OSBORNE 1 User's Reference Guide

OSBORNE™
COMPUTER CORPORATION

by Thom Hogan and Mike Iannamico

OSBORNE 1™

User's Reference Guide

by Thom Hogan
and Mike Iannamico

Acknowledgements

Particular acknowledgement should be awarded the following people for their participation in this newly revised and typeset User's Reference Guide: to Roger Gottlieb for directing and overseeing the project; to Gary Cuevas for his expertise in BASIC and program contributions. Also, a special thanks to all of those Osborne users who took the time to point out errors and inconsistencies in previous editions and who made valuable suggestions for improvement.

Part #: 2FØØ161-Ø1

Osborne Computer Corporation reserves the right to make changes or improvements to the equipment, software, and documentation herein described at any time and without notice.

Notice: Please be advised that any tampering with the internal hardware of the Osborne 1 computer may void the warranty.

The Osborne 1 computer falls under the rules regarding radiation and radio frequency emission by Class A computing devices. The following information must be supplied to users of the Osborne 1 in accordance with the FCC standard Part 15, Subpart J:

> This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class A computing device in accordance with the specifications in Subpart J or Part 15 of FCC Rules, which are designed, however, to provide reasonable protection against such interference in a residential installation. There is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment on and off, the user is encouraged to try to correct the interference by one or more of the following measures:
>
> - Reorient the receiving antenna.
> - Relocate the computer with respect to the receiver.
> - Move the computer away from the receiver.
> - Plug the computer into a different outlet so that the computer and receiver are on different branch circuits.
>
> If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful: "How to Identify and Resolve Radio-TV Interference Problems."
>
> The booklet is available from the U.S. Government Printing Office, Washington, D.C. 20402, Stock No. 004-000-00345-4.

Table of Contents

PART 1

Introduction

This manual is unlike most others in the microcomputer world.

1. This manual was written by users of the Osborne 1, not technicians.
2. The tutorial portion of this manual was written several months after the computer appeared.
3. This manual emphasizes teaching of how to make the computer perform tasks, and does not stress the rote memorization of material.
4. This manual was written, in part, using the equipment and software it describes.

The tutorial begins with chapter 1, an exploration of the physical attributes of the Osborne 1; including a section on setting it up and turning it on, and an explanation of what diskettes are. The tutorial assumes that you have no knowledge of computing and makes sure you acquire the fundamental information you need to use your new computer effectively.

Each succeeding chapter in the tutorial covers progressively more specialized information and tasks. Chapter 2 tells you how to start the system and copy diskettes; you'll also find a few other things you'll need to know in order to understand later chapters. Chapter 3 tells you all you need to know about the CP/M operating system in order to efficiently use the Osborne 1.

Chapters 4 and 5 deal with WordStar and SuperCalc, the two primary programs that come with the Osborne 1. Both of these chapters take you from no knowledge of the tasks involved to a full understanding of how to use WordStar and SuperCalc.

Chapters 6 and 7 examine CBASIC and Microsoft BASIC, and cover the procedures for using prewritten programs in those computer languages. Chapter 7 specifically instructs you on how to begin learning to program the computer.

Finally, chapters 8 and 9 explore complex and advanced topics concerning CP/M and the internal design of the Osborne 1.

The material in the tutorial section is progressively more specialized; thus, many newcomers to computing will find that by the end of chapter 5 they know everything they need to know to use their Osborne 1 computer. With the exception of the first three chapters, feel free to skip around and to stop once you think you've grasped all the information you need to use your computer. Someday you may find that the rest of the tutorial is relevant because of new uses you discover for your Osborne 1.

The tutorial is designed to be used in conjunction with the Reference Guide. It is the Reference Guide, located after the tutorial, that you'll use most often once you become familiar with how your Osborne 1 works. The Reference Guide defines all the features of CP/M, WordStar, SuperCalc, and BASIC, and explains how to use them.

We are proud of this manual, but considering the very nature of documentation, some mistakes have undoubtedly gotten by. Also, since the computer and software that accompany it are constantly being refined, so must the documentation be revised to reflect these improvements. If you find something in this manual that isn't complete, accurate, or easy to understand, please let us know so that we can make whatever changes are necessary in future revisions.

Mike Iannamico & Thom Hogan

OSBORNE 1™

User's Guide

CHAPTER 1

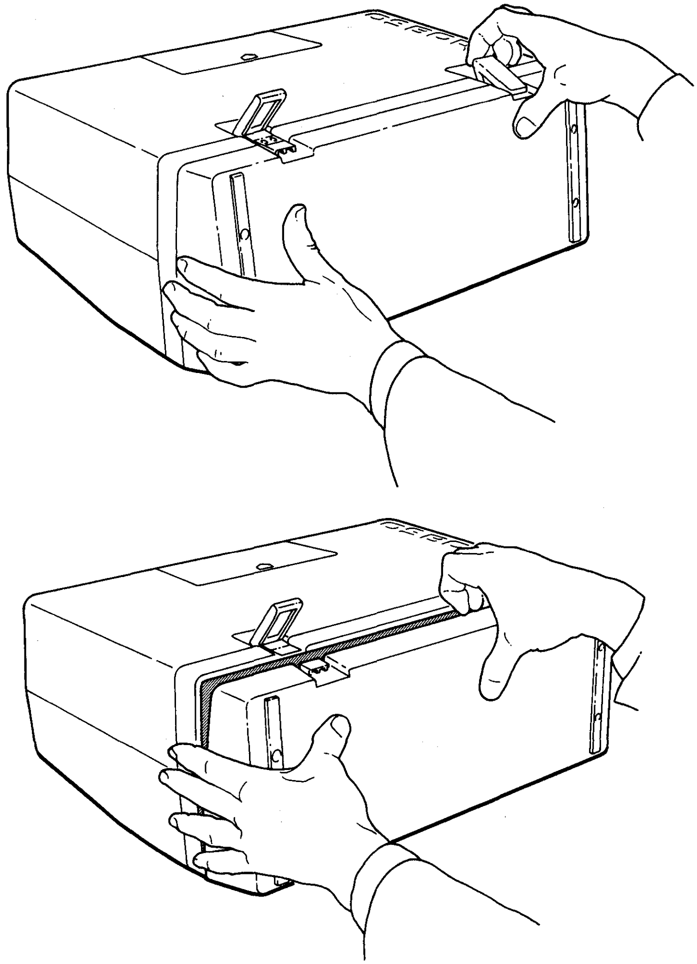
Getting To Know Your Computer

This section describes the various parts of the Osborne 1 computer. You'll also learn how to ready the machine for use.

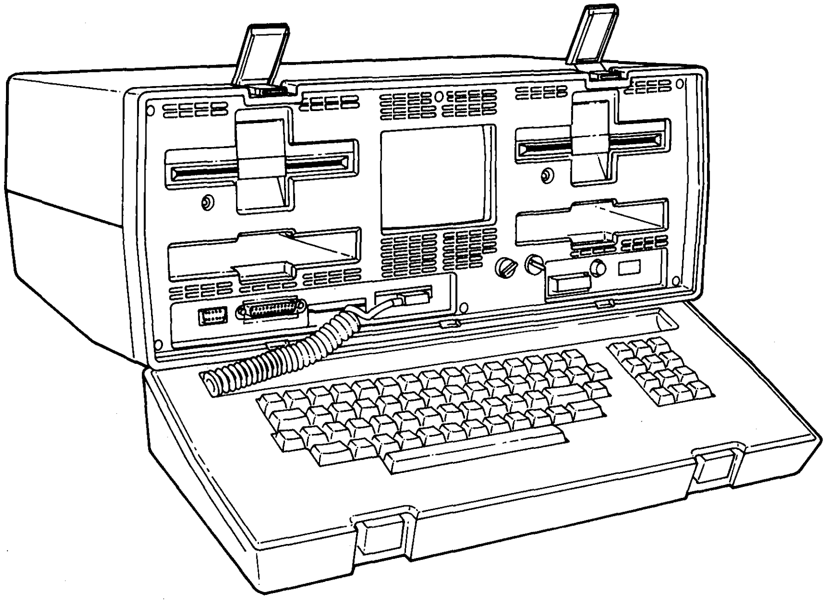
Setting Up the Osborne 1 Computer

The Osborne 1 is a sophisticated business machine. To get started, you simply place it on a flat surface, with the latches on top. The handle should be at the rear facing away from you.

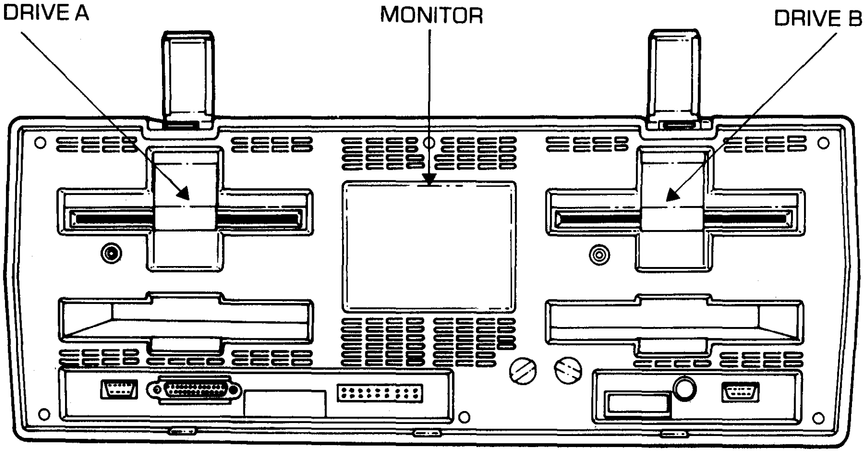
The keyboard is on the inside of the cover, which you remove by unfastening the latches. A cable connects the keyboard to the main body of the Osborne 1. Be careful that you don't violently jerk the keyboard away from the computer, doing so may damage the cable.



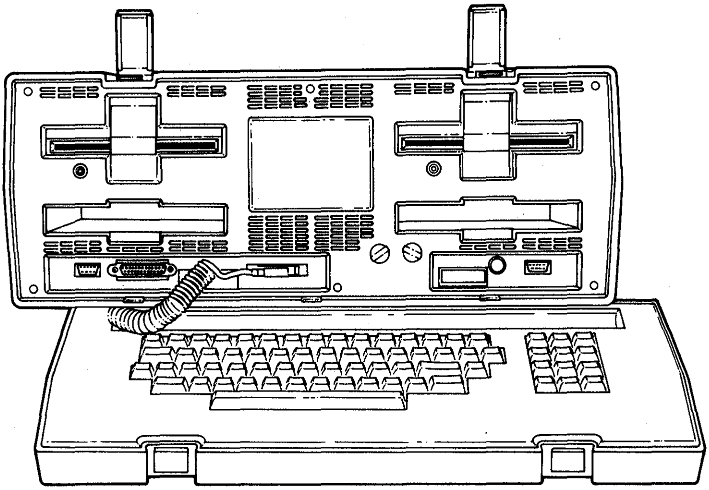
The keyboard has a standard typewriter layout, with a ten-key numeric pad on the right side:



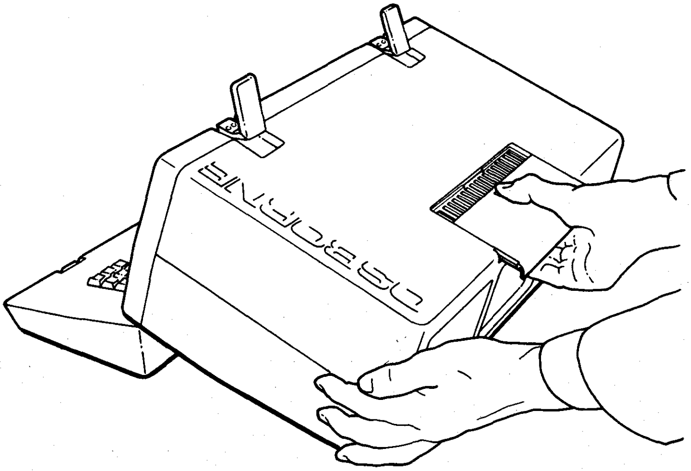
When you look at the Osborne 1, the most prominent features facing you are the screen (monitor) and the disk drives:



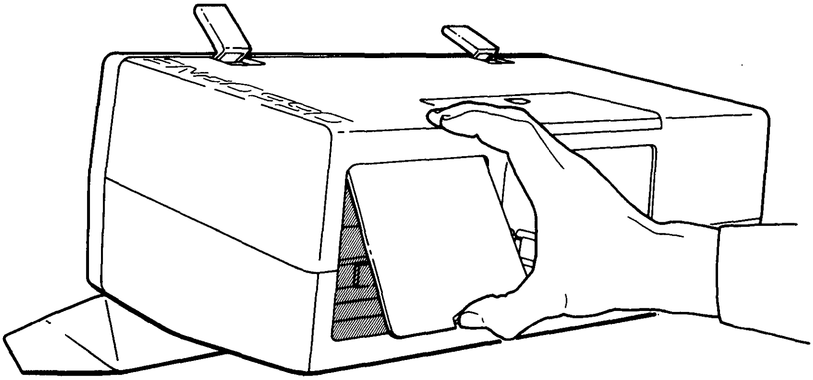
Orient the display so that it is comfortable. Some people like to set the bottom rim of the main computer unit on the back edge of the keyboard. If you try this, you'll find that the monitor tilts up toward you at an almost perfect angle.



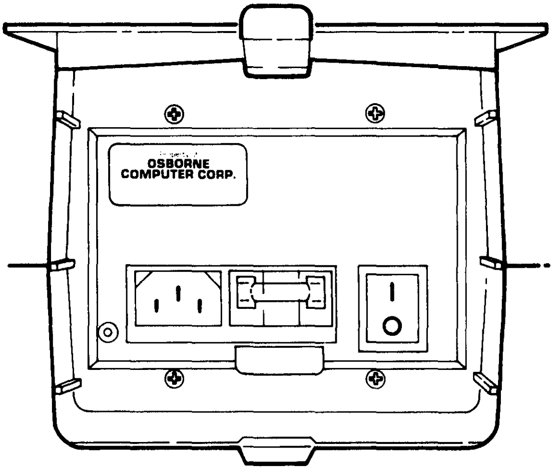
There is a vent on the Osborne 1 used to dissipate heat when the computer is in use. The vent, located on the top side (when the computer is set up for use), should always be uncovered. To open the vent, press the finger-sized, corrugated depression and slide the cover in the direction of the handle.



The power cord, power switch, fuse card, and circuit breaker fuse are all located in the covered power well, next to the handle. The hatch covering the power well opens very easily, just pull the tab and lift upward.



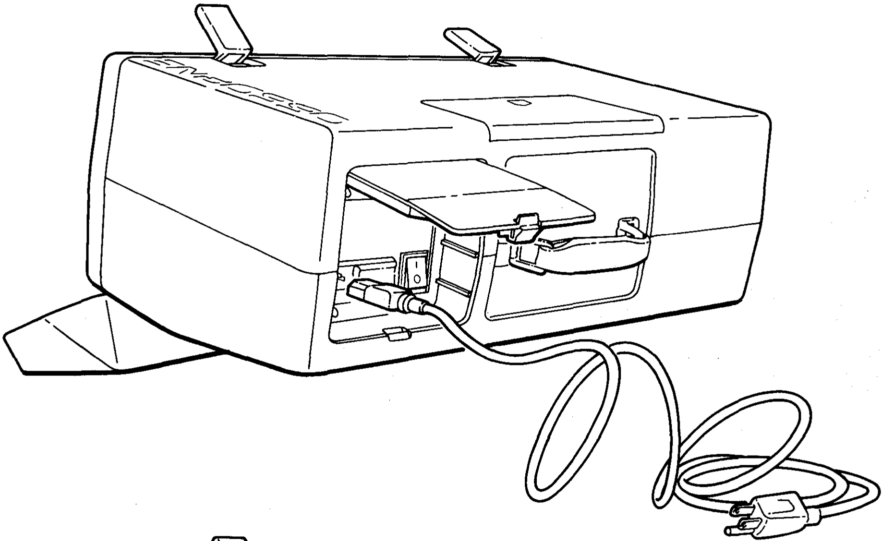
Notice that the fuse and fuse card are accessible by sliding the clear plastic fuse cover over the three-pronged plug. A one-amp fuse protects the internal components from damage caused by power overloads. To replace the fuse, slide the fuse cover away from the fuse, then pull upward on the "fuse pull" tab.

