Name . .FOOD
Description . . . . .GROCERIES
Note . . . . . . . . . .
Amount . . . . . . .55.00
Date . . . . . . . . . .01/01

Press one of these keys:

1 . . . . . . . . . . . . Help
2 . . . . . . . . . . . . Income
3 . . . . . . . . . . . . Expense
4 . . . . . . . . . . . . Refund
5 . . . . . . . . . . . . Charge
6 . . . . . . . . . . . . Charge account payment
7 . . . . . . . . . . . . Multiple account entry
8 . . . . . . . . . . . . Clear entry

Fn-F1   Help     Enter   Change data
```

HOME BUDGET has a variety of other features to help you in your quest for the proverbial balanced budget. For example, it has "warning levels" for dollar entries. If the "warning level" is set at $100.00, any entry over that amount will yield the prompt DOUBLE CHECK YOUR ENTRY. You may set your "warning level" at any plateau up to $99,999.99. This function is accessed by selecting number 6, *Change Program Controls*, from the Main Menu.

The package also allows you to review any or all accounts and their activity and to print a variety of reports on the screen or in printed form if you have a printer. If you think you may be over budget on Food, for example, the status of that account may be checked on a total spent, year-to-date, and budget balance basis. If you're profligate in your spending, *HOME BUDGET, Jr* issues a warning by giving you an *OVERSPENT* message.

16

*Spreadsheets and Multiplan*T.M.

This chapter will introduce spreadsheet concepts and describe *Multiplan*™, a versatile electronic spreadsheet that will operate on *PCjr.* We will provide a tutorial and hints on how to use the program more effectively and, finally, a sample spreadsheet for a checkbook register.

A spreadsheet is laid out as a grid of rows (across) and columns (up and down). Each intersection of a row and a column is an entry position called a cell. A cell can hold one of three things:

1. A *label,* consisting of alphanumeric data.
2. A *value,* containing arithmetic data only.
3. A *formula.*

It is the ability of a spreadsheet cell to contain formulas that gives spreadsheets much of their power—and that accounts for their popularity. A formula can refer to the contents of another cell (or cells), contain constants, and make use of a number of built-in functions provided by the spreadsheet program. (For example, all spreadsheet programs offer the AVERAGE function. When the user enters the function name in a cell, and supplies a group (or "range") of cells-addresses, the spreadsheet program calculates the average of the values in those cells and makes the result the value of the cell where the function was used.)

Remember that you will not save time when first setting up a spreadsheet. You will reap the rewards of automation only when you have to change a spreadsheet or produce another similar one.

This is because the setting up of a spreadsheet is fundamentally the same whether done with pencil and paper or keyboard and screen. It is only after the initial work is done that the major benefits of automating the spreadsheet are realized. All formulas in a spreadsheet can be recalculated by the computer whenever the user desires. For anyone who has spent hours manually recalculating an entire spreadsheet just because one figure was changed, a spreadsheet program alone may justify the cost of a complete personal computer system.

Microsoft Corporation's Multiplan spreadsheet is 255 rows by 63 columns in dimension with a complete array of mathematical functions capable of calculating formulas ranging from rudimentary to complex. It comprises various capabilities for forecasting, formatting, data management, and extreme friendliness toward the user. Released in 1982, six years after the introduction of the progenitor of electronic spreadsheets, VisiCalc™, Multiplan represents the "second generation" of electronic spreadsheets.

We feel that Multiplan, among a host of like products, is particularly well-suited to the *PCjr* user for a variety of reasons:

It will run in as little as 64K of memory.

It has a complete on-line HELP facility; once read, the manual may no longer be required.

It displays a complete menu of available options on the screen at all times (as opposed to presenting a "prompt" consisting of the initial letters of each command).

It "suggests" what it believes to be the most logical choice when a selection must be made from a list of options.

It has been adapted by IBM and is available as an "IBM logo" product from any IBM product center or authorized IBM software dealer.

In brief, Multiplan seems ideally suited to the occasional user—someone who will not "live with" the program, but who will appreciate the assistance and guidance it offers.

But wait, there's more! Multiplan's power is actually far greater than its 63 spreadsheet columns and 255 spreadsheet rows, for it has the capability of "linking" several spreadsheets and relating them to one another, while automatically calculating and recalculating entries. With this facility, and others, Multiplan differs significantly in capability and ease from the earlier generation of electronic spreadsheets.

For example, Multiplan allows the user to split spreadsheets into as many as eight separate windows, each (utilizing the IBM Version) having different colors for the borders, foreground, and background with a choice of 15 colors.

Some earlier electronic spreadsheets required manual calculations for partitioning a sheet to the printed page. Multiplan incorporates user-defined top and bottom margins and length and width. The program's facility for formatting is considerably more advanced than that of its predecessors, allowing for five options to define alignment within a cell and 10 for definition of the displayed format, such as scientific notation and fixed point.

Moreover, Multiplan interacts with the user in English, requesting commands by specific, understandable words such as *Sort*, *Format*, *Insert*, and *Move*. With those commands, Multiplan even gives proposed responses that the user will find correct approximately 75% of the time.

Multiplan has the capability of interfacing with a variety of Data Base Management and Word Processing programs. This facility allows the user to move a spreadsheet into such a program for report or correspondence purposes. For VisiCalc™ users who have converted to Multiplan, Microsoft provides the SLYK format, which allows the user to move VisiCalc files into a Multiplan format.

Describing the complete range of the Multiplan program's capabilities would require more space than we have here, but perhaps the examples presented in the following pages will help the user to gain additional insight.

There are two versions of Multiplan. One is from Microsoft®, the folks who wrote the program. It's called the MS-DOS™ version. The other version is marketed by IBM.

The versions are different in a few relatively minor ways:

1. They come in different looking packages. (This means that you can tell them apart.)

2. The function keys are defined differently. All the functions are there, they are just assigned to different keys.
3. The IBM version has a Window-Paint option with which you can change the color of background, text, or border.
4. The IBM version comes on a protected disk, which means that you cannot make more than one copy. Well, sort of . . . keep reading.

Before you can run Multiplan, you must go through the installation procedure. The procedure differs for Microsoft and IBM versions. Please read your instruction manual to find out how to go about making a Multiplan diskette. If you have the IBM version, this will mean running MPCOPY, which can only be run once.

The problem with having only one copy of Multiplan is a little complicated, but here goes. You need the Multiplan program diskette in the diskette drive while you are running the program. This is because Multiplan will use it from time to time, to access parts of the program that will not fit into memory all at once (a problem which does not occur in computers with more than 128K of memory).

At the same time, the program diskette does not have enough room to hold many spreadsheets, so you would like to have a blank diskette in the drive (a problem which does not arise with a system that has more than one drive). There is a way around this problem, as follows.

After you have run MPCOPY, format a number of diskettes without putting DOS on them. Copy MP.SYS from the Multiplan program diskette to the empty diskettes. If you want to use the on-line *help* function, copy MP.HLP too. You must start Multiplan (with the MP80 or MP40 command) from its program diskette because of the protection scheme that IBM uses. However, once Multiplan is up and running, you can switch to one of the copies. Now Multiplan can get to all of the pieces of itself it needs, and you have plenty of room to store data.

Remember: always start Multiplan from its program diskette (which should have a write-protect tab on it), then switch to an unprotected copy for storing and retrieving the spreadsheets.

Finally, ***save your work often.***

17

Multiplan$^{T.M.}$ Tutorial

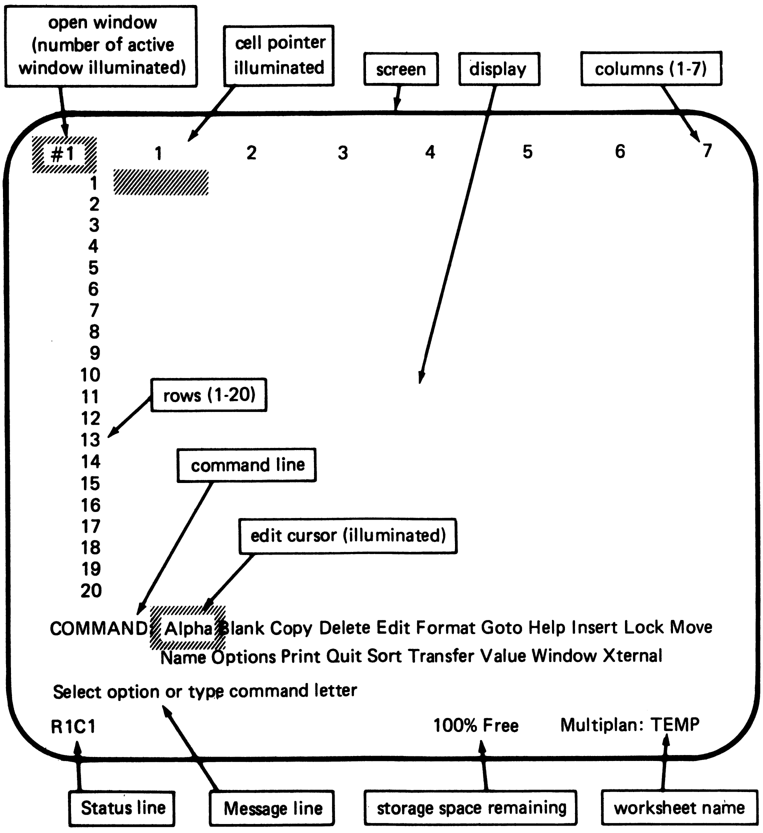
Let's assume that we have an "installed" copy of the Multiplan program. Multiplan may be loaded into memory from DOS with the command *MP.* After the program loads, it will display an empty spreadsheet.

With the blank spreadsheet on the screen, notice the highlighted Cell in the upper right hand corner, at Position R1C1. That highlighted area is known in Multiplan language as the Cell Pointer, but for purposes here will be referred to as the Cursor.

The Position reference R1C1 refers to Row 1 Column 1, with *Row* referring to Multiplan's 255 horizontal lines and *Column* to its 63 vertical lines. For example, R26C20 refers to Row 26 Column 20. When Multiplan is first loaded, the screen will display Columns 1–7 and Rows 1–20.

The spreadsheet is supposed to be 63 columns by 255 rows. So where are all the other cells? The screen can be thought of as a "window" into the spreadsheet. To see the other areas of the sheet, we must move this window around. To do that, we move the cursor.

To move the cursor, use the cursor control keys on the keyboard. The *Left Arrow* key moves the cursor to the left, *Right Arrow* to the right, *Up Arrow* up and *Down Arrow,* down. To move our "window frame" to another area, we move (or Scroll) the cursor past R1C7. When we move the cursor past Column 7, the screen displays the same 20 rows; however, now Columns 2–8 are shown. Strike the

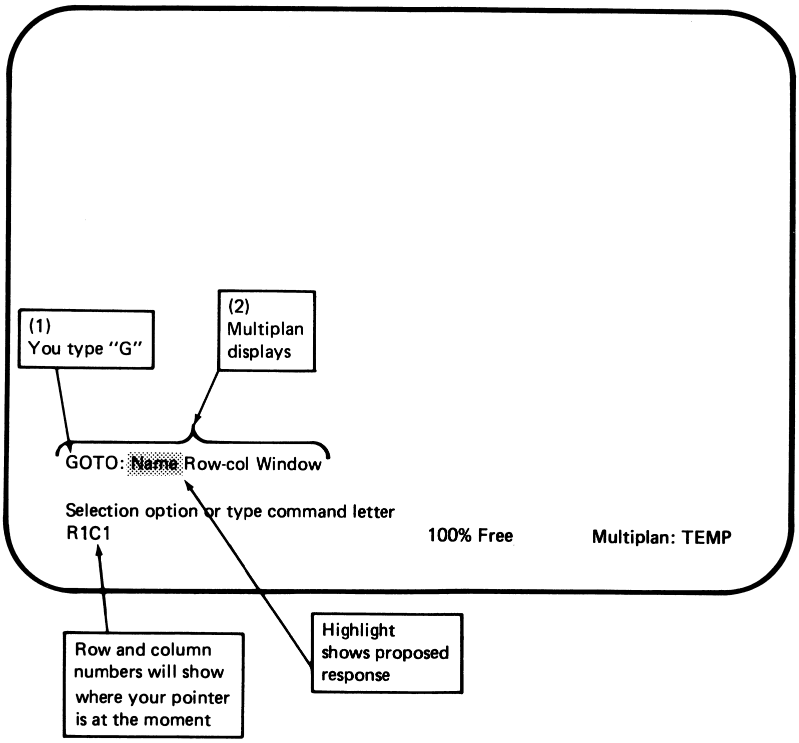

open window (number of active window illuminated)
cell pointer illuminated
screen
display
columns (1-7)
#1
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
rows (1-20)
command line
edit cursor (illuminated)
COMMAND: Alpha Blank Copy Delete Edit Format Goto Help Insert Lock Move
Name Options Print Quit Sort Transfer Value Window Xternal
Select option or type command letter
R1C1
100% Free
Multiplan: TEMP
Status line
Message line
storage space remaining
worksheet name

Home key and the display will again display Columns 1–7 with the cursor back at R1C1. The cursor movement beyond Row 20 is the same. The cursor may also be moved laterally by striking the SPACE BAR.

As it would be laborious to scroll through scores of rows and columns when a spreadsheet gets large, Multiplan allows for "page movement" through the spreadsheet. Strike the *CTRL* key and while holding it down, strike the *Right Arrow* key. The cursor will remain in the "home" position; however, Rows 1–7 will have become Rows 8–14. Striking the *Home* key or *CTRL—Left Arrow* will replace the cursor to the *Home* position at R1C1. Utilizing the *CTRL* key with the *Down* or *Up Arrow* keys will "page" the spreadsheet down or up by 20 rows.

One other cursor movement action in Multiplan may be used. Called the GOTO command, it is the most specific of those movements. In the illustration, note the Multiplan COMMAND LINE, containing the 20 primary program commands. The GOTO command will allow for cursor movement to any specific cell on the spreadsheet.

To implement GOTO, move the command line cursor to GOTO using the TAB key and strike the RETURN key, or simply type G. The latter method removes the necessity for the RETURN key. This holds true for all the Multiplan primary commands: Typing the first letter of the command will place the program automatically into that particular command. Displayed on the screen will be the following:


(1)
You type "G"
(2)
Multiplan
displays
Row and column numbers will show where your pointer is at the moment
Highlight shows proposed response