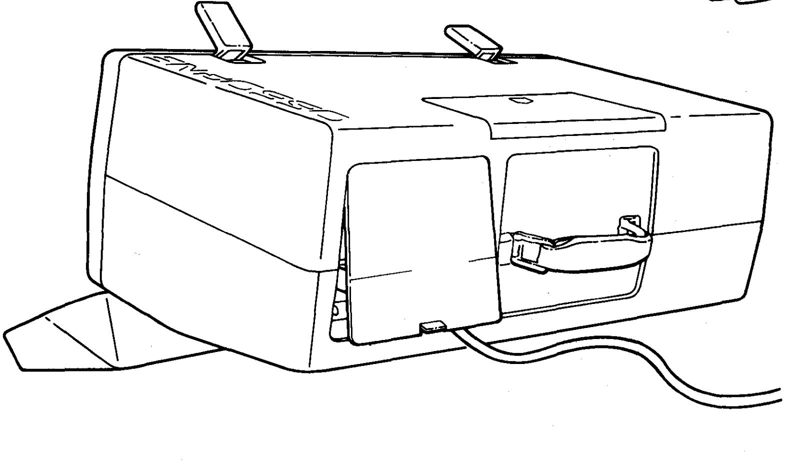


WARNING: Consult your authorized dealer on switching voltages.

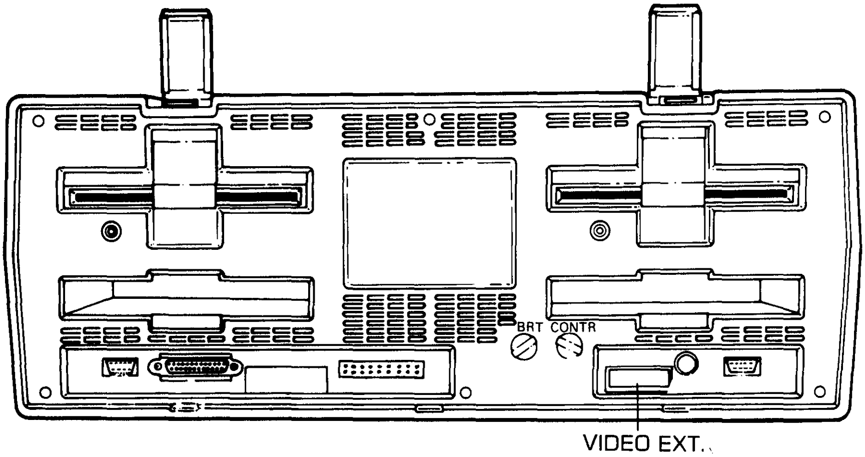
Now that we have detailed the contents of the power well, it is time to plug the computer in. Connect the socket end of the power cord onto the three-pronged plug located in the power well.

It may be necessary to slide the plastic fuse cover away to get at the plug. When the cord is securely attached to the Osborne 1, plug the other end into a wall socket. Now, press the top portion of the power switch to turn the computer ON. After you are through accessing the power well, lower the hatch to protect it from being damaged.

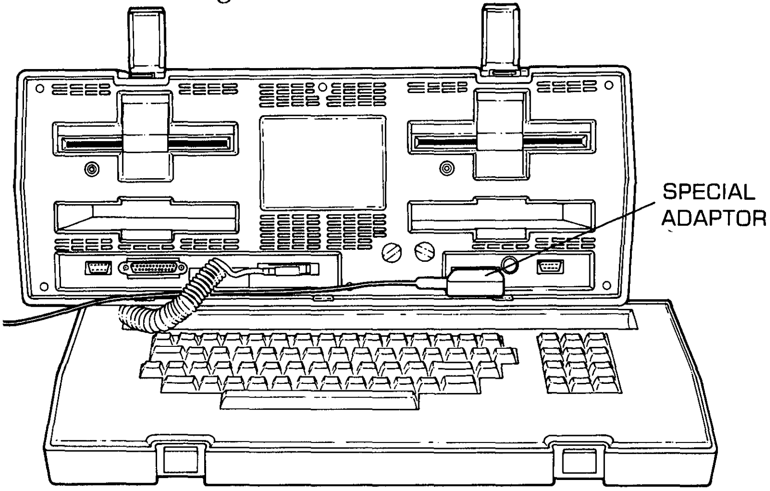




When you power up the computer, some information should appear on the screen (referred to as the "monitor"). We will discuss the information displayed on the monitor in chapter 2. If after a few moments nothing appears, check the small opening below the monitor and slightly to the right, labeled **EXT VIDEO**. A rectangular-shaped plug should securely fit this slot. If this plug is loose or disconnected, TURN OFF THE POWER TO THE COMPUTER and then reconnect it.



The plug that comes with your Osborne 1 selects the 5-inch built-in monitor as the display location. If you like, a special adaptor may be purchased which allows an external monitor to be connected through this slot.



Sometimes rough shipping of the computer may result in the brightness and contrast controls being slightly out of adjustment. If you're still not getting a display, try "twiddling" with the knobs labeled **BRT** (for brightness) and **CONTR** (for contrast). These controls work very much like those on your television set. The proper method for setting the brightness and contrast levels is as follows:

1. Turn the contrast all the way up (clockwise).
2. Adjust the brightness level to wherever suits you (if you see diagonal white lines on the display, you probably have the brightness adjusted too high).
3. Now adjust the contrast level so that each character on the display is clearly defined.

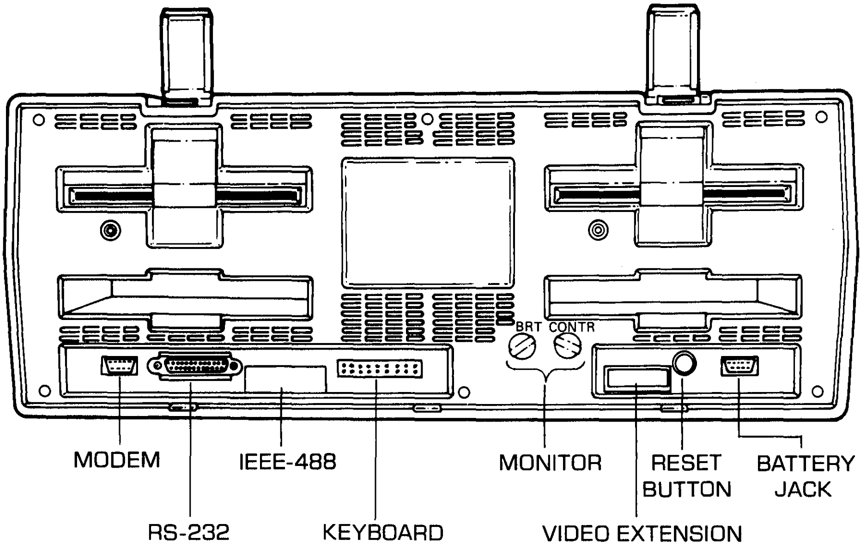
If you still do not have a display after making the above checks and adjustments, be sure you have plugged the computer into a working wall socket.

If you have performed all the steps above, but the monitor still displays nothing, then call the dealer who sold you the Osborne 1.

Plugging Things Into Your Computer

If you remove the plug in the **EXT VIDEO** slot (do so only with the power to the computer OFF) and look inside, you'll see a "connector." The connector is comprised of a flat piece of protruding plastic with little metal "spikes" on it. There are several connectors which your Osborne 1 uses to hook up to other "devices."

At the far left of the Osborne 1 is a **MODEM** connector. This connector is employed for telephone communications. With an optional Osborne-supplied modem, you can use your Osborne 1 to transmit and receive information over standard telephone lines.



Immediately to the right of the modem connector is one labeled **SERIAL RS232**. RS-232 refers to a computer-industry standardized method of connecting devices to a computer. Your Osborne 1 is compatible with many other devices through this port. Most printers use the RS-232 method of "communicating" with the computer, for instance.

If you're not sure whether or not your printer uses the RS-232 standard, ask the dealer from whom you purchased the printer to help you make it work with the Osborne 1.

Accessories advertised in computer magazines are often described as being "RS-232 compatible." You may or may not be able to attach these accessories to your Osborne 1 through the RS-232 connector, however. Some such devices require a specially constructed cable because their manufacturer did not conform to the complete RS-232 standard. If you want to use an accessory, buy it from your dealer, or at least consult with your dealer before purchasing it elsewhere.

To the right of the RS-232 connector, the next connector you'll find is marked **IEEE 488**. This is another well-known method by which computers and accessories communicate. If you have any engineering or scientific equipment, or a printer labeled as being "Centronics Interface compatible," chances are that it plugs into the IEEE-488 connector. Again, your dealer can help you connect accessories to this interface.

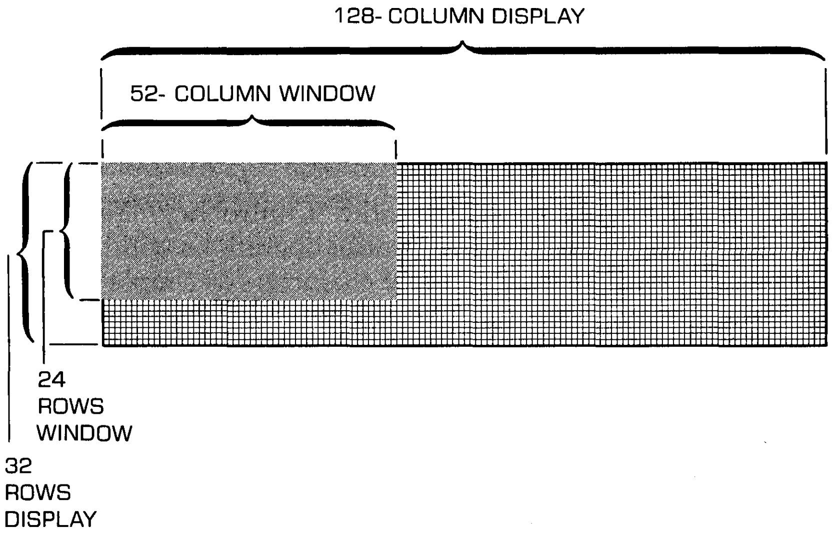
The next connector to the right, labeled **KEYBOARD**, should already have the cable that comes out the back of the keyboard plugged into it. Should the keyboard ever come unplugged, plug it back into this connector, WITH THE POWER TO THE COMPUTER OFF.

Next comes the **EXT VIDEO** connector, which we've already discussed. As stated, if you are using another monitor, you must remove the small plug from this connector and make use of a special adaptor in its place. Don't lose the original plug, however, you'll need it whenever you wish to use the internal video display. Remember that you must make the switch with the power to the computer OFF.

Last,on the far right of the front of the computer you'll find a connector labeled **BATTERY**. This connector was originally designed for the optional battery pack but is now reserved for future use.

The Monitor

The Osborne 1 displays 24 lines at a time, with 52 characters on each line. The screen shows only a small portion—usually referred to as a "window"—of a larger, unseen screen. The Osborne 1 display is actually comprised of 32 lines of 128 characters each, even though only a portion of it is visible at a time. You can see the additional area by first holding down the key labeled [CTRL] and then pressing one of the arrow keys. When you do so, the display will "slide" text under the screen, in the direction of the arrow key being pressed. In other words, you use these controls to see any 24-line-by-52-character portion of the larger display:



The monitor built into the Osborne 1 has a 5-inch diagonal screen. Optional external monitors are larger, usually 12 inches diagonally. Since an optional monitor displays the same number of characters, the characters will appear bigger.

The Keyboard

The keyboard supplied with the Osborne 1 has the standard set of letter and number keys, positioned as you would find them on a typewriter. A numeric pad sits to the right of the keyboard to facilitate the entry of numbers.

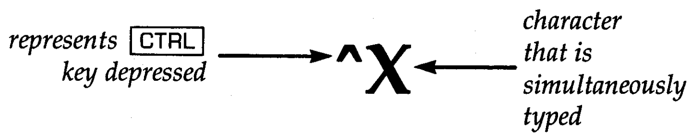
The Osborne 1 keyboard operates much like a typewriter keyboard. When you press a key, one of the characters embossed on the keytop is transmitted to the computer.

Notice that the keyboard has a SHIFT key. When you depress this key, you get the uppercase equivalent of the character shown on the letter key you are typing, or the character shown on the upper portion of the keytop. You automatically get the lowercase version of a key when the SHIFT key is not depressed.

Also at the left side of the keyboard is an ALPHA LOCK key. This key is used like a SHIFT key, except that it affects only the letter keys, having no effect on other keys. Once pressed, the ALPHA LOCK key remains depressed, and all letters selected are uppercase until the ALPHA LOCK key is again pressed.

The key labeled CTRL is known as the "Control key." You use this key to enter characters that control operations within the computer. You might think of these characters as simple commands to the computer. As with the SHIFT key, you need to depress the Control key before you press the other key, in order for the computer to recognize that you mean to give it a command.

We use the following notation throughout this manual to identify a key that is being used as a control key:

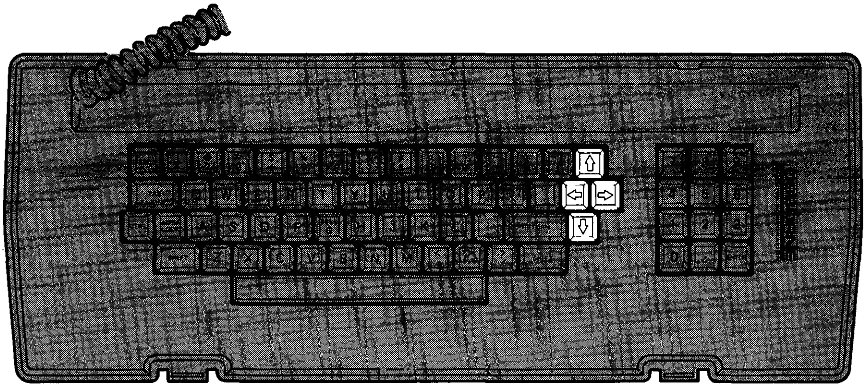

represents CTRL *key depressed*
^X
character that is simultaneously typed

A CONTROL KEY

^→ is entered by depressing the CTRL key and the → key simultaneously. Thus, ^→ means that the right arrow is being used as a control character; it does not mean to press the "^" key followed by the → key.

The ← key, when not used in conjunction with the CTRL key, backspaces over and erases unwanted characters you have typed. You can also use the key marked ESC in some programs to backspace over and erase unwanted characters. The key marked RETURN is used like the carriage-return key on a typewriter; it indicates to the computer that you have finished typing a line of information.

A few keys serve special functions on the Osborne 1. Look at the four arrow keys.



Most programs let you use the four arrow keys to move a small, bright underline, called a cursor, around the screen. Whenever you press a key, its corresponding character appears at the cursor position; the cursor then moves one character position to the right, just as the carriage or typing element on most typewriters moves one position each time you strike a key. The cursor moves in the direction indicated by the arrow key you press: left [←], right [→], up [↑], or down [↓] .

You can always use the arrow keys with the control key depressed to move the whole display one direction or another. To move the display back to its normal position, simply hold down the [CTRL] and bracket [[] keys.

Besides the keys shown on the keyboard, four additional characters are available. These four characters are accessible as control characters, meaning that the CTRL key is depressed while the key associated with the character is pressed. Here are the characters and the control sequences required to access them:

^/ = tilda (~)

^< = left bracket ({)

^> = right bracket (})

^= = open single quote (')

NOTE

You will not be able to enter any characters on the screen until we show you how.

Restarting the Computer

To the right of the brightness and contrast knob you will see a small protruding button with the word **RESET** marked underneath it. This button "resets" (restarts) the computer.

When you reset the computer, it stops doing whatever it was doing and returns to the condition it was in when you first turned the power on. When you press this button, any information in the computer's memory is virtually lost. Also, if either of the disk drives are in use—the little red light on each lights up when they're in use—you may accidentally destroy information on that diskette if you press the RESET button.

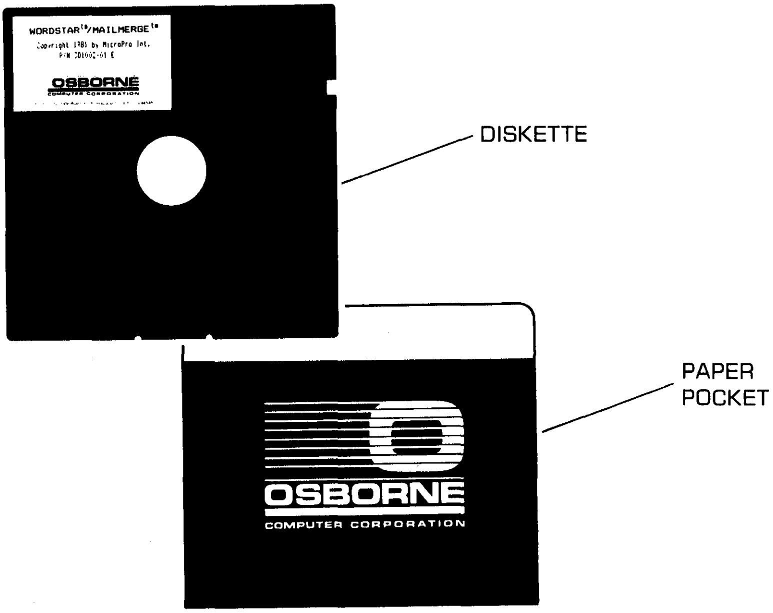
Use the RESET button judiciously; otherwise you may lose data you intended to save.

Diskettes and Disk Drives

The Osborne 1 has two disk drives, one on each side of the monitor. These drives read information from or write information onto diskettes.

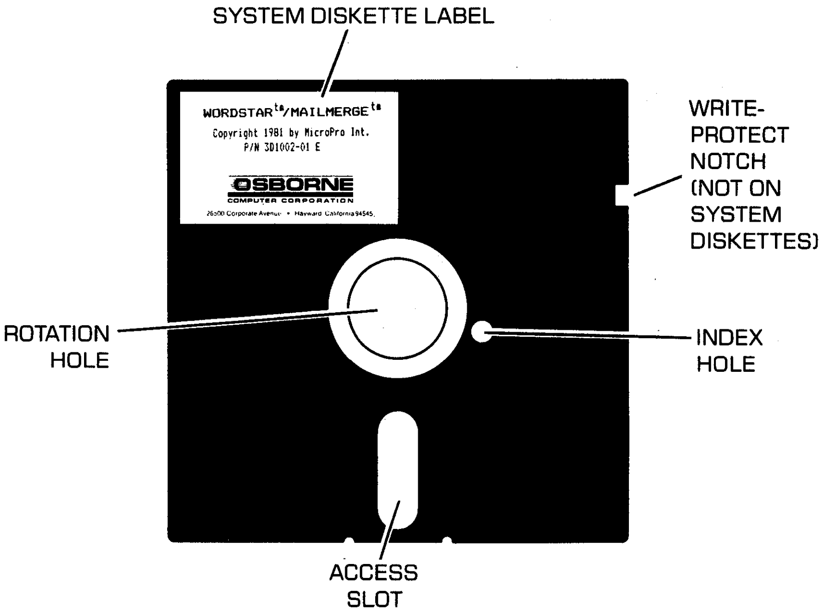
The disk drive on the left is labeled drive A, and the one on the right is labeled drive B. You should find some diskettes in a small box that accompanies the Osborne 1.

A diskette is a thin plastic disk with a magnetic coating on its surface. A diskette works along the same principles as does cassette tape, only diskettes are shaped differently. The surface of a diskette—on which information is stored—is permanently contained in a cardboard envelope. Diskettes in their cardboard envelopes are stored in a paper pocket.



To use a diskette you must remove it from the paper pocket. Do not try to remove the plastic disk from the cardboard envelope, or you will destroy the diskette.

The next illustration shows an ordinary diskette and identifies its important features.


SYSTEM DISKETTE LABEL
WORDSTAR tm/MAILMERGE tm
Copyright 1981 by MicroPro Int.
P/N 3D1002-01 E
OSBORNE
COMPUTER CORPORATION
26500 Corporate Avenue • Hayward California 94545
WRITE-PROTECT NOTCH (NOT ON SYSTEM DISKETTES)
ROTATION HOLE
INDEX HOLE
ACCESS SLOT

You must treat diskettes with care. It is easy to damage a diskette and lose the information stored on it if you are sloppy.

Do not place a diskette near any magnetic field, since this erases all information stored on the diskette. Television sets, telephones, large appliances, and stereo speakers are common objects that emit magnetic fields. Keep your diskettes at least a foot—preferably more—away from such devices.

Dust and scratches also damage a diskette's surface. Keep all exposed surfaces protected from dust and contact with foreign material. Store your diskette in a cool, clean, dry place.

A long slot in the cardboard envelope containing the diskette exposes the diskette's magnetic surface. Mechanisms within the disk drive use this slot to read information from the magnetic surface, or to write information to it. Do not touch the magnetic surface with your fingers, or you may leave finger oil on the diskette, resulting in the disk drive's inability to use the information contained at that spot.

A spindle in the disk drive uses the large, round hole in the center of the diskette to rotate the diskette within the cardboard envelope. The disk drive also uses the small, round hole at the side of the large center hole as an index, or "starting position," on the diskette surface.

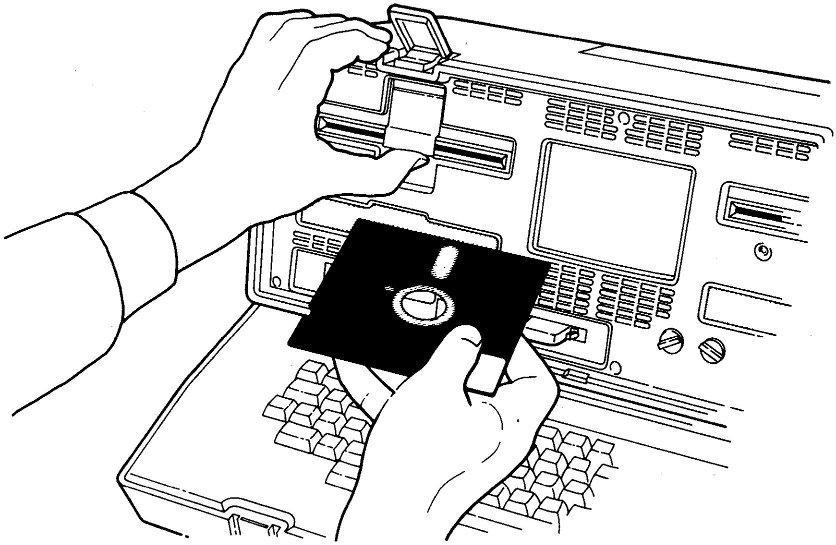
Turn the diskette within its cardboard envelope; you will see an index hole occasionally align itself with a small hole in the cardboard. All diskettes the Osborne 1 computer uses have one index hole; these diskettes are called "soft sectored." Some diskettes are "hard sectored"; they have numerous index holes—usually 10 or 16—and the Osborne 1 computer cannot use them.

NOTE

When ordering diskettes for your computer, always specify "soft-sectored", single-sided 5-1/4-inch diskettes, either single- or double-density, depending on your computer. If your dealer is out of this type of diskette, you may substitute double-sided or double-density, but these may cost you more money. Other substitutions will not work.

Putting Diskettes Into Drives

The disk drives of your Osborne 1 have doors you must open to insert or remove diskettes. The doors, which resemble small flaps, should lift easily.

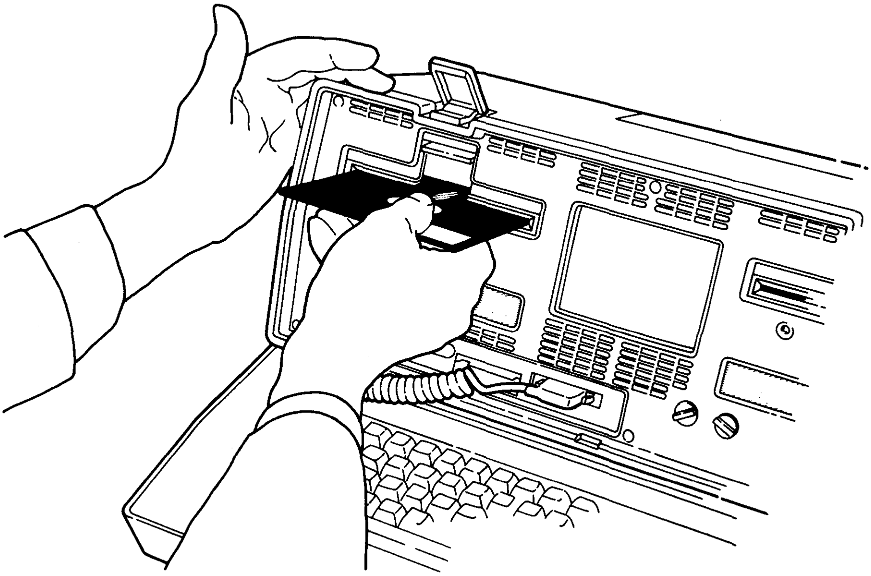


Notice the small red light just to the left and below each door. This "disk activity light" tells you whenever the diskette in that drive is in use.

Insert diskettes into the drives with the access slot pointing forward, and make sure that the small index hole is positioned to the left of the larger hole. Usually a label or manufacturing emblem will be on the top and facing outward. A small notch should be positioned closest to you, on the left side of the diskette as you insert it into the drive.

NOTE

To avoid accidental erasure, some system diskettes are supplied with no write-protect notch. Other diskettes may be write-protected with a piece of tape or sticky foil over the small notch.



When the diskette is securely in the drive, close the door. This door must be closed for the disk drive to work properly. To remove the diskette, flip the door open and gently pull the diskette out. When inserting or removing the diskette, do not bend it.

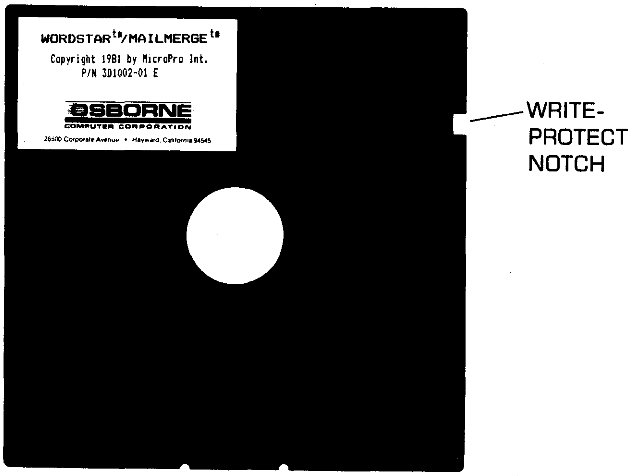
Never attempt to remove a diskette from a drive while the disk activity light is on. Doing so may cause you to lose information. You can insert and remove diskettes while the computer is running, but not while the drives are active. The disk activity lights are there to inform you when the drive is working and when it isn't; look at them before removing a diskette. There is no exception to this rule.

If you must remove a diskette when the computer is trying to "access" it (read from or write to it) , the correct procedure is to press the RESET button to interrupt whatever the computer is doing. If the computer is trying to write information onto a diskette when you press the RESET button, you run the risk of permanently losing information on the diskette, so this should be a "last ditch" measure only.

Remember that your Osborne 1 has two disk drives. You already know that the left drive is referred to as drive A and the right one as drive B. For now, simply note that you'll usually place the diskette containing your programs in the left drive (A) and the diskette onto which you wish to save your data in the right drive (B).

Some Last Words About Diskettes

Many diskettes have a "write-protect notch," shown in the following drawing. You can protect data on a diskette by placing a small label—or a specially designed silver tab you receive when you buy your diskettes—over the notch. Do not use transparent tape, masking tape, or anything else that has a "gooey" adhesive that may ooze out and possibly contaminate the disk drives.



When the write-protect notch is covered, information can still be read from your protected diskette, but data cannot be written onto the diskette. Valuable programs and data should be protected in this manner to prevent accidental overwrites or erasures.

The first thing you should do with any system diskettes that have write-protect notches is place a write-protect tab on each one. After you've made copies of these "master" diskettes as outlined in the next chapter, store the original diskettes in a safe, out-of-the-way place. It's a good idea to make a spare copy of every diskette you use with your system and to keep these "backups" apart from the diskettes you use every day.

Information is stored on diskettes in the form of "files." You can visualize a diskette file much as you would a manila envelope in a filing cabinet. All files have names with which you reference them. You'll be naming your files as you create them.

A file can contain any kind of information; its size is limited only by the capacity of the diskette. On your Osborne 1 in its single-density configuration, an individual file can contain up to 92,000 characters of data. (The system uses another 10,000 characters on the diskette, and they are not normally available to you). This translates to almost 25 pages of text if you assume each page has 65 characters on each of 55 lines.

Double-density diskettes can store 204,800 characters of data. Making similar allowance for system storage, there would be an available balance of about 185,000 for storing files. This translates into a bit more than 50 pages of text by the same measurements.

You should always clearly label your diskettes so that you can tell what you stored on them and find the diskette you need quickly. Always write on a label BEFORE you put the label on the diskette. If you must write on the label after it's on the diskette, use only a felt-tip pen and write as gently as possible; pen creases destroy diskettes.

Your Osborne 1 has two diskette holders, located under the disk drives, in which you can store diskettes. Each holder accommodates approximately 10 diskettes. Don't stuff too many diskettes into a holder, or you may have trouble removing them.

When you handle diskettes, it is always better to err on the safe side; put about eight diskettes in each holder and you shouldn't have any problems removing them.

Moving On

Now that you're familiar with the Osborne 1 and the diskettes you'll use, it's time to start learning how to use the machine.

Since the next chapter will tell you how to make copies of diskettes, make sure you have a few extra BLANK diskettes and about 30 minutes of continuous time available. If you don't have any extra diskettes or are rushed for time, we encourage you to set aside this manual and your task of learning about your new computer until you do.

If you accidentally erase a file from a master diskette because you're in a hurry or didn't make diskette copies before proceeding, you will lose more time than you thought you were gaining. There is nothing difficult or overly time consuming in the next chapter; we simply don't want you to waste time because of haste.

CHAPTER 2

How To Use Your Computer

In this chapter you'll begin familiarizing yourself with how to use the Osborne 1 efficiently.

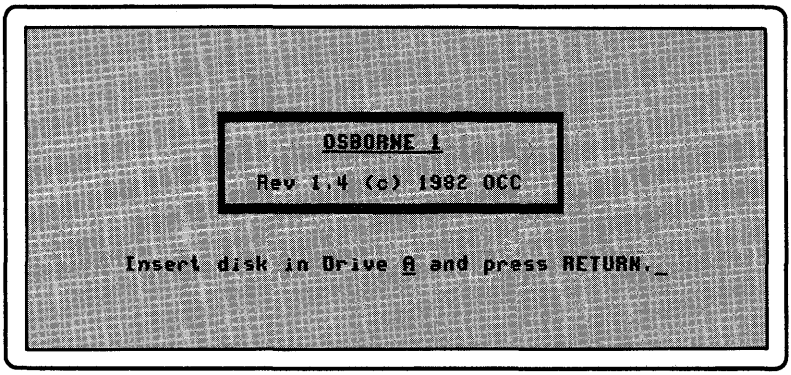
Plug It In, Turn It On

Now that you know a little about the Osborne 1, sit down in front of the keyboard. Make sure the vent on top is open; if it is not, push down on the vent cover and slide it away from you toward the handled side of the computer. We know that you already turned the computer ON in the last chapter, but just for the sake of procedure let's start from scratch.

Although the Osborne 1 does not draw a lot of power and is not likely to trigger a circuit breaker, it is wise to isolate the computer from as many other devices as possible. Sometimes electromechanical devices such as Selectric typewriters emit what are called voltage "spikes" when they are turned ON and OFF. Such electrical surges are a menace to computers, especially if the spike occurs when the computer is accessing a diskette. To be extra safe, ask your Osborne dealer about a voltage-protector into which you can plug your computer.

The power switch is at the rear right-hand side of the computer (as you face it). Make sure that there are no diskettes in the drives, and if you haven't done so already, turn the computer ON.