```
GOTO: Name Row-col Window

Selection option or type command letter
R1C1                                  100% Free         Multiplan: TEMP
```

The TAB, or the letter key representing the initial letter of the Multiplan command, are both appropriate for selection of any given command. In the event that an "illegal" key is struck, the command line will remain the same; however, a message, "Illegal option" will appear on the screen. Should the wrong command have been selected, simply strike the ESC key.

Upon selection of a given command, Multiplan will show a "proposed response." In the case of the GOTO command, the Row and Column fields will be filled with the "proposed response," which will be the current location of the cursor. (For some commands the proposed response will be a blank, while in others a "menu" will be shown.)

When a "proposed response" is not applicable to the user's requirements, simply change it, using the BACKSPACE, TAB, or SPACEBAR. If a response is correct, press the RETURN key to perform the command.

18

Multiplan$^{T.M.}$ Tutorial Part II

Before we learn to build a spreadsheet, the following explanation of "notation conventions" is necessary.

Standard typewriter keys are shown as such, e.g. A, B, C, 1, 2, 3, etc., with the exception of the SPACE BAR which is shown as ⟨sp⟩. The FUNCTION KEYS are illustrated by similar enclosures, e.g. ⟨F1⟩, ⟨F2⟩, ⟨F3⟩, etc. in conjunction, of course, with the ⟨Fn⟩ key.

The cursor control keys, those that move the cursor from cell to cell or page to page on the spreadsheet, will also be illustrated as special keys: ⟨Home⟩, ⟨Up⟩, ⟨Down⟩, ⟨PgUp⟩, ⟨PgDn⟩, ⟨Left⟩, and ⟨Right⟩.

The other special keys and their respective notations include:

The CANCEL command
⟨Bsp⟩: The backspace/delete key
⟨Tab⟩: The forward TAB key
⟨Ent⟩: The standard "carriage-return" enter key
⟨Ctrl⟩: The Control key
⟨Ins⟩: The Insert key
⟨Del⟩: The Delete key

When a keystroke is to be repeated a specific number of times, it will be indicated by that number in parentheses following the character to be repeated. For example, *(5) means "type 5 asterisks."

When entering formulas, we use either the "typing" method or the "pointing" method, whichever requires fewer keystrokes. However, the documentation shows the formulas as they actually appear in the cells of the spreadsheet, thus appearing to be the "typing" method of data entry.

Now we are ready for the production of a spreadsheet.

Now then, return the cursor to R1C1 with the Home key, and we'll use the construction of a Check Register to learn a bit more about Multiplan. By the way, if you should ever lose your place in the keying instructions and want to start a command all over, press *Esc* to cancel a command and return to the main menu.

Normally, Multiplan is set to recalculate an entire spreadsheet each time a cell entry is made. This can slow things down, particularly when all we want to do for the moment is create an empty sheet, called a *template*. To turn off the automatic calculation function, we set a Multiplan Option:

Press O for Options, then N for No Automatic Recalculation.

```
                           CHECK REGISTER                                        +
                          ================                                       + TOTALS BY CATEGORY
                                                                                 + =====================
Check  + DATE :   +                        enter 1 for:                          +
number + mm dd yy +            Payee       C/A Tax Othr  Amount  Deposit  Balance + Cred Acct   Taxes     Other
------------------------------------------------------------------------------------------------------------------
       +          + BALANCE FORWARD                                              +
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                        $0.00 +    $0.00    $0.00     $0.00
       +          +                                                              +
                                                                                 +
                                                                                 +
```

Now we will use the powerful *Format* command to do all kinds of things. Multiplan sets the format of all cells to certain defaults. These include the width of a column, the type of data a cell contains (simple numbers, dollar amounts, percentages, etc.), how the contents of the cell will be displayed (alignment within the cell and decimal positions to be displayed), and so forth. *Format* allows us to change any or all of these parameters. We begin by formatting the cells which will be used to hold the column headings. Some of these headings may extend across more than one column.

Press F for Format: Invokes the FORMAT menu which allows the user to format Cells, Default, Options or Width.

Press C to select the Cell option.

The next menu that Multiplan displays offers all the format options. It also contains the suggested cell reference: the cell that Multiplan assumes we want to specify a format for.

This cell reference will be R1C1. This is not sufficient for our needs here. We can complete the specification of the cells to be affected by this format command by keying: :R5C20. The cell reference field now contains: R1C1:R5C20. What we have done is define a rectangular block of cells by naming the upper left and lower right corners. To get to the next set of options, press the ⟨Tab⟩ key, and the cursor will jump to the next group of words in this menu.

Press C to cause Centering of cell contents.

Multiplan will automatically center the data in each cell of the block we specified. This makes nice-looking column headings.

(On the other hand, if number values were all centered in their cells, their decimal points would not line up, and the column would be hard to read. It would probably be inappropriate, therefore, to Center a column of numbers.) Now ⟨Tab⟩ to the next group of options.

Press C to permit Continuous text across columns.

If alphameric data is too large to fit in a cell, it is normally truncated to the size of the cell. Multiplan remembers the original data, however, so if the cell width is increased, more of the data will appear. (Numbers are not truncated: they are replaced with a line of asterisks. If the value were truncated, you might not be aware that there were additional digits not being displayed, and this could lead to errors.)

By specifying Continuous, we override the truncation feature. If the adjacent cell is empty, data will carry into it from the cell on the left. You will see how this works in just a moment. This completes the Format command. To tell Multiplan to execute the command, press ⟨Ent⟩.

We can summarize the keystrokes we just used as follows:

```
Cursor Location   Keystrokes
---------------   ----------------------------------------------
R1C1              F C :R5C20 ⟨Tab⟩ C ⟨Tab⟩ C ⟨Ent⟩
```

Now use the Format comand to set the column width:

```
R1C1              F W 6 ⟨Ent⟩
```

We selected the Width option instead of the Cells option, and set the width of column 1 to 6, thus replacing the default width of 10. When the ⟨Ent⟩ key is pressed, the entire spreadsheet "shrinks" to the left, as the width of column 1 is reduced.

This makes room for Multiplan to display column 8, which appears on the right of the screen.

Move the cursor to column 2 by pressing the ⟨Right⟩ key.

Now set the width of column 2 to six:

```
R1C2              F W 3 ⟨Tab⟩ ⟨Tab⟩ 6 ⟨Ent⟩
```

What happened here is that Multiplan offered only column 2 as the column to be changed. After entering the width desired, 3, we tabbed to the specification of the first column to be affected. Since that was offered correctly as "2," we immediately tabbed again to get to the specification of the last column to be affected. Multiplan was again suggesting "2," which we overrode by typing a "6." Then we executed the format command by pressing ⟨Ent⟩. As soon as ⟨Ent⟩ was pressed, the spreadsheet was squeezed down as columns 2 through 6 shrank. Since some of the columns are narrower, more can be shown on the screen, and columns 9 through 11 appear on the right. (Three, by the way, is the minimum column width allowed by Multiplan.)

Now move the cursor back to column 1 and down to row 4:

Press ⟨Right⟩, Then ⟨Down⟩ three times.

The cell reference should now read R3C1, and the cursor should be there as well.

With the cursor at R4C1, type "A" to force Multiplan into Alphameric entry mode. In this mode you can enter any combination of letters and numbers, and Multiplan will accept it as is. (In Alpha/Value mode, by contrast, Multiplan will not permit letters to be typed if a number is entered as the first character. This is because it will assume that a value is being entered, and trap the letter as an error.)

Notice that Multiplan supplies a prompt: "Enter text (no double quotes)." That is exactly what we want to do. Type "Check" but do not press the enter key. Instead, move the cursor down one row, to R5C1. Multiplan remains in data-entry mode (called Alpha/Value), and the prompt changes to: "Enter text or value."

Comply with this prompt by typing the word "number." You will notice that the moment you press the "n" of the word "number," Multiplan pops into Alpha mode. Now, still without pressing ⟨Ent⟩, move the cursor up a row and over one column, to R4C2.

Now type a space, then a plus sign. It is important to type the space first. This puts Multiplan back into Alpha mode, and the plus sign is accepted as a constant. If the plus sign had been entered as the first thing in the cell, Multiplan would switch to Value mode. In value mode, the plus sign must be followed by either a number or a Multiplan function name. Press *enter* to place the space and plus sign into the cell, and return to the main menu. The keystrokes we have covered so far in this chapter are:

Cell	Keystrokes
R4C1	A Check ⟨Down⟩
R5C1	number ⟨Up⟩ ⟨Right⟩
R4C2	⟨space⟩ + ⟨Ent⟩

We would like to create an entire column of the " + " entered in R4C2. This column will serve to separate the Check Number from the Date. It is just for appearance; it has no particular meaning to Multiplan. To create this column, we need only copy the contents of R4C2 into the cells below it:

Press C for Copy, and the first Copy menu will appear.

Press D to Copy Down from the current cell.

Enter 25 to Copy Down into 25 cells.

Press ⟨Ent⟩ to execute.

As this is your first use of *Copy*, the results should be pretty impressive. Now move right one column. (By the way, the more cells used in a spreadsheet, the longer it will take Multiplan to recalculate the sheet. If you know that you will not need 25 lines to hold your check information, you can copy down a smaller number.)

Return to Alpha mode and enter the heading "DATE:" with the keystrokes shown below. Note that we want the heading to be centered over columns 3, 4, and 5. To do this, we start with a space. Multiplan will extend DATE across the columns because we formatted them as Continuous. If we had not done this, DATE would be cut off, and would appear as: "DA".

Since we did not press ⟨Ent⟩, Multiplan is ready to accept more data in R5C3—follow the keystrokes below:

Cell	Keystrokes
R4C2	⟨Right⟩
R4C3	A ⟨space⟩ DATE ⟨space⟩ : ⟨Down⟩
R5C3	mm ⟨Right⟩
R5C4	dd ⟨Right⟩
R5C5	yy ⟨Ent⟩ ⟨Right⟩ ⟨Up⟩
R4C6	

With the cursor at cell R4C6 we will illustrate a second form of the Copy command.

Press C for Copy, then F for From.

Enter the reference to the column of "+": R4C2:R29C2.

Press ⟨Ent⟩ to execute.

When you pressed "F", Multiplan presented you with a menu that allowed you to specify either the "From cells," the "to cells," or both. Multiplan went further by filling in both options with the current cell reference. This is pretty smart: obviously you are not going to copy a cell into itself, but one of those two suggestions is likely to be correct. Therefore you will only have to correct one of Multiplan's offerings. In fact, this is exactly what we did.

Multiplan is also smart enough to understand that while we gave only one cell as the "to cell" reference, we wanted to copy an entire range of cells, as specified in the "From cell" reference. This saved us from having to figure out the "to cell" range. Properly understood, *Copy From* can be a real labor-saving command.

You have now learned the majority of Multiplan commands required to complete this template. Until new material is reached,we will just list the keystrokes used to build the Check Register. For the sake of completeness, the chart below will begin from the empty spreadsheet which Multiplan first displays.

Cell	Keystrokes	Comments
R1C1	O N ⟨Ent⟩	Turn off calculation.
R1C1	F C :R5C20 ⟨Tab⟩ C ⟨Tab⟩ C ⟨Ent⟩	Format headings.
R1C1	F W 6 ⟨Ent⟩ ⟨Right⟩	Set column width.
R1C2	F W 3 ⟨Tab⟩ ⟨Tab⟩ 6 ⟨Ent⟩ ⟨Left⟩ ⟨Dn⟩ (3)	Set widths of Columns 2 through 6.
R4C1	A Check ⟨Dn⟩	Set Alpha mode and enter headings.
R5C1	number ⟨Up⟩ ⟨Right⟩	
R4C2	⟨space⟩ + ⟨Ent⟩	
R4C2	C D 25 ⟨Ent⟩ ⟨Right⟩	Copy Down.
R4C3	A ⟨space⟩ DATE ⟨space⟩ : ⟨Down⟩	
R5C3	mm ⟨Right⟩	
R5C4	dd ⟨Right⟩	
R5C5	yy ⟨Ent⟩ ⟨Right⟩ ⟨Up⟩	
R4C6	C F R4C2:R29C2 ⟨Ent⟩ ⟨Right⟩ ⟨Up⟩ (3)	Copy From, to copy entire column.
R1C7	F W 25 ⟨Ent⟩	Make column seven 25 wide.

(Next we will set Alpha and enter the spreadsheet title. Notice that Multiplan centers the title in the column. This is a result of our first *Format* command.)

R1C7	A CHECK REGISTER ⟨Ent⟩ ⟨Down⟩	
R2C7	A=(16) ⟨Down⟩(3)	Alpha mode must be set here, because the equal sign normally will set Value mode on. The equal signs are used to underline the title.
R5C7	Payee ⟨Ent⟩ ⟨Right⟩ ⟨Up⟩	
R4C8	F W 4 ⟨Tab⟩ ⟨Tab⟩ 10 ⟨Ent⟩	
R4C8	A enter 1 for: ⟨Down⟩	This puts instructions right in the heading.
R5C8	C/A ⟨Right⟩	These categories keep track of Charge/ Accounts, . . .
R5C9	Tax ⟨Right⟩	Taxes paid, . . .
R5C10	Othr ⟨right⟩	and "other".
R5C11	Amount ⟨Right⟩	For check amount.
R5C12	Deposit ⟨Right⟩	
R5C13	Balance ⟨Ent⟩ ⟨Right⟩ ⟨Up⟩(4)	
R1C14	F W 3 ⟨Ent⟩	
R1C14	C F R4:R29C2 ⟨Ent⟩	Another column of "+".

Note

	⟨Down⟩ ⟨Right⟩	The way that the cell reference is entered: this is the same as, but takes fewer keystrokes than, R4C2:R29C2.
R2C15	A TOTALS BY CATEGORY ⟨Ent⟩ ⟨Down⟩	Set Alpha and enter a heading across three columns.
R3C15	A=(18) ⟨Down⟩ ⟨Down⟩	Underline the above.
R5C15	Cred Acct ⟨Right⟩	
R5C16	Taxes ⟨Right⟩	This will be centered.
R5C17	Other ⟨Ent⟩	
R5C17	G R 6 ⟨Tab⟩ 1 ⟨Ent⟩	The Goto command in action.

The next command makes use of the Multiplan REPT function, which will repeat the characters inside the double quotes the number of times specified. This will save many keystrokes. Since row 6 was not included in the first *Format Cont* command, the dashes fill only one column. So why have 25 of them? That's to fill the Payee column, after the *Copy Right* command.

R6C1	V REPT ("-", 25) ⟨Ent⟩	Please see text above.
R6C1	C R 16 ⟨Ent⟩ ⟨Down⟩ ⟨Right⟩(6)	Copy across.
R7C7	A BALANCE FORWARD ⟨Ent⟩ ⟨Right⟩(4)	

And now for some new Multiplan commands.

This seems like a good place to take a break. Before doing so, it is important to learn just one more command, to save the work you have done so far.

Press T for Transfer,

Then choose S for Save from the Transfer menu.

Now you must enter a name. May we suggest CHECKREG?

After typing the spreadsheet name, press *enter* and Multiplan will save your work to diskette. We advise switching to a second diskette and issuing the T and S commands again. This will provide a second copy in case something should happen to the first one. Notice that Multiplan remembered the file name from the first Transfer command.

If you wish to stop at this point, you can:

Press Q for Quit,

Then press Y to confirm your wishes to Multiplan. Bye!

19

Multiplan™ Tutorial Part III

Back again? Good. If Multiplan is running and you have inserted the diskette containing CHECKREG, then get Multiplan to reload the spreadsheet by using the *Transfer-Load* command. Type the file name. Now try *Transfer-Save*. Multiplan offers the last name used in a *Transfer* command, so you will not have to retype it. Don't bother saving now: cancel the *Transfer* command with the *Esc* key.

Multiplan has even remembered right where we left the cursor, at R7C11. Let's press on—there's not much left to do.

We should be in the column headed *Amount*. Since the amount will be used in some formulas, we will give this column a name.

Naming a cell or group of cells does a number of things:

1. Names are easier to type than cell references.
2. Names, if properly used, can help to make formulas easier to understand.
3. Names are the best way to make external Copy references.

We will explain the new terms used in just a moment. For now:

Press N to Name a cell or group of cells.

Type "amt" as the name, and ⟨tab⟩,

Type R7:33C11 as the cell reference, and press ⟨Ent⟩.