You should hear a small "beep" when you turn the Osborne 1 on. Watch the monitor. Like a TV set, it will take about 20 seconds to warm up. You should see a display similar to the one that follows:



```
                 OSBORNE 1

           Rev 1.4 (c) 1982 OCC


    Insert disk in Drive A and press RETURN._
```

We'll refer to the message on the screen as the "sign-on message." Notice that the sign-on message asks for a diskette. Although the computer is somewhat "intelligent," when you turn it on, its memory is blank. Every time you switch the power ON, you are essentially dealing with a newborn computer. This is an important fact to remember when you use any small computer.

In order to communicate and work for you, the computer needs the instructions that are on one of your program diskettes. Information you generate while the computer is active will be stored in its memory. When you turn the Osborne 1 OFF, it loses all the information that is located in its memory—one reason why diskettes are used to permanently store information.

Loading a Program

You received several diskettes with your Osborne 1. Let's examine one of them.

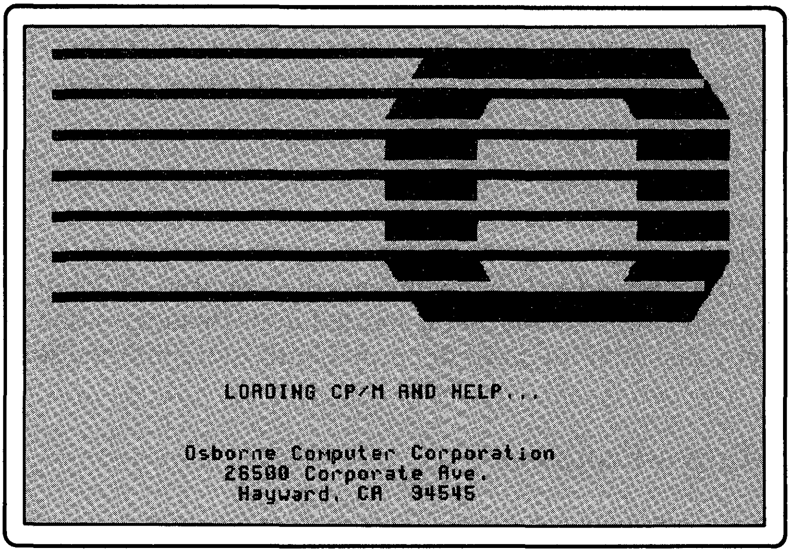
Each diskette has a label with a name, an identification number, and a serial number. You're looking for the diskette labeled **CP/M System**.

We'll refer to this diskette as the "master system diskette." When you eventually make a copy of this diskette, we'll call the copy "system diskette."

Remove your master system diskette from its envelope. Hold the diskette between your thumb and index finger with the label facing upwards and the small index hole on the left. Lift the door on the left-hand disk drive (remember, it's drive A) and carefully slide the master system diskette into the drive. It should go in smoothly and require no force. Now close the door.

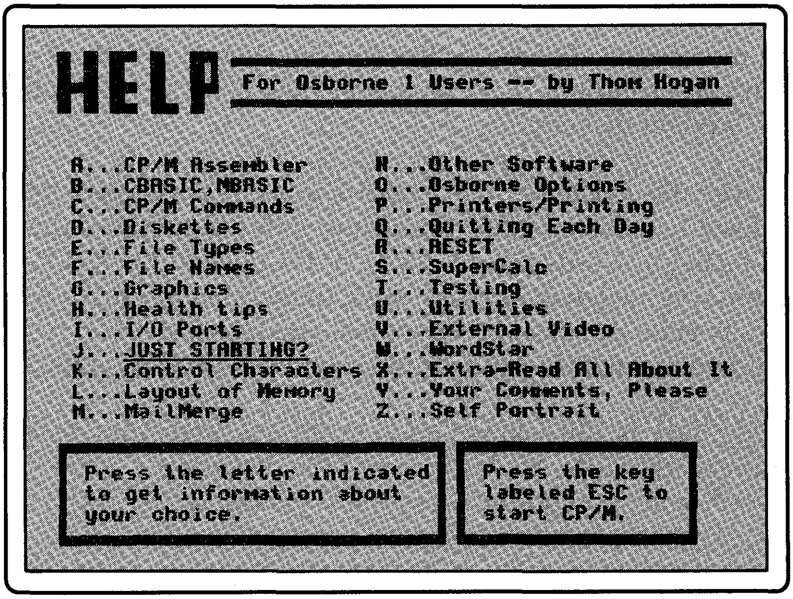
Your monitor is still asking you to insert a diskette into drive A and press RETURN. You've inserted the diskette, so as instructed, you should now press the key labeled RETURN.

The red activity light will come on, you'll hear some whirring noises, and within a few moments a message that looks like this should appear:



```
LOADING CP/M AND HELP...

Osborne Computer Corporation
   26500 Corporate Ave.
    Hayward, CA  94545
```

The disk activity light will remain on, and you'll hear some more whirring noises from the drive. Within ten seconds the monitor screen should clear and then fill with this information:



```
HELP  For Osborne 1 Users -- by Thom Hogan

A...CP/M Assembler        N...Other Software
B...CBASIC,MBASIC         O...Osborne Options
C...CP/M Commands         P...Printers/Printing
D...Diskettes             Q...Quitting Each Day
E...File Types            R...RESET
F...File Names            S...SuperCalc
G...Graphics              T...Testing
H...Health tips           U...Utilities
I...I/O Ports             V...External Video
J...JUST STARTING?        W...WordStar
K...Control Characters    X...Extra-Read All About It
L...Layout of Memory      Y...Your Comments, Please
M...MailMerge             Z...Self Portrait

Press the letter indicated     Press the key
to get information about       labeled ESC to
your choice.                   start CP/M.
```

What's happened here? Your Osborne 1 has loaded into its memory a program that contains a wealth of information about the system. We call this program **HELP** because it's designed to "help" you learn about your Osborne 1.

Your screen should now display 26 topics of interest, one for each letter of the alphabet. To obtain some information on any one of the topics, simply press the letter that corresponds to the subject of interest. For starters, press the J key. You should see the screen clear, and then some new information dealing with the operation of your Osborne 1 appears. As indicated at the top of the screen, press the RETURN key to return to the main menu.

The HELP program is virtually foolproof. Go ahead and press any key except the one labeled ESC (we're saving that option for later). If your choice is a valid one, you'll see the information you requested; all incorrect choices produce a "beep" indicating that your selection is invalid.

At this point, you might want to spend some time exploring the other topics shown in the menu. Some are designed for advanced users; you can ignore these for now.

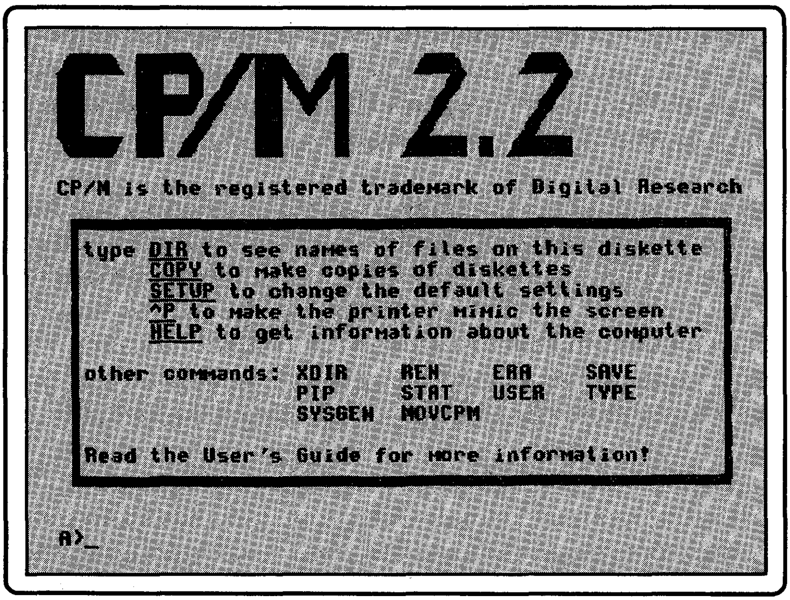
Leaving HELP

Okay, now that you've had a chance to use your first computer program, it's time to move on. Before you accidentally damage your master system diskette, we want to teach you how to make copies of your diskettes.

To leave HELP, simply press the key labeled ESC (for "escape").

The disk activity light comes on briefly, the screen clears, and a giant **CP/M** appears at the top of the screen with a copyright message underneath, and eventually an A> appears:

Besides the CP/M programs on your System and Utility diskettes, CP/M is also a general set of instructions that controls



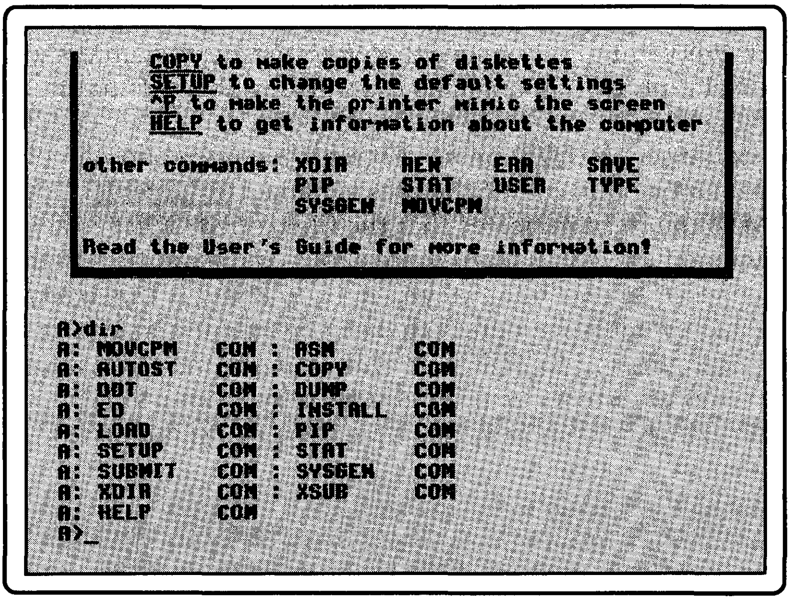
```
CP/M 2.2

CP/M is the registered trademark of Digital Research

type DIR to see names of files on this diskette
     COPY to make copies of diskettes
     SETUP to change the default settings
     ^P to make the printer mimic the screen
     HELP to get information about the computer

other commands: XDIR    REN    ERA    SAVE
                PIP     STAT   USER   TYPE
                SYSGEN  MOVCPM

Read the User's Guide for more information!

A>_
```

how your Osborne 1 works. CP/M, which stands for Control Program/Monitor, is called an "operating system" because it ties all the computer's components together and allows them to communicate. The CP/M operating system actually loaded into the computer when you first pressed the RETURN key in response to the sign-on message. CP/M, in turn, automatically loaded the instructions that constitute the HELP program, which you were just using.

Now we want to tell CP/M what to do next.

The A> that appeared is the standard CP/M greeting. It means that CP/M is ready for a command. The **A** in the message indicates that CP/M will perform your commands using the diskette in drive A (the left one), unless you precede the command with a **B:** in which case the B drive is accessed.

Let's be inquisitive for a moment and find out what files are stored on the **SYSTEM** diskette. Type the directory command D I R, then press the RETURN key. Again the disk drive will hum, and then a directory listing all of the files on the diskette will appear on your screen.



```
      COPY to make copies of diskettes
      SETUP to change the default settings
      ^P to make the printer mimic the screen
      HELP to get information about the computer

other commands: XDIR    REN    ERA    SAVE
                PIP     STAT   USER   TYPE
                SYSGEN  MOVCPM

Read the User's Guide for more information!

A>dir
A: MOVCPM   COM : ASM      COM
A: AUTOST   COM : COPY     COM
A: DDT      COM : DUMP     COM
A: ED       COM : INSTALL  COM
A: LOAD     COM : PIP      COM
A: SETUP    COM : STAT     COM
A: SUBMIT   COM : SYSGEN   COM
A: XDIR     COM : XSUB     COM
A: HELP     COM
A>_
```

The files contained on any diskettes can be listed in this manner. You assign a name to each new file as you create it. We'll deal with this subject at length in the next chapter; for the time being, note that your master system diskette already contains some files.

Formatting a Diskette

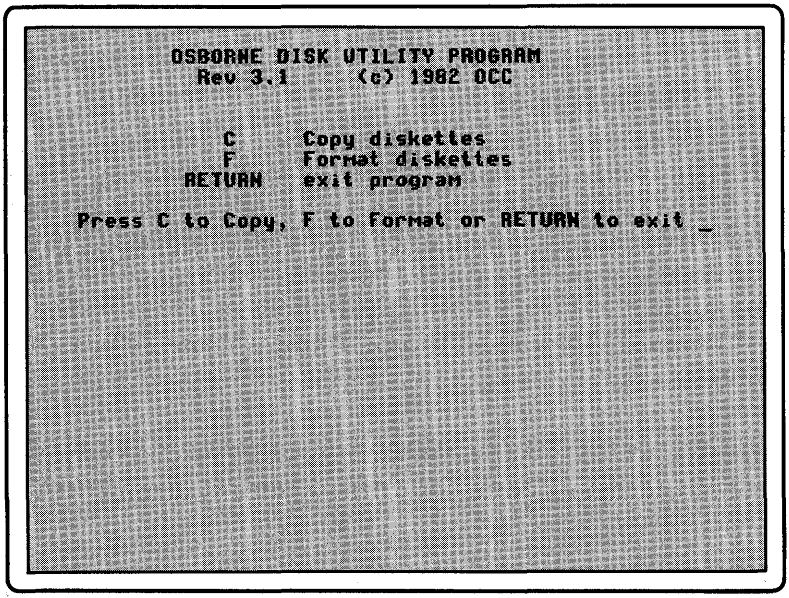
You don't need to know much about the mechanics of disk drives. What you do need to know is that information is written onto the diskette's surface in a specific way. Unfortunately, you cannot simply load a brand new diskette and expect the Osborne 1 to start writing information onto its surface. First the surface of the diskette must be prepared by a process known as formatting.

Formatting is a routine procedure that is performed automatically by the COPY program before the contents of a diskette are transferred to another. Generally, only those diskettes being used for storage of documents or reports need to be specifically formatted. You have to format a diskette only once. You can reformat a diskette you've used before, in which case all prior information on the diskette is erased.

Before proceeding, make sure that your master system diskette has a write-protect tab on it, as described in the last chapter.

You should see the A > on your screen just below the directory of files. Notice that one of the files is named COPY.COM. The **.COM** to the right indicates that the file named COPY is a "command" file. You'll learn more about command files in the next chapter, but for now, all you need to know is that a command file contains a program that can be run just by typing its name and pressing RETURN. In this case, the program which allows you to format diskettes is started by issuing COPY—then pressing the RETURN key. So, type C O P Y and press RETURN.

The computer accesses the diskette in drive A and transfers the instructions from the COPY file into the computer's memory. Upon completion of this loading procedure, the computer begins to follow the instructions. The first thing the instructions tell the computer to do is to display the following message on your monitor screen:



```
          OSBORNE DISK UTILITY PROGRAM
            Rev 3.1      (c) 1982 OCC

               C      Copy diskettes
               F      Format diskettes
             RETURN   exit program

      Press C to Copy, F to Format or RETURN to exit _
```

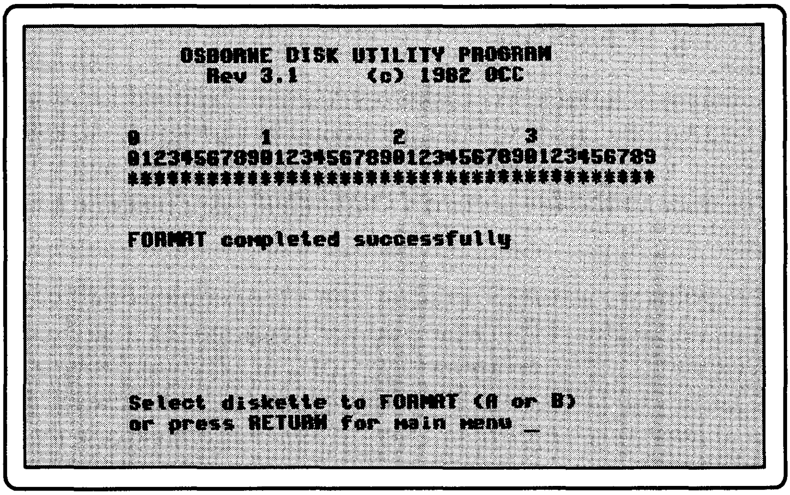
Let's format a diskette in the second drive, drive B. Take a ***Brand-New Diskette*** from its envelope and insert it into the right-hand drive in the same manner you inserted the master system diskette earlier. Close the door over the diskette.