```
R7C11: N amt ⟨Tab⟩ R7:33C11 ⟨Ent⟩
```

Notice that the cell reference is Row 7, Column 11, through Row 33, Column 11. It can be entered with fewer keystrokes as R7:33C11.

Now for our first formula. Move the cursor right two cells and down a row, to R8C13: the Balance column. The balance should be the previous balance plus any deposit, or less the amount of a check. That is:

newbal = oldbal + deposit − checkamt.

Enter this formula as follows:

Press + (plus) to set Value mode.

Use the cursor-control ⟨Up⟩ key to point up one row. This will produce R[−1]C in the command line. That is a cell reference to the prior balance value.

Press the minus (hyphen). The cursor will return to the cell it started in.

Type "amt": we are telling Multiplan to subtract the amount of the check from the prior balance.

Press + (plus).

Press ⟨Left⟩. The cursor moves to the Deposit column, and RC[−1] appears in the formula.

Press enter. A zero should appear in R8C13.

The formula contains a type of cell reference called a *relative* reference. The relative cell reference RC[−1] says, "use the value in the cell one column to the left." A second kind of cell reference is called *absolute*. We do not have any absolute references in this spreadsheet. (Although named, references to *amt* are still relative.)

The formula should be: +R[−1]C−amt+RC[−1]. You can enter the formula by "pointing" to cells, as we did, or by typing the references out: the formula ends up looking the same with either method. But the cell contents do not look correct. All the amounts should be in dollars, so *Format* the cells by entering:

```
F C R8:33C11:13 ⟨Tab⟩ R ⟨Tab⟩ $ ⟨Ent⟩
```

Now we use the *Copy* command again to copy the Balance formula all the way through the column:

C D 20 ⟨ENT⟩

Just a little more: the Totals by Category. Move the cursor to R8C15. The formula will be the old total plus the check amount if—and only if—there is a "1" in the correct column.

Cell	Keystrokes	Comments
----------	--------------------------------------	--------------------------------------
Keystrokes so far in this lesson:		
R7C11	N amt ⟨Tab⟩ R7:33C11 ⟨Ent⟩ ⟨Right⟩(2) ⟨Dn⟩	Name the amt column.
R8C13	+R[−1]C−amt+RC[−1] ⟨Ent⟩	Balance formula.
R8C13	F C R7:33C11:13 ⟨Tab⟩ R ⟨Tab⟩ $ ⟨Ent⟩	Format cells as dollars.
R8C13	C D 20 ⟨Ent⟩ ⟨Right⟩(2)	Copy the Balance formula.
Continuing with the Totals by Category:		
R8C15	+R[−1]C+(amt*RC[−7] ⟨Ent⟩	The C[−7] refers to the column headed "C/A".
R8C15	F C ⟨Tab⟩ R ⟨Tab⟩ $ ⟨Ent⟩	Formatting before copying saves keystrokes.
R8C15	C D ⟨Ent⟩	Note that Multiplan even remembers the number from the last copy.
R8C15	C R 2 ⟨Ent⟩ ⟨Right⟩	

That last copy illustrates the power of the relative reference.

Notice that the reference is still to C[−7]. Now, however, seven columns to the left is not the C/A column, but the Tax column. And that is exactly what we want! Now:

R8C16	C D 20 ⟨Ent⟩ ⟨Right⟩	
R8C17	C D ⟨Ent⟩	Multiplan remembers the amount to copy from the last Copy.

That's it! Now *Transfer-Save* the spreadsheet to both diskettes.

You might think about changing the name to CHECKREG.MDL, to indicate that it is a template. That is, a model which does not contain data.

To keep the Check Register down to a more manageable size, we suggest breaking it up by some convenient period (monthly or quarterly, for example). Carrying a balance forward from one sheet to the next can be done easily by using eXternal Copy.

First, *Name* the last row of the spreadsheet when it is all filled in. (The sheet should have only the number of rows needed to hold the data, without extra rows at the bottom. This is because each added cell requires more diskette space and more time to recalculate the sheet.)

Then the next spreadsheet can use an eXternal Copy to fill in the BALANCE FORWARD amount automatically. For example, in the last row of the old sheet (let's say it's called CHECKS84.01), in the Balance column (C13):

```
N fwd ⟨Ent⟩
```

In the next sheet (CHECKS84.2) at R7C13:

```
X C CHECKS84.01 ⟨Tab⟩ FWD ⟨Tab⟩ ⟨Tab⟩ N ⟨Ent⟩
```

We specified No link. This means that Multiplan will get the cell(s) named *fwd* from CHECKS84.01 only this once. If we wanted to get the data each and every time we loaded CHECKS84.02 from diskette, we could have pressed ⟨Ent⟩ after typing "FWD".

When you want to print the Check Register, you will find that it is too wide to be printed on one line of an 80-column printer. It will, however, just fit into 132 columns. To set condensed print for any Epson printer, or the IBM Graphics Printer, use the Print Options as follows:

P M O ⟨Tab⟩ ⟨Tab⟩ 132 ⟨Ent⟩	Set Left margin to 0, and width to 132. Note that the ⟨Ent⟩ key takes us back to the Print menu, not the main menu.
O ⟨Tab⟩ ^ O ⟨Ent⟩	The caret (uppercase 6) and uppercase O put the printer in condensed print mode.
P	Print!