The COPY program is waiting for you to press either a C, to copy a diskette, or F, to format a diskette. This should be clear from the instructions on the screen. (Do take time to read what is displayed on the screen; you'll develop bad habits if you merely assume what the message says.)

You want to format a diskette to be used later for storage, so press the letter [F]. Immediately, the screen changes and waits for you to press either an A or a B to indicate which drive the diskette you wish formatted is in. You placed your new diskette in drive B, so press the letter [B]. You should notice that the computer responds to this action by telling you to press **RETURN** when your diskette is ready (i.e., in the drive and ready to be formatted). After pressing RETURN, you are asked whether your computer is Single or Double density. (See notes on single and double density, page 758.)

Press [S] for single or [D] for double.

Formatting is now taking place. You should soon see the display illustrated below:



```
    OSBORNE DISK UTILITY PROGRAM
      Rev 3.1     (c) 1982 OCC

0         1         2         3
0123456789012345678901234567890123456789
****************************************

FORMAT completed successfully




Select diskette to FORMAT (A or B)
or press RETURN for main menu _
```

As the drive formats the diskette, asterisks are displayed under the row of numbers. The numbers refer to "track" numbers. When formatted, an Osborne diskette has 40 concentric tracks of information on it. Thus, each * you see displayed under the row of numbers indicates the track that has been formatted correctly. An E occurs instead of an asterisk if the track was not formatted correctly. If an error occurs, perform the format process again. If you still get an **E**—especially if it occurs in the same place—your diskette is probably damaged or is the wrong type, and should be replaced with another.

When the formatting is complete, this message appears:

```
Formatting completed successfully
```

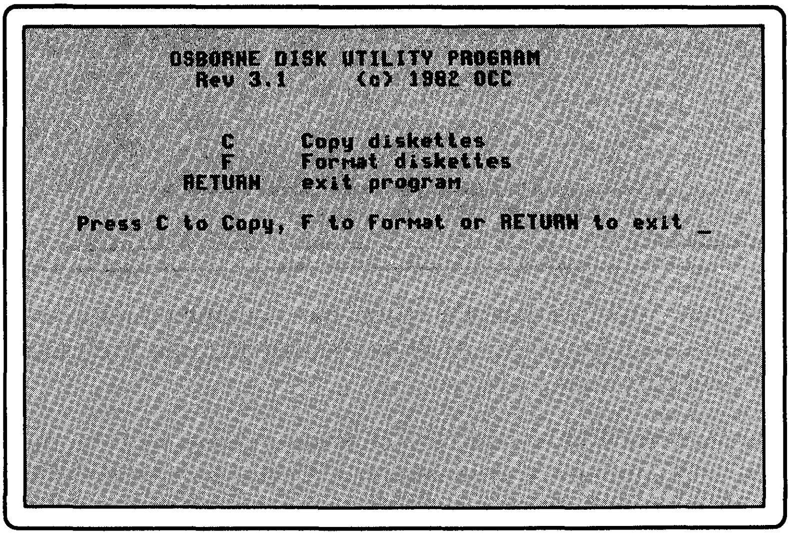
At the bottom of the screen you will see the question asking which drive contains the diskette to be formatted. If you'd like to format several diskettes, you should keep replying with a **B** each time this question is asked and insert new diskettes into drive B when prompted to do so by the computer. Pressing the RETURN key once will take you back to the original choices of "C"opying, "F"ormatting, or returning to CP/M. Pressing the RETURN key a second time returns you to CP/M's control, where you should see the now-familiar A > indicating that CP/M is waiting for another command.

Copying Diskettes

Now that you have a few diskettes ready to be used for storing files you will be creating later, let's copy the program diskettes that came with your Osborne 1. You do this by employing the instructions in the file named **COPY** which you just used to format a diskette.

At this point we strongly suggest you make copies of all your program diskettes, otherwise you are in danger of losing some very valuable software. In fact, you may wish to make two or more copies of each, as diskettes do wear—usually they stand up to at least three months of heavy use, but are extremely vulnerable to damage.

You already have your master system diskette in drive A, so let's begin by making a copy of it. Type C O P Y and press RETURN. In a few moments you'll again see the Copy program menu:

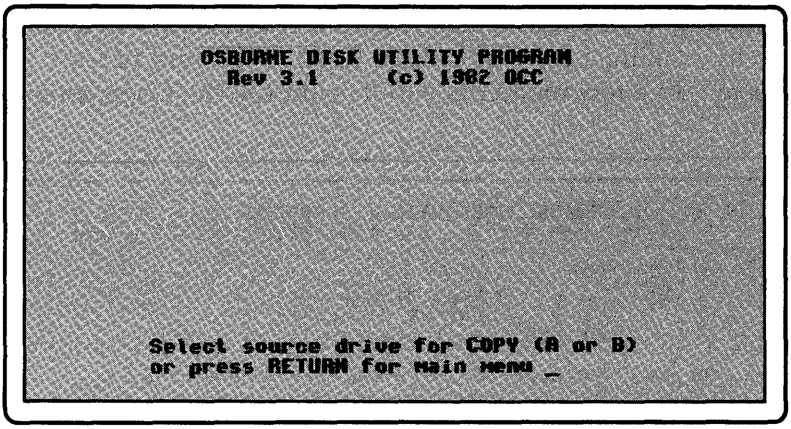


```
          OSBORNE DISK UTILITY PROGRAM
            Rev 3.1      (c) 1982 OCC

              C      Copy diskettes
              F      Format diskettes
           RETURN    exit program

     Press C to Copy, F to Format or RETURN to exit _
```

The screen displays three choices, as before. Instead of pressing **F** to format a diskette, press C to copy diskettes. You will be asked to place the original source diskette (the one you wish to make a copy of) in drive A and the destination diskette (a blank diskette or one which you wish to reuse) in drive B.

You just used the COPY program to format a blank diskette; but the COPY program also formats a blank diskette while copying information to it. In other words, if you wish to prepare a blank diskette for storing files, you select the formatting option (F) of the COPY program, but if you want to copy an entire diskette, you select only the copy option (C); you do not need to format diskettes before copying to them, only when using them for storage.

In case you haven't put your diskettes into the proper drives yet—and also so that you develop the habit of verifying the instructions you give the computer—the Osborne 1 displays the following prompt:



```
          OSBORNE DISK UTILITY PROGRAM
            Rev 3.1      (c) 1982 OCC





      Select source drive for COPY (A or B)
      or press RETURN for main menu _
```

If you are copying a diskette other than the one which contained the COPY program, follow the instructions and place it in drive A at this time.

Press the RETURN key and the copying process will commence. As the copy operation proceeds, you receive a message telling you which track is being copied. Rarely, you might see a message telling you of an error during the copying process. If this happens, first try copying the diskette again. Should you still get errors, try formatting the destination diskette before using the copy option. If the same error messages appear, you probably have a defective diskette.

It is sometimes possible to "fix" damaged diskettes, and Osborne sells a program, called Disk Doctor, that will help you do so. Inquire at your dealer if you're interested in this program.

NOTE

Although Osborne warrants that your machine and diskettes are free from manufacturing defects for 90 days, our limited warranty does not cover misuse or abuse of diskettes. Diskettes whose files are erased "accidentally" or that have fingerprints or other contaminants on the media surface will not be replaced free of charge by your dealer or by Osborne Computer Corporation.

If you receive error messages frequently when using the COPY program, it is likely that your Osborne 1 is out of adjustment and needs servicing, or that you are using poor-quality diskettes. If you experience problems with low-cost diskettes, try switching to high-quality, certified diskettes before bringing your Osborne 1 in for servicing.

At the end of the copy process, a message appears telling you that the procedure was completed successfully. In addition, you will be offered a chance to continue copying diskettes or to return to the copy/format menu that appeared when you first started COPY.

You should now proceed to make copies of all of the diskettes you received with your Osborne 1. Place the diskette you wish to copy in drive A and follow the instructions just given. You can make as many copies as you desire of the diskettes we provide, without violating the copyright or licensing agreement you signed when you purchased your computer. You cannot sell these copies or give them to a friend; to do so is illegal. Since all Osborne-supplied software contains hidden serial numbers, any copies you give away can be traced. The companies who created the programs you'll use allow you to make copies for YOUR OWN USE ONLY.

Be sure to label each diskette as you create new copies. (You may want to include the date on which you created the copy so that you'll know later how long you've been using that diskette.)

DO NOT GO ON IN THIS MANUAL UNTIL YOU'VE MADE COPIES OF ALL YOUR OSBORNE-SUPPLIED DISKETTES. Put the originals, the ones that came with the system, away in a safe place and use only the copies from here on. The diskettes we supply you are only for creating new copies. If you had to replace the programs and information on these diskettes at list price, you would spend over $1400. The original diskettes are valuable and should be treated as such.

Reset

In chapter 1 we referred to the RESET button on the front of the Osborne 1. "Reset" is a computer term that indicates a specific sequence of events within the computer.

All you have to know about the RESET button on your Osborne 1 computer is that it functions as a "restart" button. When you push RESET, the screen clears and the sign-on message you saw when you first turned on the power

```
Insert disk in drive A and press RETURN
```

reappears. Pushing RESET, then, is much like turning the power off and starting over.

It is important to understand that pushing the RESET button at the wrong time will make the computer "forget" any information in its memory that has not yet been transferred onto a diskette. Thus, if you're in the middle of entering information into the computer and have not told it to save the information on a diskette, all of the information stored in the computer's memory will be lost. Later chapters will show you how to save information on a diskette.

If you always ask yourself "Have I told the computer to save my entries on a diskette?" before you push RESET, you probably will never lose valuable data by accidentally pushing RESET at the wrong time. **NEVER PUSH RESET IN PANIC OR FRUSTRATION; ALWAYS MAKE SURE YOU REALLY MEAN TO DO SO BEFORE PRESSING THE RESET BUTTON.**

More HELP

Whenever you use your system diskette, you'll find that the HELP program is loaded into your Osborne 1's memory and that the computer executes the instructions in it. This is the process:

1. The Osborne 1 displays the sign-on message and waits for you to insert a diskette and press the RETURN key.
2. After you press the RETURN key, the instructions that comprise CP/M are transferred from the diskette into the computer's memory. One of these instructions tells CP/M to perform the next step automatically.
3. A copy of the file containing the Osborne logo goes into memory, and the monitor displays the logo.
4. A final set of automatic instructions loads the HELP program into the Osborne 1's memory and the computer begins following the instructions that make up the HELP program.

Each diskette Osborne provides follows these steps; the only difference is in step four. The diskette that contains WordStar automatically loads the WordStar instructions during step four and begins following them; the SuperCalc diskette loads SuperCalc during step four; the BASIC diskette loads and executes Microsoft BASIC during step four. As you'll eventually learn, you can change the program the Osborne 1 first loads. For now, be content that the CP/M System diskette always loads and executes HELP first, the WordStar diskette automatically loads and executes WordStar, and so on.

Setting Up

Before we finish acquainting you with the Osborne 1, we want to introduce you to another program. Called SETUP, this program performs numerous useful functions.

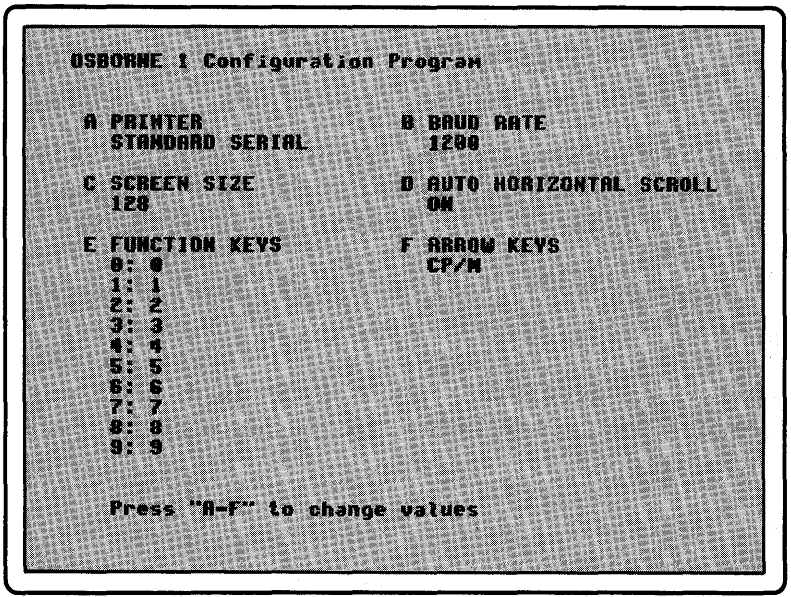
Your system diskette should be in drive A at this point. Type:

[S] [E] [T] [U] [P]

then press [RETURN].

After a few moments, a new message will appear on the screen identifying the program and asking you which drive has the diskette containing the CP/M system to be altered. This may not make any sense right now, but go ahead and press A to indicate that you want to use the system from drive A. It next reminds you that the source diskette should be in the A drive. Then press RETURN.

In a few more moments, you'll see the following display:



```
OSBORNE 1 Configuration Program

 A PRINTER                     B BAUD RATE
   STANDARD SERIAL               1200

 C SCREEN SIZE                 D AUTO HORIZONTAL SCROLL
   128                           ON

 E FUNCTION KEYS               F ARROW KEYS
   0: 0                          CP/M
   1: 1
   2: 2
   3: 3
   4: 4
   5: 5
   6: 6
   7: 7
   8: 8
   9: 9

   Press "A-F" to change values
```

You've probably already figured out that you can use the SETUP program to "set up" the CP/M operating system so that it recognizes your printer as well as providing a few other convenient options. (You will learn all about the operating system in the next chapter.) The SETUP program shows you the name of the options in dim letters and the current setting in brighter letters. Let's take each one individually.

A PRINTER: Your Osborne 1 can use five types of printers:

- Standard Serial (uses RS-232 communications protocol);
- Qume or NEC (uses ETX-ACK communications protocol);
- Diablo (uses XON-XOFF communications protocol);
- Centronics (uses parallel communications protocol);
- IEEE-488 (uses IEEE-488 communications protocol).

Almost every printer manufactured falls into one of these five categories. If you're not sure where your printer fits in, check with the dealer who sold it to you before proceeding. If you don't have a printer yet, you can skip the rest of this printer option discussion.

Currently, the setting indicates that **Standard Serial** has been chosen. If you wish to change to another type of printer, press the letter [A] to signify that it's the **PRINTER** option you want to alter (followed by RETURN for double-density). This gives you the five choices just mentioned. Each choice has a letter associated with it; just press the letter of your choice and the CURRENT PRINTER setting automatically changes. Your choice isn't permanently recorded until later, so don't worry about making a mistake at this point.

Also in the printer catagory are the options for addressing a particular IEEE device number and a feature for inputting a printer initialization string. Both of these options should be used only by those knowledgable in their implementation. Use the [RETURN] to get from the printer menu back to the original display as indicated in the message at the bottom of the screen.

B Baud Rate: The second setting you may wish to change is the "speed" at which the serial port on your Osborne 1 communicates. There are two choices: 300 baud (30 characters per second) and 1200 baud (120 characters per second). Pressing the letter [B] changes the selection from its current setting to the other one. Pressing [B] again changes it back.

C Screen Size: You can tell your Osborne to "think" that its screen is any number of characters wide, from 1 to 128; the screen will still show only 52 at a time, however. The ability to change the screen size is handy if you're using software designed for a different computer, one with a display size different from the Osborne's. To change the screen size, press [C], then press [A], [B], or [C] , depending upon whether you want 52, 128, or some other size. If you press **C**, you'll need to enter the size you desire (1–128, inclusive).

Be careful about what size you specify; some sizes don't make any sense or will have an effect on the programs you run. NEVER CHANGE THE SCREEN SIZE TO ANYTHING EXCEPT 128 ON YOUR WORDSTAR DISKETTES, for instance, or you may find the program MAY not function properly. Changing the system diskette to 52 will also make the HELP display a little erratic. We recommend leaving the setting on all diskettes at 128.

D Auto Horizontal Scroll: We've done everything in our power to make your Osborne 1 more useful than systems with larger screen sizes. Consequently, we've come up with an optional setting that makes the screen automatically scroll horizontally when you move the cursor past the 52nd character on a line.