```
                      CHECK REGISTER                                                  +
                     ================                                                 + TOTALS BY CATEGORY
                                                                                      + =====================
Check  + DATE :   +                          enter 1 for:                             +
number + mm dd yy +          Payee           C/A Tax Othr  Amount   Deposit  Balance  + Cred Acct   Taxes     Other
------------------------------------------------------------------------------------------------------------------------
       +          + BALANCE FORWARD                                           817.43  +
  431  + 12 12 83 + The Big Company                       $100.22             $717.21 +     $0.00     $0.00     $0.00
  432  + 12 12 83 + The Little Company                     $23.14             $694.07 +     $0.00     $0.00     $0.00
  433  + 12 14 83 + The Kinda Medium Co.                   $62.17             $631.90 +     $0.00     $0.00     $0.00
  434  + 12 14 83 + Vizta                     1           $187.60             $444.30 +   $187.60     $0.00     $0.00
  435  + 12 14 83 + County Tax Office             1       $435.00               $9.30 +   $187.60   $435.00     $0.00
       + 12 19 83 + pay                                             $746.00   $755.30 +   $187.60   $435.00     $0.00
  436  + 12 20 83 + Grumbles Dept Store       1           $214.56             $540.74 +   $402.16   $435.00     $0.00
```

A final word:

We have barely scratched the surface of Multiplan's capabilities; there are many more features and tricks to using Multiplan than we have been able to cover here. However, the primary intent of this chapter was to give you an idea of how to use a spreadsheet program. At one time it was said that people bought a personal computer to have something to run their spreadsheet program on: the $200 program sold the $2,000 machine.

There is no doubt that automated spreadsheets can be a powerful productivity aid. Give them a try!

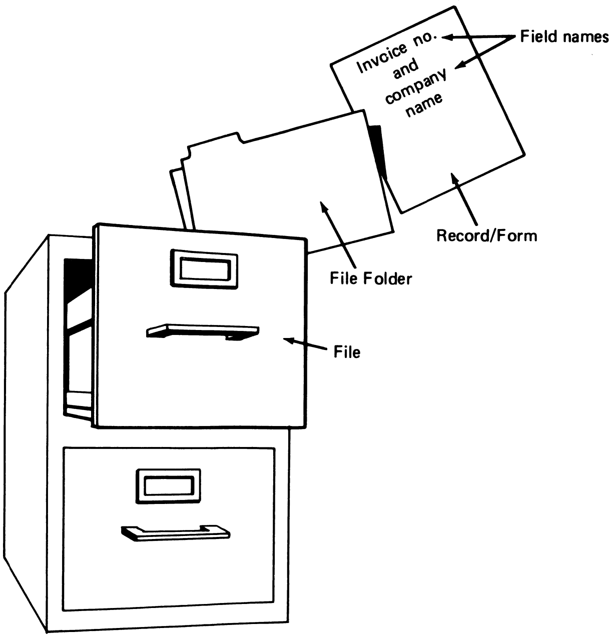
20

pfs FILE and Other File Managers

File managers are simply computer programs that allow you to store and retrieve information as you would from that old three-drawer metal file cabinet in your home or office. To help you understand the concept of a computer file manager, some simple definitions are in order.

The program itself, whether it be *pfs:file* or any other file manager, is the file cabinet or drawer. Inside the drawer are file folders. With the computer, think of these as *files*. A file manager's file might contain invoices. It would be labeled INVOICES. In the INVOICES file would be *records* or *forms*. Essentially, these records are the same as the individual invoices kept on sheets of paper inside a folder in a file drawer. Finally, on each record are *fields* which contain the appropriate data for each record. In the case of an INVOICE file, fields might contain an invoice number, company name, product shipped, address, balance due, and other information.

With your *PCjr* and a file manager such as *pfs:file*, it's possible to keep a great many records in a file, sort them in a variety of ways (by ZIP code, name, product or whatever you might choose within the program's limits), and retrieve them to your screen or printer based on various search criteria you specify. Also, *pfs:file* will print individual forms to various user-specified criteria; with its companion program *pfs:report* it will print user-specified tabular reports and calculate numeric information.

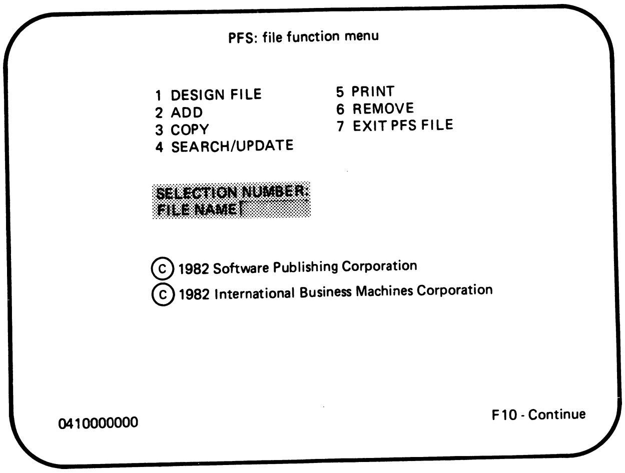


Both *pfs:file* and *pfs:report* will run on *PCjr* with the one-disk drive and 128K configuration. According to the programs' documentation a printer is optional; in reality, however, it is almost a necessity.

With both programs there are some drawbacks to the one-drive system. In *pfs:file* you will not be able to use the COPY or *Change Design* functions with particularly large files because the program requires workspace on the program diskette. With *pfs:report*, one-drive will prevent the user from sorting long reports for the same reason.

The documentation for *pfs:file* is tutorial in nature and requires little effort on the user's part; however, let's walk through a bit of it just to give you an idea of its ease and perhaps help you to determine whether or not a purchase is in order.

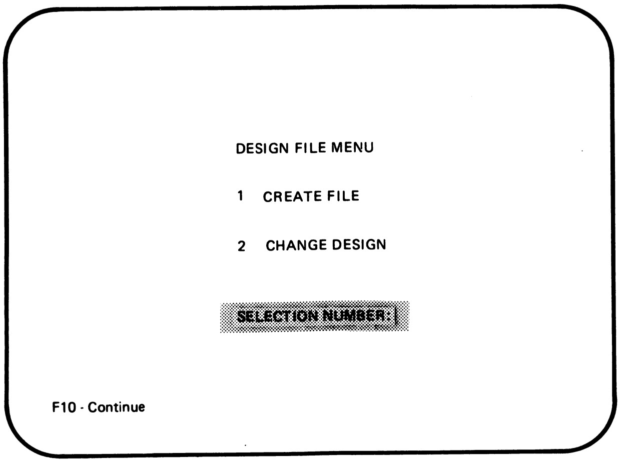
Once the *pfs:file* diskette is configured to be self-loading, it is necessary to back it up. With *PCjr's* one drive you will, of course, require your original *pfs:file* disk and one blank formatted disk for the backup. In the backup process, which is essentially the same for both *pfs:file* and *pfs:report*, be prepared to swap your program disk and backup disk in and out of the drive at least 20 times. In the long run, the effort is well worth it.



```
                    PFS: file function menu

        1 DESIGN FILE          5 PRINT
        2 ADD                  6 REMOVE
        3 COPY                 7 EXIT PFS FILE
        4 SEARCH/UPDATE

        SELECTION NUMBER:
        FILE NAME:

        (c) 1982 Software Publishing Corporation
        (c) 1982 International Business Machines Corporation

0410000000                                      F10 - Continue
```

The backup disk will be used for *pfs:file's* operation. To start the program, insert the disk in the drive and turn on the computer. If the system is on, strike *Ctrl-Alt-Del* and you will eventually see *pfs:file's* Function Menu.

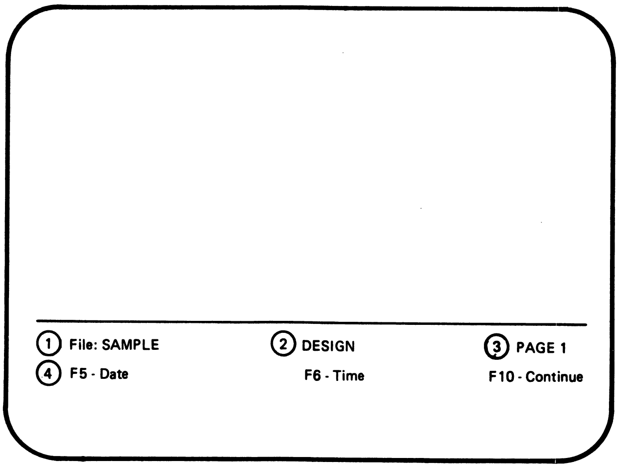
The first step is to design a file, selecting option one from the menu, then pressing the *F10* key to "continue." The program requires *F10* to implement any of its functions. Here we will create a simple file of names and addresses and call it "Friends." After the selection, *pfs:file* pops up another menu:



```
                    DESIGN FILE MENU

                    1   CREATE FILE

                    2   CHANGE DESIGN

                    SELECTION NUMBER:

F10 - Continue
```

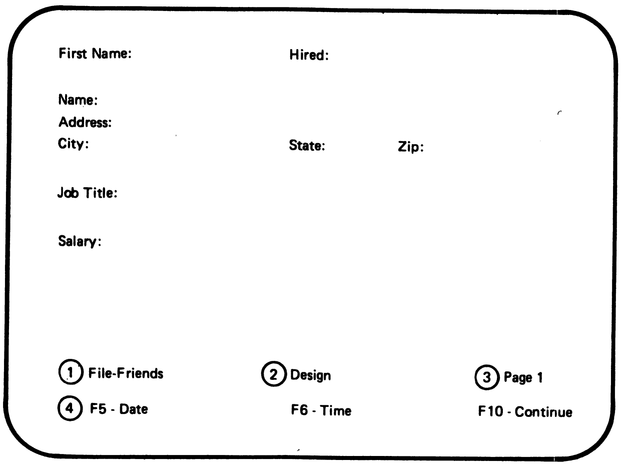
from which we again select number one, *Create File.*

The *pfs:file* screen will show on the display and we are ready to create the form to be used in our "Friends" file.



1 File: SAMPLE	2 DESIGN	3 PAGE 1
4 F5 - Date	F6 - Time	F10 - Continue

In order to create the form, fields must be designated. With *pfs:file,* this is a snap. Simply move the cursor, using *PCjr's* cursor movement keys to the spot on the screen where the field will be placed. The *Enter* key is used to move the cursor to the beginning of the next line.



```
First Name:                 Hired:

Name:
Address:
City:                       State:        Zip:

Job Title:

Salary:
```

1 File-Friends	2 Design	3 Page 1
4 F5 - Date	F6 - Time	F10 - Continue

Note in the illustration how each field is followed by a colon (:). The *pfs:file* requires this to let it know the beginning of the data entry area. In addition, the user must be sure to allow enough space after the field name for the longest piece of data that will be placed in the field. The fields can be moved around, modified, or deleted prior to entering data and you can even have multi-page forms should they be required.

Once the form is completed, store it so it can be recalled later for data entry. The *F10* key is used to accomplish this; strike it and *pfs:file* returns to its Main Function Menu.

Adding data to the form requires little more than selecting the *ADD* function on the Main Menu, striking the *F10* key, and typing in the data. Once the form is complete, the *F10* key is again struck and the screen filled with another blank form.

Once a number of forms are entered in the file, *pfs:file* has functions that allow the user to retrieve the data in a number of ways. By selecting the *Search/Update* function from the Main Menu, you may, for example, search for a specific ZIP code. In this case, the selection of a "numeric item match" might be in order. Using this option, *pfs:file* allows a search for ZIP codes (or any numbers, for that matter) equal to, greater than, or less than the specified number. Say we wanted all names in the ZIP code 20002: On the blank form displayed after the search selection is made, an entry of "=20002" would be made and F10 would be hit. The *pfs:file* then goes to work displaying the forms based on the criterion selected.

Similar searches can be accomplished based on a full item match, used in finding a specific item; a partial item match used for retrieving several pieces of information; and a "not" match specification—selection of all items that do not include a specific name or number.

The *pfs:report* operates much like *pfs:file*; it allows the user to define how data from *pfs:file* will be printed out and then prints it.

The program is able to produce a report composed of up to 16 vertical columns and an unlimited number of rows, with each column corresponding to a field in a *pfs:file* file. Each row in the report will contain the data stored on a single *pfs:file* form. The *pfs:report* can sort the rows either alphabetically or numerically and can do various numeric calculations.

The *pfs:file* may be used without *pfs:report*; however, the reverse is not true. If hard copy is needed—and with file managers it usually is required—it will be easier and more efficient to use the two programs in conjunction.

Let's not forget the ubiquitous FREEWARE. There is an excellent file in the public domain and in wide use; it's called PC-FILE. It will, in fact, do just about everything *pfs:file* and *pfs:report* will do and do it well. Moreover, it combines the capabilities of both, allowing for report printing and file management in one program.

PC-FILE, at this writing, will not allow the user to design forms on the screen as will *pfs:file*, nor will it print out reports in the variety of ways accomplished by *pfs:report*. It will, however, sort faster than both programs and allow the user to redefine certain keys on the keyboard. PC-FILE may be obtained by sending a blank formatted disk (or $35.00) to Jim Button, P.O. Box 5786, Bellevue, WA 98006. Again, the contribution is preferable to the hassle of disk mailing.

PC-FILE III, a significantly enhanced version of PC-FILE, is now generally available on bulletin board systems throughout the country. It may be obtained directly from Jim Button for $45.00.

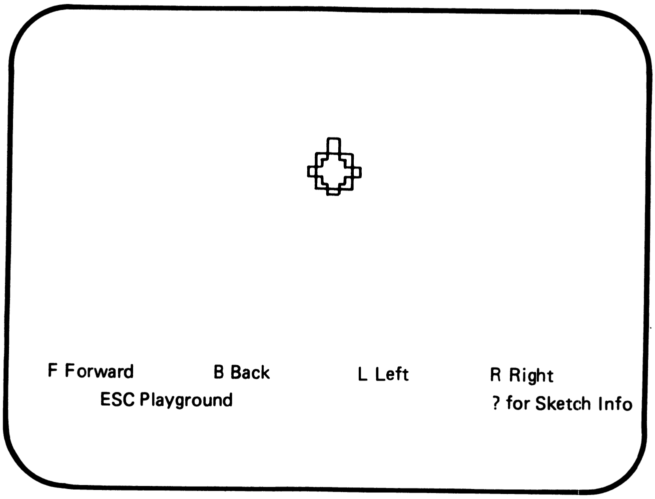
21

Games and Other Programs for the Pcjr

Though you will be reasonably safe in assuming that the 128K, one-disk drive requirement for software compatible with the *PCjr* will hold for many packages available, you must exercise caution in purchasing software requiring graphics capabilities. For various technical reasons, many high-resolution graphics games available for the IBM PC will *not* run on *PCjr.* Among those that will not run on *PCjr* are *Flight Simulator*™ by Microsoft® and *Night Mission Pinball*™ by SubLogic™. We mention these two because of their extreme popularity and wonderful graphics on the PC. For both of those reasons, you can be reasonably sure that they will appear in *PCjr* form sometime in the near future.

Many graphics programs, games, and educational software are available for *PCjr*, however. In this chapter we give you a sampling, or taste of what to anticipate. In other words, if you're an impulse buyer, you might want to read this before your plastic or cash begins to fly.

LOGO™ is a computer language, like BASIC (or any other computer language, for that matter), that is really oriented towards children. Although it is not restricted to the younger set, it is often used in schools to give kids a foundation in programming and also to teach them reason and logic. *LOGO* is also known as "Turtle Graphics" because it uses a character that looks like a turtle to draw various pictures on the screen.


F Forward B Back L Left R Right
ESC Playground ? for Sketch Info

LOGO is more likely to keep your kids, or you, at the computer learning about computer programming than BASIC because it does use graphics and it is considerably easier to follow than BASIC.

With *LOGO* you have the capability of developing programs and graphics and saving them for reuse later. Those programs can range from simple telephone and address directories to complex and beautiful graphics designs.

For the *PCjr* the package requires one disk drive, a color display or television, and 128K.

BUMBLEPLOT® takes advantage of the *PCjr*'s graphics and sound capabilities. It is an educational program for children aged 8 to 13. It is a collection of five games through which children are introduced to various mathematical skills. In "Trap and Guess," the player attempts to "trap" a number between +3 and −3 generated by the computer. The player enters the two numbers between which he or she believes the number lies, then guesses or continues to narrow the parameters. "Bumblebug" is a similar game, a bit more sophisticated, in which the player traps a "bug" within a numbered grid. "Hidden Treasure" expands the theme to higher numbers utilizing X and Y axes and coordinates through which the player seeks 16 different treasures.

In "Bumble Art," the player develops geometric designs using a grid and in "Roadblock" the player uses a more advanced grid to play a cops and robbers type of game.

BUMBLEPLOT requires a *PCjr* with a disk drive, 128K of memory, and cartridge BASIC. It may also be run on the IBM PC or PC XT with one disk drive, 64K of memory, and a color/graphics monitor adapter.

SCUBADVENTURE is an arcade-like game requiring only the minimal *PCjr* cartridge system. Players control a three-diver expedition team which encounters the various tribulations of scuba diving. The team searches out treasure and rare tropical fish.

MOUSER involves building a better mousetrap with the *PCjr.* The player is a farmer in a multi-room farmhouse overrun by mice. Trapping the mice is the object (cartridge system only).

In *CROSSFIRE,* you defend the streets of a city from an invasion of insects. All you've got are three ships with insecticide missiles (cartridge system only).

In *MINESHAFT,* the player searches for diamonds through treacherous mineshafts and must fend off runaway robot miners. You get points for collecting diamonds and disabling the robots (cartridge system only).

MONSTER MATH is an educational program that teaches addition, subtraction, multiplication, and division on six different levels. It's a timed program that has a bit of entertainment in that a monster appears on the screen until the problem is solved. This program requires a disk drive and 128K of memory.

ANIMATION CREATION is a colorful graphics program that allows the user to create various animation sequences, save them, and display them in rapid sequence as if from a projector. This program requires a disk drive and 64K of memory.

JUGGLES BUTTERFLY is a preschool program that assists children in acquiring reading and math readiness skills. The set of three game programs teaches various concepts such as left and right and above and below while introducing the preschooler to the computer.

22

Down to BASICs

We will not be trying to teach BASIC in this book: it deserves (and requires) a book of its own. However, if you already know BASIC, or intend to learn it, there is some information about *PCjr* BASIC—the cartridge BASIC in particular—that you should know. Many people ask if BASIC is a "universal language" for all personal computers. The answer is that each computer manufacturer has implemented different versions of the language. IBM, not to be outdone, now offers *five* versions of BASIC! These versions of BASIC are as follows:

Cassette BASIC. Cassette BASIC comes inside all IBM PC computers, as part of their Read Only Memory. (See Chapter 8 "Words About Words" for definitions.) All IBM PC cassette BASIC's appear alike. That is, they have the same set of specifications, even though they may do things differently internally.

Standard BASIC. Standard BASIC is a program supplied for the PC and PCXT on the DOS system diskette. It adds diskette file support to the ROM (cassette) BASIC, plus a few other functions.

Advanced BASIC. Advanced BASIC (for the PC and PCXT) adds more graphics functions, additional single-voice sound features, and "event trapping" ("ON . . ." statements). It is called BASICA on the DOS system diskette.

Compiled BASIC. Compiled BASIC requires use of the BASIC Compiler program product. In some ways it offers fewer functions than Advanced BASIC (e.g., no built-in full-screen editing), but Compiled BASIC programs tend to run much faster than any of the other varieties. Compiled BASIC will run on any of the IBM personal computers.

Cartridge BASIC. Cartridge BASIC (for *PCjr* only, of course, since only the PCjr takes cartridges) offers three-voice sound and an advanced graphics system which takes advantage of the improved graphics abilities of the *PCjr.*

If you will be writing BASIC programs, you should be aware of the differences between the versions. It is nice to make use of the extra graphic modes of the *PCjr,* but if you do so, your program will not run on other models of the PC. If you are writing a program which does not absolutely require graphics, then stick with Standard BASIC: many PC's do not have graphics adapters, and therefore cannot run BASICA programs which use the graphics commands.

Summaries of the commands which are either unique to, or different in, Cartridge BASIC are provided below. The summaries are provided as a quick reference to the new or changed commands: the BASIC manual is the best source of specific details. Additional information about sound and graphics will be found in Chapter 24 on "Advanced Sound" and Chapter 23 on "Advanced Graphics."

BEEP statement. The ON and OFF parameters have been added. When BEEP is ON, sounds (from BEEP, SOUND, and PLAY statements) are sent to the internal speaker. When BEEP is OFF, no sounds go to the internal speaker. (Also see SOUND.)

NOISE statement. Sends selected combinations of frequencies to the external speaker. Requires that a SOUND ON statement be executed first. Parameters of the NOISE command are

Source: A value from 0 to 7. Values 0 to 3 produce a "drone" at a distinguishable frequency, sounding like a note with distortion. Values 4 to 7 give a steady hiss. When source is either 3 or 7, the noise frequency is determined by the frequency used in voice 3. The source values 0 to 2 and 4 to 6 produce frequencies from higher to lower.

Volume: A value from 0 to 15.

Duration: A value from 0 to 65535, representing the number of clock ticks that the noise should last. The IBM PC clock ticks 18.2 times per second.

The BASIC manual does not mention that the values must be integers. In fact, the example in the BASIC manual is incorrect. The correct NOISE program is

```
10 SOUND ON
20 FOR N=0 TO 7
30 NOISE N,15,250
40 PLAY "","","V0"
50 FOR I=1 TO 6
60 PLAY "","","V15;O=i;CDEF"
70 NEXT I
80 NEXT N
```

1. An explosion:

```
10 SOUND ON' explosion
20 FOR V=15 TO 0 STEP -1
30 NOISE 6,V,1
40 NEXT V
```

2. "Laser fire" (Note the use of voice 3 to control the NOISE frequency. SOUND must have a volume of at least 1, even if it is not supposed to be heard. Also, the durations specified in the SOUND and NOISE statements should be identical.):

```
10 SOUND ON' laser fire
20 FOR V=14 TO 4 STEP -2
30 SOUND V*20,1,V,3
40 NOISE 7,V,1' freq. from voice 3
50 NEXT V
```

ON PLAY statement. Specifies a subroutine to be executed when the music buffer falls below the specified number of notes. (Also see PLAY(n) function.) This statement can be used to insure continuous music during program execution. The ON PLAY statement is used in conjunction with the PLAY ON/ PLAY OFF statements. PLAY ON must be active to cause the

ON PLAY statement to work. Once an ON PLAY has been executed, PLAY ON and PLAY OFF can be executed to turn the trapping on and off. The format is:

```
ON PLAY(n) GOSUB line
```

Where: n is a value from 1 to 32, specifying how many notes should be left in the buffer when the trap occurs.

Line specifies the first line of the subroutine to be executed.

PALETTE statement. Allows redefining of a palette attribute. The operands are (attribute1,attribute2). For example, the statement "PALETTE 1,3" alters the meaning of color 1 from its normal setting of blue to the color cyan. All references to COLOR 1 from that point on will refer to cyan. (Also see "Advanced Graphics," Chapter 23.)

PALETTE USING statement. Offers a more flexible way to set palette entries. The operands are

arrayname(start): An array of 16 integers, indexed from the value of "start." An entry with a value of − 1 will not modify the current color for that attribute.

PCOPY statement. Copies the contents of one screen page to another. See "Advanced Graphics" for more information.

PLAY(n) function. Returns the number of notes left in the music buffer specified by *n*. If *n* is not 0, 1, or 2, then then number of notes remaining in voice buffer 0 is returned. (Also see ON PLAY.)

PMAP function. Used to translate between "physical" and "world" coordinate systems. (See the VIEW and WINDOW statements and "Advanced Graphics," Chapter 23.) By using PMAP, the program can draw an object without regard to the logical limits of the screen, then map the drawing to the actual viewing area.
The options of the PMAP function are:

z = PMAP (x,0): change world *x* coordinate to physical *x*.
z = PMAP (y,1): change world *y* coordinate to physical *y*.
z = PMAP (x,2): change physical *x* coordinate to world *x*.
z = PMAP (y,3): change physical *y* coordinate to world *y*.

SCREEN statement. New modes have been added. Please refer to "Advanced Graphics."

SOUND statement. A "voice" operand (values 0, 1, or 2) has been added. If this operand is used, a SOUND ON must have been executed first.

SOUND ON/OFF statement. Turns on (off) sound to the external speaker and enables (disables) multivoice sound functions. SOUND ON/OFF operates independently from the BEEP statement, so sounds can be directed to internal or external speakers, both, or neither.

TERM statement. Run terminal emulation program. This is a complete BASIC program, built into the cartridge ROM. As a result of entering the TERM statement, or executing it in a program, any old program is erased from memory and the Terminal Emulator is loaded into memory and run. The program will check for and use the internal modem if it is installed. When it is first executed, the program will display a menu which allows you to select:

line speed (if an external modem is being used)
either 7 or 8 data bits
parity
host echoing
screen width (this applies only to 128K machines—64K machines are limited to 40-column screens)
the setting of *F1* for modem commands (internal modem only).

(For more information on the meaning of these and other communication terms, please refer to Chapter 12.)

VIEW statement. Defines "viewports" which divide the display screen into separate areas. See also WINDOW (below), and "Advanced Graphics." The VIEW operands are:

SCREEN: If the SCREEN option is omitted, then all points to be displayed are relative to the viewport settings. That is, coordinate values are added to the viewport *x*- and *y*-axis coordinates, and must lie within the viewport area. When the SCREEN option is used, all coordinates are absolute, and any values which lie beyond the boundaries of a viewport are simply not displayed.

(*x*1,*y*1) - (*x*2,*y*2): The coordinates of the viewport's upper left and lower right corners. By specifying opposite diagonal corners, a rectangular viewing area is defined.

attribute: (Optional) Viewport background color from 0 to 15 or 0 to 3, depending on graphics mode in use.

boundary: (Optional) Draws a border of the specified color (same values as "attribute"), space permitting.

The following example creates two overlapping viewports, each with its own square:

```
10 KEY OFF:CLS:SCREEN 1,0
20 VIEW (0,0)-(200,100),1,2
30 GOSUB 100
40 VIEW (150,20)-(280,80),2,3
50 GOSUB 100
60 END
100 REM Draw a square
110 LINE (5,5)-(100,5)
120 LINE (100,5)-(100,50)
130 LINE (100,50)-(5,50)
140 LINE (5,50)-(5,5)
150 RETURN
```

WINDOW statement. (Not new with the *PCjr*, the WINDOW statement has been included here for the sake of completeness.) Redefines the coordinate system used for graphics. Screen coordinates are normally predetermined in BASIC by the graphics mode in effect (set by the SCREEN statement). In SCREEN 2, for example, the upper left corner is always specified as (0,0), while the lower right corner is always (639,199). WINDOW allows you to specify any range of coordinates. For example, the statement:

```
WINDOW SCREEN (-1000,0)-(1000,500)
```

defines a window (screen) which has an *X*-axis extending from − 1000 to + 1000, and a *Y*-axis covering from 0 at the top to 500 at the bottom. If the SCREEN option had been omitted, the greater values for the *Y*-axis would be toward the top of the screen; i.e., just the reverse of the statement shown.

WINDOW permits the program to draw in a coordinate system best suited to it, instead of forcing the program to translate its coordinates into the standard SCREEN statement default values. Also see VIEW (above) and "Advanced Graphics."

The following demonstration program uses WINDOW to produce a three-dimensional drawing effect:

```
10 KEY OFF:CLS:SCREEN 1,0
20 X=100
30 FOR Q=1 TO 50' zoom the window
40 WINDOW (-X+Q,-X+Q)-(X-Q,X-Q)
50 GOSUB 100
60 NEXT Q
70 END
100 REM Draw a square
110 LINE (Q,Q)-(X/2,Q)
120 LINE (X/2,Q)-(X/2,X/2)
130 LINE (X/2,X/2)-(Q,X/2)
140 LINE (Q,X/2)-(Q,Q)
150 RETURN
```

23

Advanced Graphics

Computer-generated graphics is an exciting field which has applications in business, science, industry, and entertainment. The *PCjr* offers a wide variety of graphics capabilities, including three new graphics modes not available on the IBM PC or PCXT. (This is due to an enhanced version of the PC Color/Graphics Adapter, which is offered as part of the 64KB Memory and Display Expansion Option.) These new modes are supported by cartridge BASIC, but not by BASICA 2.1.

Display Types

To take full advantage of any of the 16-color modes or the 640 × 200 four-color mode, a high-quality video output device called an RGB monitor is required. A brief discussion of the kinds of video devices which may be attached to the *PCjr* follows.

Video output from the *PCjr* can be directed to three types of displays: an ordinary television (color or black-and-white), a composite monitor, or an RGB monitor.

The highest quality (and most expensive) of these is the RGB monitor. *RGB* stands for Red-Green-Blue. This device accepts separate signals for each of the three primary video colors produced by the *PCjr.* The result is the sharpest picture possible from your computer, since the least amount of extra circuitry exists between it and the screen. RGB monitors also have a greater "bandwidth," which means that they will produce a wider range of colors in addition to a sharper picture. A special cable (available from IBM) is required to attach the *PCjr* to an RGB monitor.

At the other end of the spectrum is the ordinary TV. TV signals consist of audio and video information "modulated" by the addition of a radio frequency, or RF. The *PCjr* has an option which permits attachment to a TV. This option adds RF modulation, only so that the tuner in the TV can remove it. This added step causes some loss of signal quality. A television set will not reproduce colors—and, in some cases, shapes—accurately in the 80-character and 16-color graphics modes.

The composite monitor takes a signal of combined ("composite") red, green, and blue signals. By avoiding the RF modulation, greater clarity and color accuracy are possible. Nevertheless, a composite monitor cannot compare with the improved bandwidth of an RGB device. The display of a component TV system can be used as a composite monitor. By the same token, if you select a composite monitor for your *PCjr*, it can also be used to produce a beautiful TV picture, when attached to an external tuner (such as that found in a video tape recorder).

The following charts present the graphics capabilities of the IBM Personal Computer family.

IBM Personal Computer Display Modes

Graphics Mode	*Colors*	*BASIC SCREEN*	*Memory used*	*Compatible with PC?*	PCjr *needs expansion?*
40 character	B&W	0 (1)	2K	Yes	No
40 character	16	0 (1)	2K	Yes	No
80 character	B&W	0 (1)	4K	Yes	Yes
80 character	16	0 (1)	4K	Yes	Yes
160 × 200	16	3	16K	No	No
320 × 200	4	1 (2)	16K	Yes	No
320 × 200	4	4 (2)	16K	No	No
320 × 200	16	5	32K	No	Yes
640 × 200	B&W	2	16K	Yes	No
640 × 200	4	6	32K	No	Yes

1. SCREEN 0 is set to 40 or 80 characters by the WIDTH statement.
2. SCREEN 1 and SCREEN 4 differ in the way that colors are selected.

***PCjr* Graphics**
(Not available on other PC's)

Mode	*Colors*	*Expansion*
160 × 200	16	
320 × 200	16	Yes
640 × 200	4	Yes

Display Concepts

The inside of the face of a picture tube is covered with phosphors which glow when struck by electrons. A color tube uses different types of phosphors, grouped into tiny patterns of "color sub-dots," which produce different colors. The electrons are fired from an "electron gun" in a tight beam.

A picture is "painted" on a display screen by the actions of the electron beam. The beam sweeps left to right across the screen to produce a single line of the picture, then quickly returns to the left edge a bit lower down to produce the next line, and so on. These lines are called "scan lines." The electron beam is so accurately aimed that it can hit the correct phosphors within the color sub-dots to produce the desired colors.

After 200 scan lines are drawn, the beam returns to the top of the screen to start the process all over again. (This discussion is limited solely to the IBM PC video system. Other computers may use fewer or greater numbers of scan lines, while television always interlaces a greater number of lines.)

While the screen always contains 200 lines, the number of graphics points (called *pixels* for "picture elements") in each line may be set by the program (from 160 to 640). The more discrete points used to create an image, the better the picture quality, or *resolution*. The greater the resolution, the more memory is required (see chart).

From Computer to Screen: Memory Mapping . . .

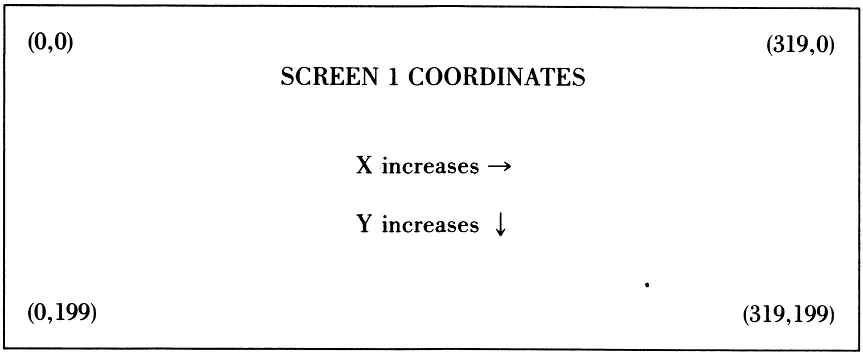
Two different schemes are used to produce a display. The first is used by the character modes. In the character mode, the computer uses the ASCII code for each character in display memory to look up the definition of the character shape. The display hardware then sends the appropriate character shape to the video output. A standard set of characters for the first half of the character set is contained in ROM, and cannot be altered. However, you can supply alternate definitions in RAM for the second half of the character set, and direct the display driver to use them instead. Thus, the Greek "alpha" can be made to look like a spaceship or a "Pacman." Each character is 8 scan lines high, giving a maximum of 25 lines in character mode (200 total scan lines divided by 8 scan lines per character).

The graphics modes use a technique called *memory* (or *bit*) *mapping*. This term means that there is a direct correspondence between the contents of computer memory and what appears on the screen. A "one" bit results in a lit pixel; a "zero" bit produces a dark spot. The computer program directly controls what appears on the screen, without any additional actions on the part of the video driver. This has the advantage over character modes of increased control over the picture. The disadvantages lie in increased memory requirements and programming effort.

Each graphics pixel is "individually addressable." What this means is that a program can manipulate each and every pixel on the screen, because each and every pixel has a unique identifier.

In SCREEN 1, for example, there are 320 × 200, or 64,000 separate points which the program can turn on or off. These could be numbered 0 to 63,999, but they are not. Instead, graphics modes use a coordinate system. In this system, a pixel is normally identified (or "addressed") by specifying its horizontal distance from the left border of the display and its vertical distance from the top of the display. The coordinates are usually written as *(x,y)*, where *x* is the horizontal value and *y* the vertical. This makes the upper left (0,0). In SCREEN 1, the lower right corner is (319,199).

BASICA and cartridge BASIC allow the programmer to reorient the entire screen coordinate system any way he or she chooses, using the WINDOW statement. This can be very useful, for example, when plotting mathematical functions. For the purposes of discussion, however, the variations created by the WINDOW statement would only confuse the issue. Hence, we will continue to think of the screen as illustrated below.



A display screen is not square. Furthermore, each pixel has this same characteristic, called the *aspect ratio.* What this means to the computer artist is that a drawing 100 pixels high and 100 pixels wide is not square. The aspect ratio of the IBM display is roughly 3:5. That is, the picture is about three-fifths as high as it is wide. As the horizontal resolution doubles, so does the aspect ratio. Thus:

Mode	*Resolution*	*Aspect Ratio*
Low	160 by 200	3:5
Med	320 by 200	6:5
High	640 by 200	12:5

In low-resolution mode (160 by 200), this can be translated into graphics terms as: a square which is 100 pixels high must be 120 pixels wide. You will have to adjust the coordinates of your drawings to allow for the aspect ratio.

... and a Dash of Color

For the sake of simplicity, the discussion of memory mapping ignored the issue of color. The IBM computer handles color information in rather complicated ways. While the BASIC programmer may not have to know how the computer handles color, it probably would be helpful to understand some of the details.

In character mode, color information for each character is stored in computer memory immediately following the character code.

What this means is that the characters you see on the screen are contained in every other byte (the even-numbered ones) of display memory. The odd-numbered bytes contain a value which the video circuitry interprets as foreground color, background color, intensity, and whether to blink the character or not. The eight bits in the color byte are:

Blink	Background colors:			Intensity	Foreground colors:		
	Red	*Green*	*Blue*		*Red*	*Green*	*Blue*

The red, green, and blue color components can be combined in combinations to give seven different colors (seven being the largest value that can be represented in three bits: read about *binary arithmetic* in Appendix 2 for more information). When the Intensity (brightness) bit is added to the color information, the eye interprets 16 different colors.

Luckily, BASIC relieves us of much of the worry of knowing how this happens. You will notice that the BASIC COLOR statement for text permits the values 0 to 15 for foreground, but only 0 to 7 for background. Now you know why: the Blink bit uses up what might otherwise have been a Background Intensity bit.

In fact, you can change the meaning of this bit to "Background Intensity," thus giving 16 background colors: in 40-column width: OUT and H3D8, 8; in 80-column width: OUT and H3D8, 9

Color in the graphics modes is handled differently. Obviously, there is not enough computer memory to store one color byte per pixel: that would be almost 64K just for the color bytes in medium resolution!

So another method is used for determining the color of a pixel. Depending on the graphics mode selected, a pixel is stored in memory as either two or four bits. For four-color modes, only two bits are required (to hold the values 0 to 3). For 16-color modes, half a byte is used. If you refer back to the Graphics Modes charts at the beginning of this chapter, you can see how this all hangs together.

Four-color modes can store four pixels per byte. A medium resolution screen has 320 times 200, or 64,000 pixels. Hence, at four pixels per byte, the display requires 64,000 divided by 4, or 16K-bytes of memory. When the medium-resolution 16-color mode is used, only two pixels can be defined in a byte. Hence the 64,000 pixels require 64,000 divided by 2, or 32K-bytes of memory. Thus, with 16-color modes it is easy to specify any color that the *PCjr* is capable of producing. In four-color modes, however, the question arises, "Which four colors?" Enter the PALETTE.

Cartridge BASIC permits the programmer to pick any four colors and assign them to the two bits used to hold color information. The two bits are then referred to as attribute bits, instead of color bits. This can be confusing. In fact, the usual term for what IBM has done here (although we have not seen it used in their literature) is *graphics indirection*. This means that we tell the computer to determine color from a two-step process. First, it uses the attribute bits to look up the color value. Then it translates the color value to the video signal. The attribute can be thought of as the name of a place where the color information is kept. That place is called the *palette*. Cartridge BASIC provides complete freedom in the choice of palette colors. BASICA supplies two preset palettes.

Graphics indirection adds a great deal of flexibility and power to the computer artist's tool box. Because we can change the meaning of an attribute, we can change the color of a group of pixels all at once, without having to change each and every one of their attribute bits. In fact, you may find that four-color modes using graphics indirection are often more useful than 16-color modes: to change the color of an object in a 16-color mode all pixels in that object must be redrawn. In a four-color mode, only one change needs to be made.

Of course, when you want more than four colors on screen at one time, or you need to change the color in only part of an area of one color, the 16-color modes are a must.

Screen Pages

Screen paging is quite simple. It means that the computer can be displaying from one area of memory while you build another screen in a different area of memory. There are a number of reasons that you might want to do this, but they usually have to do with speed. By switching screens to one that has already been built "off-stage," a program can give excellent response.

A good example would be a hierarchy of menus (also called nested menus). If you have different menus built on different screens, a single SCREEN command will provide almost instantaneous switching among them. A user selects one option and—"click"—there is the next menu. If he changes his mind and wants to return to the main menu, "click" again and it's back.

For character modes, the BASICA SCREEN statement will support up to eight pages in 40-column width, or four pages in 80-column width. In cartridge BASIC, the only limitation on the number of pages in any graphics mode at all is the amount of memory available.

And Finally . . .

When it comes to computer graphics, the limits are set only by your imagination. Try things out: Seeing is believing! And a final hint—if the speed at which your programs run is a problem, consider the BASIC Compiler. While the Compiler does not support all BASICA or cartridge BASIC statements, it will handle most of them. (Unfortunately, at this writing the Compiler will not handle any of the BASIC statements unique to the cartridge.) If you can program around this limitation, you will find that a compiled BASIC program may run from five to 20 times faster than interpreted BASIC.

24

Advanced Sound

The *PCjr* has two means of producing sound. The first method uses the same circuit found in other members of the IBM PC family, thereby providing compatibility with them. This standard sound system is limited to producing one frequency at a time. The sounds generated in this fashion are directed to the small internal speaker inside the computer.

The second method of producing sound makes use of a special Complex Sound Generator chip, new with the *PCjr.* This chip can generate three different frequencies and "white noise" (a hissing sound produced by a combination of many frequencies), all at separate volumes. It sends its output to an external speaker. The speaker can be in the display or a separate amplifier, as in a stereo system. ("Display" means TV, composite monitor, or RGB monitor—although the IBM RGB monitor does not include an audio circuit. See "Advanced Graphics" for descriptions of these devices.)

The advanced sound system of the *PCjr* opens up new possibilities for creativity, by making possible three-part harmony and realistic sound effects. Before delving into the details of the advanced sound system, a few words about sound and music in general might be appropriate.

The text which follows uses the abbreviation "Hz." This stands for Hertz, *the standard term for "cycles per second."*

The Nature of Musical Sound

Musical sound is generally taken to mean the frequencies from the open "E" of the double bass (41 Hz), up to the high B flat of a piccolo (3729 Hz). The IBM sound system can produce pitches extending from just below this range to well above it. These frequencies, however, tend to be of little use. The lower notes are more felt than heard; the higher notes may allow your computer to call your dog, but otherwise are of little value. Furthermore, many sound reproduction systems cannot handle the full range of audible frequencies, which is generally considered to extend from about 30 Hz to around 12,000 Hz. Few people, in fact, will hear tones at the extremes of this range.

The tone generator in the *PCjr* permits the simultaneous production of three separate frequencies, or *voices*, each with its own volume. The stated range of tones is 37 to 32,767. Obviously, most of these tones are neither audible nor reproducible by even the best high-fidelity equipment. (In fact, the BASIC manual states that silence can be produced with a SOUND value of 32,767. Actually, any value above 14,000 or so will have the same result.)

We identify instruments by the "overtones" they produce. It is these ancillary notes that allow us to distinguish, for example, a trumpet's middle-C from a violin's middle-C. The IBM sound generator produces only pure sine-waves; that is, a note with no overtones or "color." Nevertheless, much can be done to produce a pleasing—if somewhat plain—sound. To accomplish this requires at least some understanding of harmony.

Harmony

The most prominent characteristic of music is melody, usually carried by the highest voice. When music has more than one part, or voice, *harmony* exists (whether intentional or not). Western music has a history of ever-increasing harmonic complexity. (Compare Schoenberg to Bach to Palestrina; e.g., tone-row to Baroque to Gregorian chant.) We have become accustomed to expect a certain kind of resolution in response to dissonance.

Dissonance can be defined as the sound produced by two (or more) distinct frequencies which are less than a minor third apart. It tends to push the music on, to reach some more comfortable harmony: consonance. This may be summed up by the following chart:

Interval	**Example** *(C root)*	*Usual Perception*
minor 2nd	C-D flat	Very dissonant
major 2nd	C-D	Dissonant
minor 3rd	C-E flat	Sorrowful
major 3rd	C-E	Resolved, restful
fourth	C-F	Hollow, "open"
augmented 4th	C-G flat	Mysterious
fifth	C-G	Hollow, "open"
major 6th	C-A flat	Resolved, restful
minor 6th	C-A	Sorrowful
minor 7th	C-B flat	Dissonant
major 7th	C-B	Very dissonant
octave	C-C 𝄢	Hollow, "empty"

The chart is meant as a general guideline only. The "usual perception" of the intervals can be debated, especially as they so often depend on their musical context.

Programming Sound in Basic

Cartridge BASIC provides four sound-related statements: BEEP, NOISE, PLAY, and SOUND. Brief descriptions of these statements will be found in the chapter on BASIC. This chapter will concentrate on PLAY and SOUND.

PLAY and SOUND do the same thing, but in different ways. SOUND permits the generation of any frequency, and is required when NOISE source is 3 or 7. PLAY allows only the production of 84 specific notes, starting with C three octaves below middle-C (65 Hz) and ending with B three octaves above (7902 Hz).

Another major distinction between SOUND and PLAY is that PLAY has an "MB" (Music Background) option which puts up to 32 notes into a sound buffer. Notes in the sound buffer are played while the BASIC program continues to execute. The ON PLAY statement can be used to execute a subroutine which refills the sound buffer when a specified number of notes remain in it, thereby permitting continuous background music during program execution.

(Buffered sound need not be limited to music. The sound buffer could be used for a warning buzzer or bell, while the BASIC program proceeds to issue a corresponding message, and to ask the user what action should be taken. This technique will allow the program to be more responsive to its user, who will not have to wait for the sound to stop before entering the next input. Similarly, the sound buffer can be used to hold the sound of an explosion, while a game program animates an appropriate picture.)

The key to using the PLAY or SOUND statements lies in translating musical notation to its computer equivalent.

Translating Music to BASIC

Music is written on a five-line staff, the lines of the treble clef being assigned the notes *EGBDF*, from bottom to top. The bass clef notes are *GBDFA*. Translation to a corresponding PLAY statement value may be derived from the chart below. (The chart omits the lower- and upper-most octaves).

```
+-----------------------------------------+-----+-------+----------+
|                Musical                  |Note | Note  |Frequency |
|                Notation                 |Name |Number |  (Hz)    |
+-----------------------------------------+-----+-------+----------+
|G-clef (Treble): 𝄞                   -O- |  C  |  61   |  2093    |
|                                  -O---- |  A  |  58   |  1760    |
|                       ----------O------ |  F  |  54   |  1396.9  |
|                       --------O-------- |  D  |  51   |  1174.7  |
|                       -----O----------- |  B  |  48   |   987.8  |
|                       ---O------------- |  G  |  44   |   784    |
|                       -O--------------- |  E  |  41   |   659.3  |
|F-clef: 𝄢          -O- (middle-C)       |  C  |  37   |   523.3  |
| ------------O--                         |  A  |  34   |   440    |
| ----------O----                         |  F  |  30   |   349.2  |
| --------O------                         |  D  |  27   |   293.7  |
| ------O--------                         |  B  |  24   |   246.9  |
| ----O----------                         |  G  |  20   |   196    |
| ---O-                                   |  E  |  17   |   164.8  |
| -O-                                     |  C  |  13   |   130.8  |
+-----------------------------------------+-----+-------+----------+
```

For the sake of clarity, the chart does not show all notes of the chromatic scale. These are:

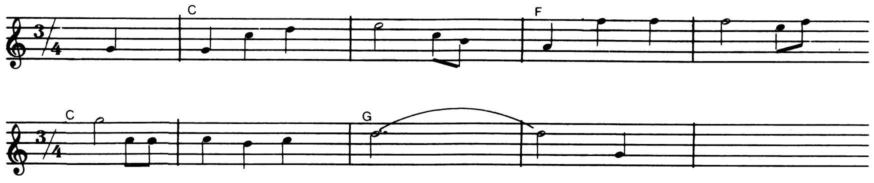
```
C C#/Db D D#/Eb E F F#/Gb G G#/Ab A A#/Bb B C
```

Thus, middle C (note 37) is followed by note 38 (C sharp or D flat: the tones are identical in a tempered scale, such as that found on a piano or in the PLAY statement).

As an example, let's try "Home on the Range." First the sheet music is translated to note numbers. Then, the value of the string called MELODY$ is set by the addition of note-length values. (Please refer to the PLAY statement in the BASIC manual for information about the parameters being used here.) Blanks are inserted in the string to indicate where each measure begins. This is done to make locating a given measure easier: blanks are ignored by the PLAT statement.

The string called BASS$ is created by starting from the notes of the chords. A bass line is usually more pleasing to the ear if it has some melodic qualities, so notes have been added which introduce not only a "counter-melody," but a degree of rhythm and harmonic tension as well. It is these finer points of creating musical sound which provide artistic rewards in return for the additional programming time they require.

The tune:



The program:

```
 10 SOUND ON
 20 MELODY$="t160 14 n44 n44n49n51 12n5318n49n48
      14n46n54n54 12n5418n53n54 12n561
      8n49n49 14n49n48n49 11n51 14 n44"
30 BASS$="t160 14 n0 12n3714n36 n37n36n32 12n3014n41
      n39n37n36 n32n30n29 n29n30n
31 n32n39n36 n32n34n36"
40 PLAY MELODY$,BASS$
```

25

The Future of PCjr

As Casey Stengel is supposed to have said, "Making predictions is difficult and especially when it involves the future." But we'll be brave, and describe what you might expect in the near future for your *PCjr.*

PCjr has expansion potential that could rival its big brother, the IBM PC. Beginning with memory (though IBM has stated that it will only support some 128K of Random Access Memory), *PCjr* has the capability of handling substantially more, up to 1,000K; that is, 1,000,000 bytes, or a megabyte, just like the IBM PC.

Because of *PCjr's* use of the same microprocessor (the micro chip that is actually the computer), the Intel 8088, the system does have the capacity for memory expanded beyond the current 128K. But who will provide it? There are virtually scores of Original Equipment Manufacturers (OEM's) out there building such expansion devices (usually boards that fit inside the computer) for the IBM PC. It's a pretty good bet that these OEM's will be announcing similar expansions for the *PCjr.*

Expanded memory could come in a variety of forms. The first might be an expansion board fitting into a slot within *PCjr.* This option would probably fit into the slot designated for the internal modem, which would be fine, as you can always buy an external communications device. A second form of memory expansion would be a replacement board for one already in the *PCjr.* Such devices are currently on the market for the IBM PC. And finally, you can be sure that some smart manufacturer is going to come up with an expansion chassis: a box that looks just like *PCjr*, configured to hold not only expansion boards for memory and other functions, but a larger power supply. We'll get to increased power supply in a moment.

How will more memory be advantageous to the *PCjr* user? At the very least, it will allow access to a greater variety of software. However, software requiring more than 128K often requires more than one disk drive. As you know, more than one disk drive requires a greater power supply. Soon after IBM announced the *PCjr*, a major OEM (who currently provides a broad range of accessory expansion devices for the IBM PC) announced that it would have a disk-drive expansion device ready for the new machine in early 1984. Now, that does broaden your *PCjr* horizons. With two drives and 128k, the *PCjr* user will be able to use current applications software more efficiently and substantially more software that is on the market for the PC. Two drives mean that you will not have to swap diskettes in order to save data. Your program diskette will run in drive A, while the data diskette will be in drive B. Of course, there is no room in the *PCjr* for a second drive because of the physical constraints, so your second disk drive will sit on top of or beside the unit. But with drives, computing life will be much easier.

Some pioneering manufacturer is probably already dreaming up a fixed, or hard disk, for the *PCjr*. A *fixed disk* is a diskette that is sealed (not removable). They generally hold at least five megabytes (5,000K) of data, as opposed to the 360K held on a floppy disk. On a five-megabyte fixed disk, you would be able to hold the equivalent of nearly 14 completely filled floppy disks. That's a lot of data. For the IBM PC, hard disks now range from five megabytes up to 200, with the most popular being the 10-meg system. Incidentally, there are also removable hard or fixed disks, meaning that you can purchase five megabyte cartridges (sealed, of course) and feed them to the hard disk chassis as you would a tape cartridge. Could these, too, be in the future? Fixed disks are eraseable, just like the floppies—you may reuse them.

With regard to storage, you might look toward a 2.5 megabyte floppy disk drive for the *PCjr*—there's one available for the IBM PC.

Those are just for starters. With increased memory and storage capacity a whole world of software becomes available to the *PCjr* user. There are sophisticated data-base management systems used to run companies large and small. These DBMS programs, as they are called, are used to track inventory, personnel, and almost any other data imaginable. While *Home Budget Jr.* is currently available for the *PCjr;* greater capacities would allow use of full-blown accounting systems. These include modules for *Accounts Payable, Accounts Receivable, Billing, Inventory, Checkwriting,* and *General Ledger.*

The attachment of a letter-quality printer to your *PCjr* and the use of a full-service word processor is something to be considered in the future.

Appendix 1

PCjr and the IBM PC

A Technical Comparison

The *PCjr,* although based on the same 8088 microprocessor as the IBM PC and PC XT computers, has significant internal differences.

Programs can be written so as to be fully compatible across the current PC line of IBM computers as long as these differences are understood and either avoided or allowed for.

As an alternative to avoiding program differences, logic can be included to determine which machine a program is running on, so that noncompatible branches of code can be executed. Machine type can be determined by inspecting ROM location FOOO:FFFE (segment F000, offset FFFE), which contains the following hexadecimal values for the machines indicated:

FF—IBM PC
FE—PC XT
FD—PCjr

The following sections will discuss some of the known internal differences between the PCjr, PC, and PC XT computers.

BIOS

The Basic Input/Output System (BIOS) in ROM of the *PCjr* is not the same code found in other IBM Personal Computers. However, the documented interrupt vectors have been maintained. The only problems that may arise would be with those programs which use the BIOS code directly, thus violating the warnings issued by IBM from the outset.

Program Timing

Programs running in Random Access Memory (RAM) may run somewhat slower on the *PCjr* than they would when run on other members of the IBM PC family. On the other hand, cartridge-based (ROM) programs will usually run faster than their RAM counterparts. This means that programs which rely on internal timings (e.g., 8088 instruction speeds) may run differently on the two classes of machines. IBM has always recommended using the system timer (as opposed to instruction timings) where time-critical functions are needed. In BASIC, this would mean using the TIMER function as opposed to a FOR-NEXT loop.

Memory Usage

Unlike the other IBM Personal Computers, display memory in the *PCjr* is taken from main memory. The other PC's require a separate "adapter" board for either monochrome or color/graphics displays.

These adapter boards include memory specifically designated for use by the displays. The elimination of this scheme in the *PCjr* means that less memory is available for program and data storage. The amount of memory "lost" to the display functions can range from a low of 2K for the 40-character display mode (SCREEN 0, in BASIC), up to 32K for the higher graphics modes.

Diskette Format and Access

The diskette format used by the *PCjr* is determined by DOS 2.1, and is normally 40 tracks of nine 512K-byte sectors each, for a total of

360KB per diskette. This format is fully compatible with the usual diskette format produced by DOS 2.0. In addition, diskettes produced with different formats, such as the eight sectored tracks of DOS 1.1, or single-sided diskettes, can be read by DOS 2.1.

However, the *PCjr* does not use Direct Memory Access (DMA) when performing diskette i/o. DMA refers to the ability of the peripheral to momentarily halt CPU operation, so that the peripheral can directly access memory shared with the CPU. Lacking this ability, the *PCjr* CPU must wait during the entire i/o operation. This has implications in program timing. There is also a potential for loss of data if a program attempts "overlapped" i/o operations. (Say, for example, performing i/o to both diskette and keyboard.)

Keyboard

There are obvious physical differences between the keyboard of the *PCjr* and the full 83-key layout of the other IBM PC's. However, the *PCjr* keyboard generates the same "scan codes" as its big brothers, and therefore should be compatible for most applications. For those using the BIOS interrupts, IBM recommends that interrupt hex 16 (Read Keystroke) be used to read the keyboard.

Advanced Graphics and Sound

All graphics modes shared between the *PCjr* and the other PC's are fully compatible. The new graphics modes, however, cannot be reproduced by the color/graphics adapter of the PC and PC XT.

Similarly, the multiple-voice sound capabilities of the *PCjr* rely on a chip not present in the PC and PC XT, and so cannot be duplicated on those machines.

Appendix 2

Binary and Hexadecimal Systems

The real trick with computers today is learning what software to buy, and how to make it work best for you. The days when a computer user had to know binary arithmetic are behind us. This Appendix, then, is for the curious, and is not required reading.

Computer architecture is based on the bit, the fundamental unit of memory. The bit can be in one of two states: either a one (or on, or true, or yes, etc.) or a zero (off, false, no). Thus, the binary system uses only two characters to represent numbers: *1* and *0*.

Like the decimal system, the binary system of notation relies on the convention that a digit's position in a number corresponds to the base of the system raised to a power. That is to say, in a decimal number the one's position holds the number of units, the 10's position the number of 10's, and so on. Thus:

1324 equals 4 times 10 to the 0 power (which is 1),
plus 2 times 10 to the first power (10),
plus 3 times 10 to the second power (100),
plus 1 times 10 to the third power (1000),

Or: $(4 \times 1) + (2 \times 10) + (3 \times 100) + (1 \times 1000)$. The binary system has a base of 2. Thus, in binary:

1101 equals 1 times 2 to the 0 power (which is 1),
plus 0 times 2 to the first power (2),
plus 1 times 2 to the second power (4),
plus 1 times 2 to the third power (8),

Or: (1×1) + (0×2) + (1×4) + (1×8). In decimal, the binary value 1101 is represented as 13.

Doing arithmetic in binary involves the same principles and processes as doing arithmetic in our familiar decimal system. In binary, one plus one equals zero with a one carried to the left. Thus:

```
 1101     1101     1101     1101
+0001    +0010    -0001    -0010
-----    -----    -----    -----
 1110     1111     1100     1011
```

In the last subtraction above: 1101 minus 0010 is decimal 13−2. Is the binary answer of 1011 also 11?

Eight bits make up a byte. A byte can hold the values 00000000 through 11111111. In decimal, this is 0 through 255. Since writing binary numbers takes so much space and is awkward to read, another system of notation was developed. This system is called *hexadecimal.* The term hexadecimal, meaning "16," comes from the fact that 16 characters are used to represent the possible values in four bits: 0 through 15. (Four bits is half a byte, not 50 cents, and is sometimes referred to as a "nibble.") By convention, hexadecimal notation uses the numbers 0 through 9 as you would expect, and then continues:

Hex	=	*Dec*	=	*Binary*	*Hex*	=	*Dec*	=	*Binary*
A		10		1010	D		13		1101
B		11		1011	E		14		1110
C		12		1100	F		15		1111

Thus, a byte of all ones (11111111, or decimal 255) is represented as *FF.* To distinguish hex notation from decimal, hex numbers may be preceded by *&H* (used in BASIC: e.g., *&HFF*), or followed by *H* (used in Assembly Language: e.g., *FFh*).

Hexadecimal arithmetic may seem strange at first. For example, a "carry" in hex is 16:

Hex =	*Dec*	*Hex* =	*Dec*	*Hex* =	*Dec*	*Hex* =	*Dec*
14	20	14	20	14	20	14	20
+1B	+27	+1D	+29	− 2	− 2	−C	−12
2F	47	31	49	12	18	8	8

Here are some more examples (try to puzzle them through):

2134	2134	2134	2134
+0008	+C01F	−0008	−0CF1
213C	E153	212C	1443

The chart below is used to convert between decimal and hexadecimal.

Example 1: Hex to decimal. This conversion is done by straightforward table look-up. Convert *&HB3E4* . . .

B000	=	45,056	plus . . .
300	=	768	plus . . .
E0	=	224	plus . . .
4	=	4	
B3E4	=	46,052	

HEXADECIMAL COLUMNS							
4		3		2		1	
Hex =	Dec	Hex =	Dec	Hex =	Dec	Hex =	Dec
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,584	E	224	E	14
F	61,440	F	3,840	F	240	F	15
BYTE				BYTE			

Example 2: Decimal to hex. Converting in this direction involves inspecting the table for the largest value equal to or less than the decimal value remaining, and subtracting the table value to get the next decimal value. To illustrate, convert 2134 . . .

```
2134 decimal . . .   contains 2048 = &H0800
2134 - 2048 = 86,    contains   80 = &H0050
  86 - 80   =  6                   = &H0006
```

Hence, 2134 = *&H0856*. Double check this, if you wish, by converting *&H0856* back to decimal. Also try converting 46,052 to hex.

Appendix 3

Computer Communications

The topic of computer-to-computer communications (data transmission) could easily fill a volume of its own. The practical aspects of using the *PCjr* for communications are discussed in the body of this book. The *PCjr* was announced as supporting only "asynchronous communications." But asynchronous communication ("async") is not the only means of transmitting data.

The other IBM personal computers have options which support the two other major communication schemes in current use: Binary Synchronous ("bisync" or BSC) and SDLC (for Synchronous Data Link Control).

This appendix will define these three methods of data transmission, and discuss some of their pros and cons.

Some Fundamentals

Data communications depends on the sender and receiver agreeing on the interpretation of certain bit patterns, or "codes," to mean certain things; for example, the letters of the alphabet, "start a new line," or "I'm finished sending." Two codes are in common use today: ASCII for most small computers, and EBCDIC for large mainframe computers.

ASCII (for American Standard Code for Information Interchange) is normally a seven-bit code. EBCDIC (Extended Binary

Coded Decimal Interchange Code) is an eight-bit code. We will concentrate on the ASCII code. In addition to the code being used, the communicating computers must agree on what error checking they will perform. Error checking is desirable because of the nature of telephone circuits.

Voice communications can suffer from a bad connection. Data transmission is far more sensitive than voice. Therefore, even minor telephone problems that would not impair a voice call can alter data bits. (These problems can arise from a number of sources. A more detailed discussion is beyond the scope of this appendix.)

Error checking can take many forms. The most primitive of these is called *parity.* Parity refers to the number of bits which are "on" in a character. When parity is used, a bit must be added to the ASCII character code. This extra bit can then be set to either a zero or a one, as required to make the total number of one bits come out to be either even ("even parity") or odd ("odd parity"). Once the computers have agreed on parity, each can detect a so-called "one-bit" error. This means an error when only one bit of a character is changed, resulting in the wrong parity.

Most line errors, however, will affect many bits of data in a random fashion, so the usefulness of parity checking is limited. For this reason, many communication programs allow you to set parity off, meaning, in effect, "Why bother?"

The transmission codes (including special-purpose codes), the type of synchronization, and the error-detecting schemes used combine to form a *communications protocol.*

Asynchronous

Asynchronous data transmission goes back to the days when information was sent by teletypewriter (TTY). In async, each character has bits added to its ASCII code to provide synchronization between sender and receiver. These bits say, in effect, "Watch out! Here comes the start of a character," and "Okay, that stops that character." Because of this, async is also referred to as "start stop." The result of adding start and stop bits is that it takes a minimum of 10 bits to transmit an eight-bit character code (including parity). These extra two bits translate to a 25 percent increase in the amount of time it takes to send data.

Bisynchronous

As the technology improved, more accurate internal clocks were eveloped for communications devices. This means that once syn-hronized, two communicating stations can send a large number of haracters before the danger arises that they may drift out of syn-hronization. The binary synchronous communication protocol takes dvantage of the better hardware. Messages are sent in blocks of a redetermined size (typically 256 or 512 bytes).

Each block begins with at least two synchronization characters nd a start-of-text (STX) character. The block ends with an end-of-ext (ETX) character followed by one or more "block check charac-ers." These last bytes contain a value based on the one-bits in the ext being sent. The value is calculated by the sending station and dded to the block. The same calculation is performed by the receiv-ng station as the text is received. The calculated value is then ompared to the transmitted value. If they match, the receiving tation sends an ACK, to acknowledge correct transmission. If the alues do not agree, the receiver sends a negative acknowledgment NAK), to get the sender to retransmit the block.

The definition of the bisync protocol includes the error detection ust described. This is in contrast to async, which only offers parity or error detection, and as an option at that.

But the real advantage of using bisync is the speed at which data nay be sent. The upper limit of async communications at this writing s 1200 bps (bits per second). As we have described, this translates to 20 cps (characters per second). Bisync, on the other hand, can ransmit at speeds up to 9600 bps over ordinary telephone circuits. Because only eight bits are sent per character, this means an effective ransmission rate of just under 1,200 characters per second. To do his, however, involves more expensive equipment. This accounts for he continued popularity of async among those users who do not need o transmit large amounts of guaranteed-correct data.

Asynchronous: XMODEM

In an effort to improve the reliability of asynchronous data trans-nissions, the XMODEM protocol was developed by a communica-ions engineer named Ward Christensen. XMODEM uses a block-transmission scheme very much like the one described above for binary synchronous communications. Blocks of 128 bytes are preceded and followed by control information which includes the block number and a block check character.

The use of XMODEM will help ensure virtually error-free communications, at a relatively small increase in transmission time over the standard async protocol. However XMODEM, while popular, is by no means universally used. Moreover, since XMODEM is not formally recognized as a standard protocol, the danger exists that an "XMODEM" protocol implemented by one communications program will not operate properly with the "XMODEM" protocol of a different program.

SDLC

SDLC (Synchronous Data Link Control) was developed by IBM as part of its Systems Network Architecture (SNA). SDLC is not unlike bisync, in that the basic transmission "unit" is a block of data, not a single character. SDLC extends this scheme to allow one station to send a number of blocks before demanding acknowledgments from the receiver. The protocol is not limited to a transmission blo of any particular size. It is designed to take advantage of satelli based communications, which can sometimes cause problems f bisync. SDLC is the "protocol of choice" for future IBM mainfram products. Where its future lies in personal communications is far less clear, as it requires sophisticated (read "expensive") hardware and software.

$14.95

STEVE STERN GREG YOUNG

PC*jr* PRIMER

A GUIDE TO THE IBM® PC*jr*

The IBM® PCjr is here—and, fortunately, so is the **PCjr Primer.** This book will teach you everything about the PCjr—its components, how to set it up, how to run it, and what makes it different from the IBM® PC. You won't have to rely on the complicated user's manual provided with the PCjr; instead, Steve Stern and Greg Young have simplified all the important concepts for you.

All you need to know to start programming your IBM PCjr is covered in **PCjr Primer.** For once you won't have to wait three months *after* you buy something for the appearance of a coherent, clear guidebook. **PCjr Primer** discusses printers and other peripherals, how the detached infra-red keyboard works, how and why the PCjr-DOS™ operates, and how to talk to other computers through the modem. After you thoroughly understand what makes the PCjr run, **PCjr Primer** explains three major pieces of software that you can use with your PCjr—**Homeword, Home Budget, Jr.**, and **Multiplan**™.

Start reading this exciting book today. **PCjr Primer**'s entertaining style will put you on the right track for getting the most out of your IBM PCjr.

Cover photo by Jim Kirby

Cover design by Nancy Sutherland

A Reston Computer Group Book
RESTON PUBLISHING COMPANY, INC.
A Prentice-Hall Company
Reston, Virginia

Special thanks to Math Box

0 7
21898 54099

0-8359-5409-9