As with selection B, two choices exist. Pressing [D] turns automatic scrolling ON or turns it OFF, depending upon its current setting. We recommend that you leave it ON; WordStar diskettes should always have this feature ON, while SuperCalc diskettes should always have this feature OFF.

E Function Keys: When you press the control key and any of the number keys, something special happens. A **^5** tells your Osborne 1 to automatically display the characters associated with function 5. Simply stated, the function keys (0–9, each issued by holding the control key down) can be programmed to represent a sequence of characters or commands. Therefore, these keys become a shorthand method of issuing the desired command or entering a sequence of characters. For example, if you found yourself constantly typing the word ANTIDISESTABLISHMENTARIANISM, you could enter that sequence of characters for a given function key. After programming the function key, you type the word ANTIDISESTABLISHMENTARIANISM by simply pressing the associated function key (i.e, holding down the control key while pressing a number key).

The **E** option of the SETUP program lets you program each function key. Let's try programming some function keys!

Press [E] to see the function-key menu. Next, press a number key (0–9) to begin programming that key. For a quick demonstration, press [1]. Notice that your cursor sits next to a **1:** and that a message states that there are **77 (D.D.) Characters Remaining**. This last message means that you can use up to 76 characters for the ten separate function keys, but no more than that. Try issuing the CP/M command below:

[D] [I] [R] [SPACE BAR] [*] [.] [*] [RETURN]

The **<cr>** is a shorthand way of signifying a **RETURN**, and is shown along with the rest of the command for function key 1. You inform SETUP you're done programming a function key by pressing [ESC] twice in succession; do so now. Note that the command you programmed for function key 1 is now listed to the right of the **1:** in the table at the top of the screen.

You can continue programming the keys until you're satisfied that all are exactly as you want them and then press [RETURN] to get back to the original SETUP display.

We suggest that you refrain from programming the function keys for any diskette until you've used that diskette for a while. Note which commands or sequences of characters you use often, and program these into the function keys.

F Arrow Keys: The arrow keys are also function keys of a sort, but very special ones. Unfortunately, CP/M and WordStar expect the arrow keys to have different internal values associated with them. For the arrow keys to work properly in WordStar, they must have one set of values; to work properly with all other software, they need a second set. It is best not to mess with this option unless you know what you are doing.

As there are two options, simply pressing F changes the setting from **CP/M** to **WORDSTAR**, and vice versa.

We've now introduced you to all the SETUP options. To save the settings you selected, press RETURN and you will leave the configuration menu. Next press A, B , or the RETURN key to save the new "setup" on the diskette in drive A, drive B, or M to save it in memory until the computer is reset or turned OFF. If you've changed any setting and wish to save it, press A, as that is the diskette from which you got the system you have made the changes to.

One final note about SETUP: SETUP does not change anything except in the operating system of the diskette being configured. In other words, if you add function keys and now expect them to work, they won't, at least not until you restart the computer using the diskette on which you saved your new setup. In other words, to start using your new settings, wait for the disk activity light to go OFF and then push the **RESET** button to restart the computer. Now put the diskette that you changed into the A drive—if it isn't already there—and press RETURN. Your new settings should then be in effect.

Certified Computer User

If you've made it this far without too many false starts, you're a certified computer user. In these few pages you've encountered basic concepts and procedures with which to use your Osborne 1 computer.

The above statement may surprise you. Although a computer is an extremely complex device internally, you don't need a degree in computer science to use it. The basic premise on which almost all the software you'll ever use on the Osborne 1 computer works is this:

1. The computer displays a message or prompt.
2. You respond with an instruction or with information.
3. The computer processes your instruction or data and goes back to step 1.

What often makes computers difficult to understand is that the message or prompt is cryptic or ambiguous, or that you have to memorize the instructions or information you need to give the computer.

We've attempted to make the Osborne 1 as simple as possible to use. The messages and prompts it displays have gone through tests and evaluations to rid them of ambiguity and make them as clear-cut as possible. It is, unfortunately, impossible to account for every conceivable use of the Osborne 1 or to make the system entirely "idiot-proof." In fact, if we made the computer simple enough so that anyone could use it, you might complain that it was not as effective a tool for you as it could be. Nevertheless, if you find messages or instructions that are confusing, let us know so that we can correct the problem in future versions.

The Reference Guide contains listings of all the commands, error messages, and other specific information you need to use your Osborne 1 efficiently. Before you proceed with the next chapter, it may be wise to briefly thumb through the Reference Guide to see what information is there and how it is presented.

Incidentally, you'll notice that throughout this Guide, when we want you to achieve certain precise results we'll ask you to type very specific sequences, character by character, like this:

/ F C A RETURN

Often, character sets of this kind are too long to be typeset in a single line, so they are shown like this:

`S` `U` `M` `(` `(` `A` `1` `:` `A` `4` `)` `-` `A` `1` `0` `:` `A` `1` `2` `)` `*` `(` `A` `1` `1` `*` `P` `1` `)`
`+` `(` `A` `5` `:` `A` `8` `)` `)` `RETURN`

In all such cases, continue typing the characters *without pressing RETURN* until you get to an actual `RETURN` character, such as at the end of the final line, above.

In addition, we denote control characters in a sequence like this:

`^` `A` `K`

which means type a control A (`CTRL` `A`), then a `K`. We'll let you know if we use the `^` symbol to mean the "caret" above the 6 key.

CHAPTER 3

Learning and Using CP/M

In this chapter, we'll introduce you to the CP/M operating system that directs the activities of your Osborne 1. It is important that you understand the information presented in this chapter before you begin regular use of your computer.

Operating Systems

CP/M stands for "Control Program/Monitor." It is usually referred to as an "operating system." An operating system is a set of instructions that tells your computer how to use its various components.

As such, CP/M represents the basic intelligence of your Osborne 1 computer. The CP/M operating system is different from the CP/M System and Utility programs. The operating system occupies a reserved area of each program diskette supplied with your computer. This CP/M operating system "knows" how to send information to the printer, to and from an optional modem (a device used to communicate with remote equipment via telephone lines), and to and from the disk drives; knows how to get programs from the diskette and place them in memory for use; and performs other important tasks.

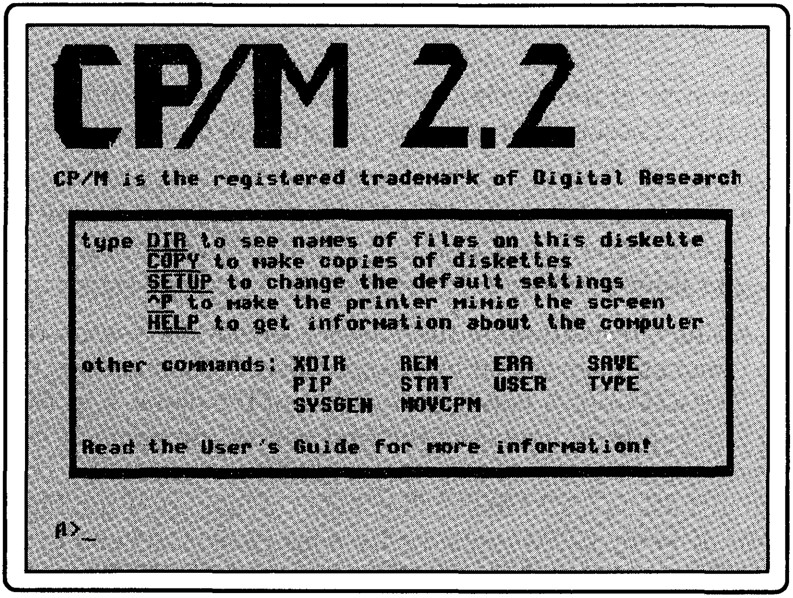
Most of the time, you probably won't notice CP/M's presence. The operating system loads automatically into the memory of your computer if you follow the instructions we gave you in the last chapter.

You might like to think of the CP/M operating system as an intelligent "engine" inside your computer. Like the engine in your car, you know it is there, but it is rare that you actually have to pay much attention to it. Like an auto's engine, CP/M does require you to remember a few important things in order to use it; you shouldn't put regular gas in a car that requires unleaded, for example, and CP/M similarly requires you to remember a few simple facts about it. If you need to transfer a copy of the CP/M operating system to a diskette, use the SYSGEN program described in the Reference Guide.

Lesson 1: File Names

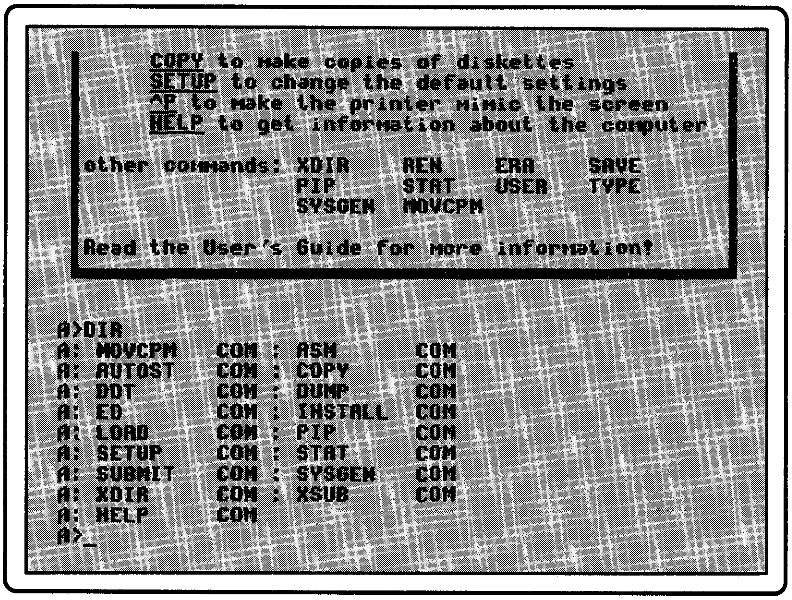
In chapter 1, we told you that all information on a diskette is stored in files. CP/M requires that each file have a unique name associated with it. Let's take a look at a diskette and find out what these file names are like:

1. Turn the power to your Osborne 1 **ON**.
2. Put the copy of the CP/M master diskette you made into the left-hand drive and press RETURN, as the message on the monitor screen requests.
3. After a few moments, the HELP program should appear. Press the ESC key to get to CP/M.
4. Across the top of your screen you should see the word CP/M, in large letters, and an informative message underneath. A couple of lines underneath that you'll see A > and an underline character. The underline character shows your "cursor" and indicates where the next information you enter from the keyboard will appear, as illustrated below:



```
CP/M 2.2

CP/M is the registered trademark of Digital Research

type DIR to see names of files on this diskette
     COPY to make copies of diskettes
     SETUP to change the default settings
     ^P to make the printer mimic the screen
     HELP to get information about the computer

other commands: XDIR    REN    ERA    SAVE
                PIP     STAT   USER   TYPE
                SYSGEN  MOVCPM

Read the User's Guide for more information!

A>_
```

5. Type D I R and press RETURN. The disk drive will activate for a moment and a list of files will appear on your screen:



```
     COPY to make copies of diskettes
     SETUP to change the default settings
     ^P to make the printer mimic the screen
     HELP to get information about the computer

other commands: XDIR    REN    ERA    SAVE
                PIP     STAT   USER   TYPE
                SYSGEN  MOVCPM

Read the User's Guide for more information!

A>DIR
A: MOVCPM   COM : ASM      COM
A: AUTOST   COM : COPY     COM
A: DDT      COM : DUMP     COM
A: ED       COM : INSTALL  COM
A: LOAD     COM : PIP      COM
A: SETUP    COM : STAT     COM
A: SUBMIT   COM : SYSGEN   COM
A: XDIR     COM : XSUB     COM
A: HELP     COM
A>_
```

You'll notice that the A > is back, again followed by your cursor (underline).

Look carefully at the display on your monitor screen; you should notice certain things about the information presented on it.

- The letter "A" recurs. First, it always appears before you issue a command, and, second, it appears on each line of information.
- The information on the screen seems to be a series of names, each set off by a colon; also, sometimes large spaces appear between portions of the name, and this depends on how long the first part of the name is.

You've just asked CP/M for a "directory" (a listing) of the files on your diskette. CP/M replied with a list of files, several to a line, and then returned for another command (the A >).

File-naming conventions in CP/M have three basic components:

$$\underbrace{\mathbf{A\ :}}\ \underbrace{\mathbf{F\ I\ L\ E\ N\ A\ M\ E}}\ \underbrace{\mathbf{.\ T\ Y\ P}}$$

1. a letter and colon to represent the drive	2. the actual file name	3. a period and 3 characters for the "type" of file it is

1. A letter and colon to represent the disk drive in use: CP/M considers the first drive (the one on the left) to be drive A, the second to be drive B. A colon separates the disk drive "identifier" from the next part of the file name.
2. A one- to eight-letter file name: The file name may consist of any combination of letters and numbers. You can also use a few symbols (such as +), but it is best to avoid these, as many are not legal:

   < > . , ; : = ? * [ ] illegal characters in file name

   CP/M does not distinguish between uppercase and lowercase file names; thus **FILE** would be the same as **file** to CP/M.
3. A zero- to three-letter file type may follow a period (the period used to separate the file name from the file type).

The directory you just received of the files on your diskette follows the above rules. The letter that precedes each line tells you the disk drive the files are on, a file name follows, separated from its file type by spaces, separated from the next file name by a vertical line.

You may wonder what different kinds of files there are.

CP/M predefines some common types of files. "**COM**" is the most important of these file types. Any file that resides in a file of this type is assumed to be a program or a command. If you enter its file name, leaving off the type, CP/M will load and execute the instructions in that program file. Try issuing the following CP/M command:

> [S] [T] [A] [T] and press [RETURN]

If you're still using the copy of the CP/M master diskette, you'll find on it a file named **STAT.COM** whose instructions the Osborne 1 will load into memory and follow. After a few moments of disk activity, a message should appear on the monitor screen telling you how much space is available on your diskette. We'll return to STAT later; for now, you should have learned that typing the file name of any COM-type file will cause the computer to load and execute the instructions.

Other file types commonly used in CP/M include the following:

.ASM	assembly-language source code
.BAK	a backup, or copy, of a file
.BAS	a BASIC program source file
.CAL	a SuperCalc data file
.DAT	a data file
.DOC	a text or document file
.INT	a CBASIC2 program file
.TXT	a text or document file
.$$$	a temporary file, or a file not properly completed

As you learn about WordStar, SuperCalc, CBASIC2, and Microsoft BASIC, we'll tell you more about the file types these programs use.

Lesson 2: Disk Drives

We already mentioned that CP/M uses letters to represent the disk drives. The left-hand drive on your Osborne 1 is known as drive A and the right-hand one is known as drive B.

So that CP/M knows when you are referring to a drive and when you're typing a file name or command, a colon always comes after the drive letter. If you enter:

B:FILE and press RETURN

the computer will attempt to find **FILE.COM** on the right-hand drive and load its instructions into memory for execution. Try issuing the above command now.

Whoops! We didn't tell you to put a diskette into drive B, so the disk-drive activity light will come on and stay on. The only way to recover from this error is to push RESET. Do so now and restart CP/M. We had you make this mistake intentionally. The only legitimate way to recover from such an error is to push the RESET button. Do not succumb to temptation and slide a diskette into drive B while the disk activity light is on.

Return to CP/M by pressing ESC after HELP identifies itself; you should see the A > again.

That A > is a meaningful prompt. The **A** tells you that CP/M will attempt to do everything on drive A unless you instruct it otherwise; the A > is simply a separator to indicate that your instructions to CP/M will appear immediately to the right of it. An A > prompt means that CP/M considers the left-hand drive, the A drive, the "default" drive.

You can make B the default drive. Take another of the diskettes you copied in the last chapter and place it in the right-hand drive. Issue:

B: and press RETURN

You should see that the right-hand drive is accessed for a moment (a check to determine if there is a diskette there) and then see a B > appear. Drive B is now the default, and any instructions you give CP/M will assume that you are dealing with the files in drive B. Try issuing the directory command (DIR, remember?). You should receive a different set of file names.

Now issue A: and press RETURN to set the left-hand drive as the default once again.

Lesson 3: Some CP/M Commands

CP/M has only seven built-in commands, and you've already learned two of them. In addition to the commands, CP/M has a few control character commands about which you might want to learn.

You already know the commands for changing disk drives and for displaying a directory:

A: establishes left-hand drive as the default,
B: establishes right-hand drive as the default,
DIR displays a list of file names on the monitor screen.

Let's look at each of the other commands, one at a time.

Command #3: ERA filename.typ

To erase a file from a diskette, you issue **ERA** followed by a space and the name and type of the file you wish to delete. If the file is not on the default drive, you must also specify the drive it is on.

Since there is nothing on your drive that you should wish to erase, let's put something there that you can erase. Type:

S A V E SPACE BAR 0 SPACE BAR

J U N K . F I L RETURN

All we're doing here is saving a blank file named **JUNK** whose type is **FIL**.

Ask for a directory of the diskette using the D I R command you learned earlier. You should see **JUNK.FIL** somewhere at the bottom of the directory. It is good to develop the habit of asking for a directory of the diskette before you erase any file on it, as CP/M erases files without asking you for verification.

Okay, now type:

[E] [R] [A] [SPACE BAR]

[J] [U] [N] [K] [.] [F] [I] [L] and press [RETURN]

The disk will activate and in a few moments your prompt will be back again. CP/M issues no message to verify that a file has been erased. You must ask for another directory to find out if CP/M actually deleted the file. Do so now with [D] [I] [R] [RETURN].

Command #4: REN newfile.typ=oldfile.typ

To rename a file you use the **REN** command. You'll note that the new name goes on the left and the old name goes on the right of the equal sign. Let's rename a file, issue:

[R] [E] [N] [SPACE BAR] [S] [T] [A] [T] [U] [S] [.] [C] [O] [M] [=]

[S] [T] [A] [T] [.] [C] [O] [M] and press [RETURN]

You should remember the STAT file. Since it gives you a message regarding the status of the diskette, we've just changed its name to **STATUS**. Don't forget to include the file types when you rename files.

STATUS.COM is the same as what used to be STAT.COM; only the name has changed. You can, if you desire, change the names of the files we supply with your Osborne 1. If STATUS makes more sense to you than does STAT, leave the name changed and make a note of the change in the Reference Guide. If you prefer to leave your files the same as they come, type:

[R] [E] [N] [SPACE BAR] [S] [T] [A] [T] [.] [C] [O] [M] [=]

[S] [T] [A] [T] [U] [S] [.] [C] [O] [M] and press [RETURN]

to restore the original name to the file you just used.

You can change a name of a file on the B drive by prefacing the file name with the B: disk drive identifier. The following command is invalid, however, since you can't rename a file from one drive to another:

REN B:STATUS.COM=A:STAT.COM RETURN

invalid! ←←←←←←←

Command #5: TYPE filename.typ

You can display files containing characters that can be displayed (data, as opposed to computer instructions) on the monitor screen by using the **TYPE** command. Since all the files on your CP/M diskette contain only instructions, put the copy you made of your WordStar diskette into the right-hand drive and issue:

T Y P E SPACE BAR B :

W S M S G S . O V R and press RETURN

You should see a sentence or two that identifies the contents of the file. Generally, you don't need this command unless you're trying to verify the contents of a file.

Command #6: USER n
Command #7: SAVE n filename.typ

These two commands constitute advanced topics, so we'll deal with them in a later chapter.

Lesson 4: Ambiguity and CP/M

Sometimes you may want to refer to a group of files at one time, rather than to a single file. CP/M identifies files in two ways without your having to enter the exact file name and its type. Both methods are called "ambiguous file references" or "CP/M wildcard specifiers."

First, you can use an asterisk (*) to substitute for either the entire file name or the entire file type, as in

ERA *.BAS

or

ERA PROGRAM.*

A second method of substitution involves the question mark (?). CP/M interprets the question mark as an "I don't care what's in this position" character. For example:

ERA PROGRA?.BAS RETURN

erases all files that have file names seven letters long, have **PROGRA** as the first six characters, and have the **BAS** file type.

Try issuing each of the following on your Osborne 1 and note what happens:

```
D I R RETURN
D I R SPACE BAR * . C O M RETURN
D I R SPACE BAR * . B A S RETURN
D I R SPACE BAR
C ? ? ? ? ? ? ? . * RETURN
D I R SPACE BAR * . C ? M RETURN
```

The first of these commands gives you a directory of the entire diskette. The second presents a directory of all COM-type files. The third presents a directory of all BAS-type files. The fourth displays a directory of all files whose names begin with C. The last displays a directory of all file types starting with C and ending with M.

You may use the asterisk and the question mark with the DIR and ERA commands, but not with the others. Later you will learn about the PIP and STAT programs, and these, too, can recognize the ambiguous file references you've just learned.

Lesson 5: Control-Character Commands

Control characters are something like capital letters on computer keyboards, but instead of holding down the SHIFT key to create them, you hold down the control key (marked CTRL). The best way to accustom yourself to typing control characters is to first hold down the control key, then press a letter, and finally let go of the control key. Eventually good typists are able to enter control characters as quickly as they can any other character.

We represent control characters in this manual by preceding the letter to be pressed with a small "caret" (^). CP/M and WordStar also use this symbol to indicate a control character.

^C = control C
^P = control P
^S = control S

Whenever you see the CP/M prompt, you may issue any of the above characters.

^C causes CP/M to restart. The practical effect of restarting is that it causes CP/M to "forget" any information it has learned about a diskette and start from scratch. If this doesn't make any sense to you, consider what would happen if you switched diskettes in a drive without telling CP/M. If it expected a certain diskette in the drive when you had replaced it with another, it might not know where to put information onto that diskette and might accidentally erase something you wanted to save.

Fortunately, CP/M can detect if a diskette is changed. Unfortunately, CP/M displays an error message if it detects that you changed diskettes:

```
BDOS ERR ON A: R/O
```

The above message stands for "disk error on drive A:; I can only read from that diskette, not write anything onto it."

So, anytime you change a diskette when the prompt indicates that CP/M is waiting for a command, issue a **^C** to tell CP/M that you've just changed diskettes. Try that right now with one of the diskettes you created with the COPY program in the last chapter:

1. put a different diskette in drive A;
2. issue a ^ C; and
3. Type a D I R and press RETURN to see what's on that diskette.

NOTE

When you change diskettes while you're running a program, what you have to do to tell the computer you've changed diskettes varies from program to program. WordStar, for example, requires you to issue a CHANGE LOGGED DISK command. This is one reason why reading the manuals that come with a program is vital.

Put your CP/M system diskette back in drive A.

Another control character, **^P**, "toggles" the printer. CP/M usually sends characters only to the monitor screen. If you have correctly connected a printer to your Osborne 1 and have first run the printer SETUP program (described in chapter 2), issuing a ^ P will tell CP/M to now send all characters to both the monitor screen and the printer. Issuing ^ P again turns off the stream of characters to the printer. In other words, if the printer is mimicking the screen, issuing **^P** turns the printer OFF. You can visually determine if this feature is engaged by the way characters appear to stream from left to right rather than all at once on the screen. By the way, if you issue **^P** and nothing prints, be sure the printer is turned ON and correctly plugged in.

The last control character described in this chapter is control-S. Issuing **^S** temporarily makes the Osborne 1 computer pause. Issuing it causes anything being displayed on the screen to freeze. Issuing another **^S** (any character, actually) restarts the computer. This command is helpful if a lot of information is being displayed which begins "scrolling" off the top of the screen; issue **^S** to make the computer pause in such a circumstance.

Lesson 6: PIP and STAT

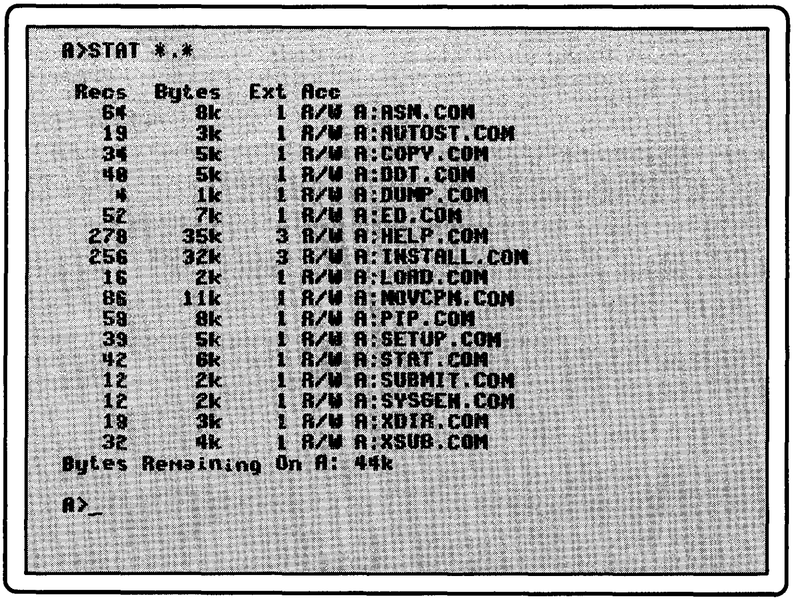
Two helpful programs included with CP/M are PIP (which stands for Peripheral Interchange Program, a fancy way of saying "copy") and STAT. Both of these programs are stored on the CP/M diskette as COM-type files, so you execute them by typing their file name.

Both programs have several options you'll want to know. Let's start with STAT.

Type:

[S] [T] [A] [T] [SPACE BAR] [*] [.] [*] [RETURN]

and you should soon see a list of files—it may not match the one below—in alphabetical order, and some other information about those files.



```
A>STAT *.*

 Recs  Bytes  Ext Acc
   64     8k    1 R/W A:ASM.COM
   19     3k    1 R/W A:AUTOST.COM
   34     5k    1 R/W A:COPY.COM
   40     5k    1 R/W A:DDT.COM
    4     1k    1 R/W A:DUMP.COM
   52     7k    1 R/W A:ED.COM
  278    35k    3 R/W A:HELP.COM
  256    32k    3 R/W A:INSTALL.COM
   16     2k    1 R/W A:LOAD.COM
   86    11k    1 R/W A:MOVCPM.COM
   58     8k    1 R/W A:PIP.COM
   39     5k    1 R/W A:SETUP.COM
   42     6k    1 R/W A:STAT.COM
   12     2k    1 R/W A:SUBMIT.COM
   12     2k    1 R/W A:SYSGEN.COM
   18     3k    1 R/W A:XDIR.COM
   32     4k    1 R/W A:XSUB.COM
Bytes Remaining On A: 44k

A>_
```

Recs stands for the number of 128-character "blocks" (called records) the file in question occupies. This is an internal piece of "housekeeping" information that most users don't need to know.

Bytes represents the length of the file in kilobytes, each kilobyte being 1024 characters long. A byte is equivalent to one character of information. The Osborne 1 can store as much as 185K bytes of information on each double-density CP/M diskette.

Ext represents the number of physical extents occupied by the file. This is another internal housekeeping item and shouldn't concern you unless you begin learning to program.

Acc stands for the file access attribute. **R/W** indicates "read/write," which means you can read from the file and write new information to it. Sometimes you might encounter **R/O** in place of **R/W**. **R/O** stands for "read/only," meaning that no new information can be written into the file. Immediately to the right of the access attribute are the disk drive, the file name, and the file type.

STAT functions as a fancy directory, one that gives you some extra information about each file on the disk. Try each of the following commands:

```
S T A T SPACE BAR
D ? ? ? ? ? ? ? . * RETURN
S T A T SPACE BAR * . C O M RETURN
S T A T SPACE BAR
? U ? ? ? ? ? ? . * RETURN
```

Now try the following:

```
D I R SPACE BAR
D ? ? ? ? ? ? ? . * RETURN
D I R SPACE BAR * . C O M RETURN
D I R SPACE BAR
? U ? ? ? ? ? ? . * RETURN
```

You should note that DIR and STAT work essentially the same way; only the information they present differs. We'll cover other uses of STAT later in this manual; but most users will find the above functions more than enough for now.

NOTE

We've included another special program named ***XDIR*** *on all the diskettes we provide. XDIR performs the same as STAT. If all you want to do is find out what is on a diskette and how much room it takes up, use XDIR in place of STAT. We placed XDIR on each diskette instead of STAT because XDIR takes up less space on a diskette, leaving more room for your files. Later in this manual you'll find out about other uses for STAT, uses XDIR does not duplicate.*

Now we move on to PIP, a program that has multiple uses. For now, we'll study one: how to copy individual files—as opposed to entire diskettes—between diskettes. We'll quickly cover the other uses of PIP later.

Issue:

P I P and press RETURN

After a few moments, you should see an asterisk (*), which indicates that PIP is waiting for further commands. PIP commands are structured much like the REN (rename) command:

newfile.typ=oldfile.typ

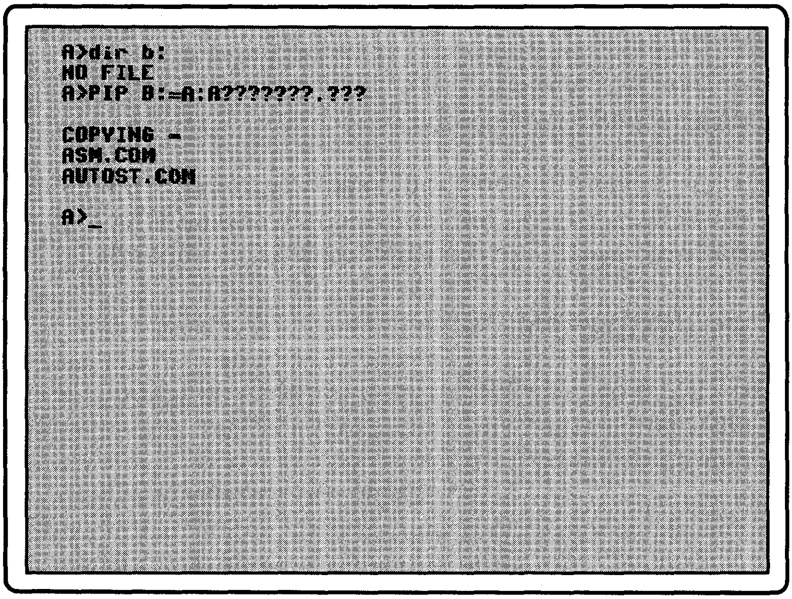
Unlike REN, however, PIP lets you use the ambiguous file references you learned earlier. Let's try PIP out; press the **Reset** button and follow the instructions below:

1. Take a blank, unformatted diskette and put it into drive B. Your system diskette should be in drive A.
2. Format the blank diskette as you learned in the second chapter, but do not use the COPY program to duplicate the system diskette. (In other words, with the CP/M system diskette in the left drive and the A > appearing on the screen, type C O P Y and RETURN, then F, and B, then RETURN and finally D for double density. Get back to CPM; press RETURN twice.
3. Type D I R SPACE BAR B : and press RETURN to see the directory of files on drive B. You should get the NO FILE message.
4. Type P I P SPACE BAR B : = A :

   A ? ? ? ? ? ? ? . ? ? ? RETURN

   (We're being sneaky here. First, we just showed you a direct way to issue PIP commands that skips the PIP prompt, an asterisk. Second, we abbreviated the "**newfile.typ**" to "**B:**"—which means that PIP should copy whatever fulfills the right side of the equation onto drive B, using the same file names.)

5. You should see something like the following:



```
A>dir b:
NO FILE
A>PIP B:=A:A???????.???

COPYING -
ASM.COM
AUTOST.COM

A>_
```

6. Try the following sequence of other PIP commands and see what they do. (Hint: use **DIR B:** or **XDIR B:***.* to see what was copied onto drive B.):

[P] [I] [P] [RETURN] ← initiates PIP

with the * prompt present on the screen, type:

[B] [:] [=] [A] [:] [*] [.] [C] [O] [M]

[RETURN] ← valid PIP sequence

[^] [C] ← remember control-C?

[X] [D] [I] [R] [SPACE BAR] [B] [:] [RETURN] ← examine disk in B

[E] [R] [A] [SPACE BAR] [B] [:]

* . * RETURN ← note the message ERA gives and press Y and RETURN

P I P SPACE BAR

B : = A : D ? ? ? . C O M RETURN

X D I R SPACE BAR B : RETURN ← examine disk in B

Lesson 7: Error Messages

Since we can't always be perfect, it is a good idea to be aware of the error messages CP/M may present and what they mean. Rather than lead you through a series of errors (we don't want you to learn any bad habits), we'll just describe each error message and what it means. If it takes you a while to make an error, just remember to look here to find out what CP/M was trying to tell you.

?

Any sequence of characters followed by a question mark indicates that CP/M looked on the disk for a file of the type **COM** with those characters as the file name and couldn't find it. Check the directory of the diskette to make sure that the file you thought was there really is.

NO FILE

Indicates that no files in the directory matched what you specified in your command. If you issue D I R and RETURN, then receive this message, no files at all are present on the diskette.

BDOS ERR ON x:

Appears when CP/M cannot find a diskette in the drive (A: or B:) you specified, the diskette hasn't been formatted, or the drive door is not properly closed. If the words **BAD SECTOR** immediately follow the message, then you either have a bad or an improperly formatted diskette. If the abbreviation **R/O** appears, you may have forgotten to issue a ^C when you changed diskettes, or you told CP/M at some point not to allow anything to be added to or deleted from that file.

FILE EXISTS You tried to rename a file with a name that already existed. Since you can't have two files on the same diskette with the same name, try using a different name.

Time to Move On

At this point, that's all you really have to know about CP/M. In the next chapters, we'll concentrate on teaching you how to use WordStar and SuperCalc, the two primary software programs that accompany your Osborne 1 computer.

CP/M has more to offer, and we'll return to it in the advanced sections of this manual. You might also want to look at some of the books that have appeared on CP/M, most notably *The CP/M Primer*, published by Howard Sams, and *The Osborne CP/M User Guide*, published by Osborne/McGraw-Hill.

CHAPTER 4

WordStar and MailMerge

This chapter will take you step by step through the procedures of creating, formatting, storing, and printing documents using WordStar and its companion program, MailMerge. The chapter explains frequently used features in detail, while touching briefly on those used less often.

What Is WordStar and What Does It Do?

WordStar is a program that enables the Osborne 1 to function as an efficient word-processing system. The term "word processing" aptly applies to the process of manipulating text (letters, reports, books, and the like) with a computer.

WordStar word processing lets you enter text at the keyboard just as you would on a typewriter. In actuality, the keyboard is one of the few design features held over from the antiquated typewriter technology. As you read on, it will become obvious why word processing makes typewriters seem cumbersome and archaic.

To begin with, information you enter at the keyboard is immediately displayed on the monitor, where you can move, change, or delete it with easy-to-learn commands. When you first begin using WordStar, margins and line spacing are preset to the most common settings, but can be altered at will.

You can compose whole paragraphs of text without using the carriage return; when you reach the right-hand margin and WordStar detects a word that is too long to fit on the current line, it automatically moves the word to the next line.

Once you've entered a rough draft of your document, you can rearrange and edit its parts until you are satisfied with its accuracy and organization. As you become proficient in using WordStar, you will learn how to locate a specific word or phrase; optionally replacing it with another, manipulating sections of text, and reconstructing fragmented paragraphs. A menu at the top of the screen aids you in learning these more complex commands.

Special-effect commands can further enhance your document. For example, you can specify underlines, boldface, subscripts, superscripts, and specific formatting that will appear in the printed version of the document. Furthermore, the MailMerge feature lets you merge text and/or data files together, thereby generating form letters, boilerplate text (text created from files of preexisting, commonly used sections of text), mailing lists, and large documents. In fact, this entire manual was written using WordStar.

After you have prepared a document, you store it as a file on a diskette. Your text will remain safe and securely stored on the diskette until you want to edit, print, or send the file to another computer over the phone lines.

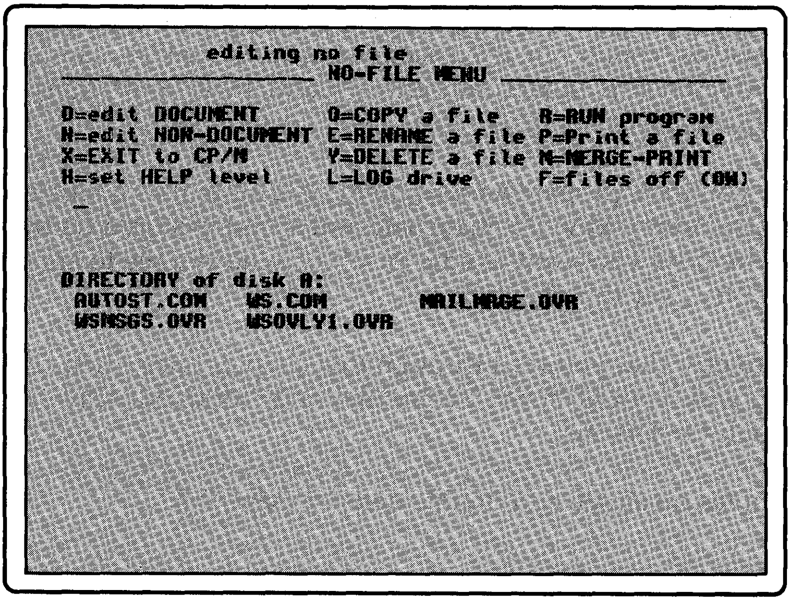
We suggest that you read through this chapter once, then sit down with your Osborne 1 computer and work your way through the lessons.

See notes on single and double density, page 758.

Starting WordStar

The process of loading WordStar is simple.

1. Press the **RESET** button (unless you've just turned the Osborne 1 ON).
2. Put a copy of the WordStar diskette you made in chapter 2 into drive A and secure it by pulling the door down over the diskette.
3. Place a blank, formatted diskette in drive B and secure it. You will use this diskette to store the documents you create using WordStar. If you wish to edit any document files that you created in an earlier WordStar session, you should put the diskette on which those files reside into drive B.
4. Press the RETURN key in response to the sign-on message displayed on the monitor. (Typing WS from the A> prompt may also be used to start WordStar.) After a few moments, you'll see the Osborne logo, then the WordStar sign-on message, and finally the no-file menu will appear:



```
                 editing no file
____________________ NO-FILE MENU ____________________

D=edit DOCUMENT      O=COPY a file   R=RUN program
N=edit NON-DOCUMENT  E=RENAME a file P=Print a file
X=EXIT to CP/M       Y=DELETE a file M=MERGE-PRINT
H=set HELP level     L=LOG drive     F=files off (ON)
 _


DIRECTORY of disk A:
 AUTOST.COM   WS.COM         MAILMRGE.OVR
 WSMSGS.OVR   WSOVLY1.OVR
```

You are now using WordStar.

Lesson 1: Starting From the No-File Menu

The listing of command definitions at the top of your screen is the "no-file menu," called such because at this stage you are not working on any particular file. The no-file menu provides an introductory level where files are manipulated, and where preparations which affect these files are made. A single letter identifies each possible operation. You select one of the listed operations by pressing the letter that precedes it on the menu. The definition of each operation is fairly explicit, and as you read on, you will gradually understand the operations that can be performed from this menu. A detailed description of each operation and what it does can be found in the Reference Guide that accompanies this tutorial.

File directory

Immediately below the no-file menu you will notice a list of file names. This listing currently shows a directory of all the WordStar program files on the diskette in drive A. This directory is like the one you learned about in the last chapter (the CP/M command DIR), except that here files are arranged in alphabetical order. The no-file FILE DIRECTORY command F turns this directory OFF or ON again like a switch. You will encounter many such "toggle-switch" commands when using WordStar. For now, try pressing the letter F (FILE DIRECTORY) and watch the resultant action on the screen. Before proceeding, make sure that the file directory is once again displayed by pressing F again.

Changing Disk Drives

Before actually creating any documents using WordStar, you should get in the habit of telling the computer which disk drive to use for storing your documents. If you don't give WordStar any instruction to the contrary, it will use drive A to store and retrieve documents. Since the WordStar program diskette in drive A is already full of files, you should always store your files on drive B. A blank diskette placed in drive B will have the maximum capacity of 80 single-spaced, typed pages.

REMEMBER: You don't want to store your files on the WordStar program diskette in drive A. So, place a formatted disk in drive B, and issue the no-file CHANGE LOGGED DRIVE command L to switch control to drive B. This message will appear:

```
THE LOGGED DISK DRIVE IS NOW A:

NEW LOGGED DISK DRIVE (letter, colon, RETURN)?
```

The "logged drive" means the drive currently being used to store and retrieve information. To "log" the B drive so information is recorded on your storage diskette, enter B : and press the RETURN key. The A drive will whirr a moment, and then the B drive will become active, as indicated by its activity light.

Notice that the file directory, which used to list the files on drive A, has been replaced by a new directory of the files on drive B. If this is the first time you've used your storage diskette, there aren't any files on it, so nothing is listed in the directory.

Creating A File

By now you are probably anxious to begin word processing. Issue the no-file CREATE OR EDIT A DOCUMENT command; press [D] to begin creation of a document file.

NOTE

WordStar distinguishes between files it calls "documents" and those it calls "non-documents." The difference between the two is simple: a document file is formatted according to your instructions (margins, headings, and other special instructions), while a non-document file contains only characters exactly as you entered them. You create document files if your primary need is to create letters, memos, reports, or other documents using WordStar. You create non-document files if you are editing programs or program data using WordStar.

Whenever you ask WordStar to create a document file, some guidelines on creating a file appear on the screen and WordStar asks:

```
NAME OF FILE TO EDIT?
```

This prompt (WordStar question) may seem confusing since you probably think that you want to create, not edit, a file. Actually, WordStar doesn't pay much attention to the difference. If the file you name doesn't already exist on the diskette in the logged drive, WordStar creates a new file under the specified name, which will contain the document you generate. If the file you name already exists on the logged diskette, WordStar loads the first portion of that file into the memory of the Osborne 1 and allows you to begin editing it.

The name you assign a file will subsequently be used to recall the file, so make sure that the name you pick is meaningful to you. Remember, a file name consists of up to eight letters or digits, a period, and three optional characters called the file type. For example, enter:

TRIAL.TXT

as the name of your file; don't press RETURN yet! Notice that this is a valid name, and gives a clear indication of the file's contents: trial text. A file named **ARC08T.V01** gives little indication of what's in the file.

Now let's experiment with the control commands that are listed just above the prompt that requests a file name. Once you leave the no-file menu, WordStar uses control characters—referred to as "commands"—to handle all word-processing operations. To issue a WordStar command, simply depress the CTRL (^) key while simultaneously pressing the appropriate letter to issue a command.

The command **^S** deletes the last letter in the file name **TRIAL.TXT**, and **^D** restores it. Try that right now. Press ^S, then ^D, and make sure that WordStar behaves in the manner you expect. A ^Y deletes the entire file name and a ^R restores it. Try each command now, first erasing the file name you entered, then restoring it.

In passing, we'll note that the arrow keys on your Osborne 1 perform the following functions with WordStar:

→ **is the same as ^D**

← **is the same as ^S**

↑ **is the same as ^E**

↓ **is the same as ^X**

When you're actually editing a document, the arrow keys move the cursor as you will see. We'll return to our discussion of the arrow keys when we talk about moving the cursor.

Use **^U**, the INTERRUPT command, whenever you wish to abort a command currently in progress. What's the command in progress right now, you ask? Since you're not typing anything at the moment, the last command that is still in effect is the one in which you asked to edit a document file. In this case, then, the INTERRUPT command (**^U**) would abort the creation of the file named **TRIAL.TXT**, causing you to start again from the no-file menu. Issue ^U and observe that you receive this message:

```
* * * INTERRUPTED * * * Press ESCAPE Key
```

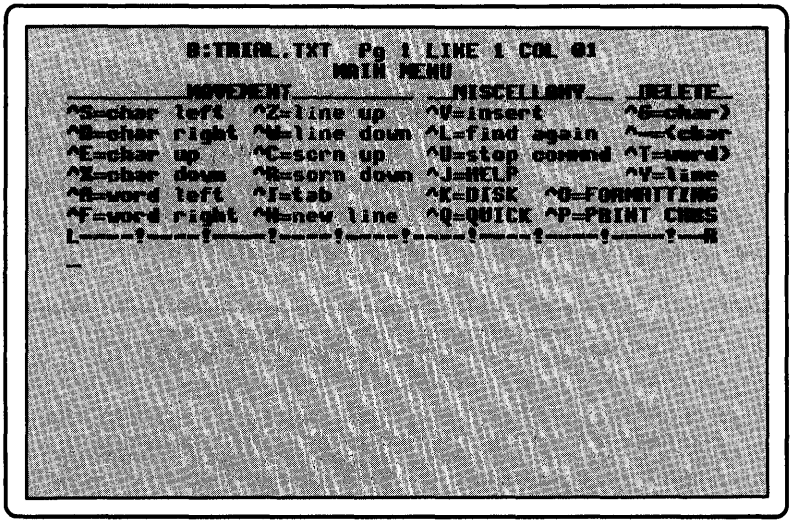
Press the ESC ("escape") key on your Osborne 1 as advised in the message, and you will return to the no-file menu. You just aborted the CREATE OR EDIT A DOCUMENT command. Anytime you mistakenly issue a command, use the INTERRUPT command (**^U**) as you just did.

Again issue the CREATE A FILE command by pressing the D key, and when WordStar asks you for the name of the file to edit, once again enter:

TRIAL.TXT

Make sure the file name **TRIAL.TXT** is correctly entered on the prompt line, then press the RETURN key. The message NEW FILE will appear at the left edge of the screen just below where you last typed. WordStar automatically creates the new file for you. If a file named **TRIAL.TXT** already existed on the storage diskette in drive B, then WordStar would have accessed it (computer term for "opening a file"), retrieved the information from it, and then displayed the text on the screen so you could edit it. Creating a WordStar document file the first time is comparable to clearing your desk and getting out a pad of paper and a pencil. Editing a file that already exists is like taking out a pad of paper that already has writing on it.

Your screen should now look like this:



```
            B:TRIAL.TXT  Pg 1 LINE 1 COL 01
                        MAIN MENU
_________MOVEMENT______________  __MISCELLANY__ _DELETE_
^S=char left  ^Z=line up    ^V=insert        ^G=char>
^D=char right ^W=line down  ^L=find again    ^-=<char
^E=char up    ^C=scrn up    ^U=stop command  ^T=word>
^X=char down  ^R=scrn down  ^J=HELP          ^Y=line
^A=word left  ^I=tab        ^K=DISK   ^O=FORMATTING
^F=word right ^N=new line   ^Q=QUICK  ^P=PRINT CHARS
L----!----!-----!----!----!----!----!----!-----!----R

_
```

At last you are ready to begin word processing. A menu of commands in the upper portion of the screen shows the most-often-used WordStar commands. Five other such menus assist you in learning specific other parts of the WordStar word-processing system.

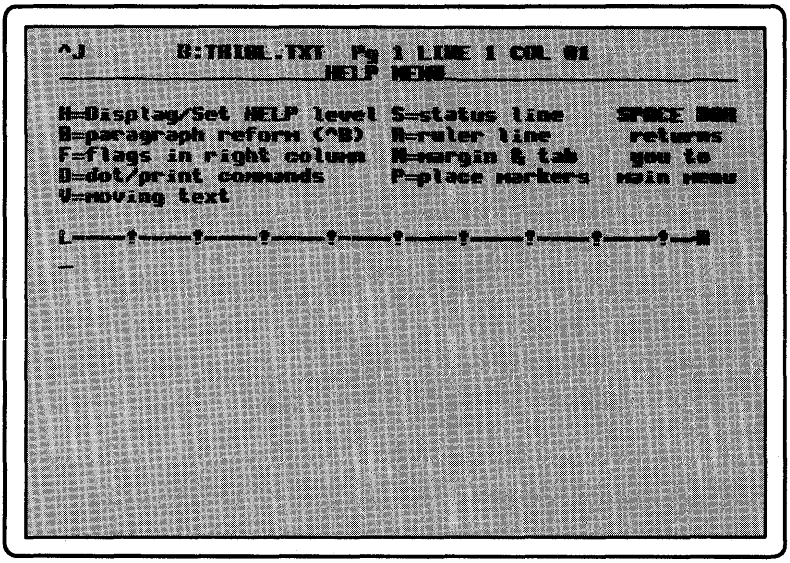
The commands at the top of the screen are all single-letter commands, which you use by holding down the CTRL key and pressing the letter corresponding to the action you desire.

Look at the menu, and you'll see a number of commands that move the cursor. Many higher-level WordStar commands consist of a sequence of control letters. The first letter accesses a specific catagory and the next letter selects the command from within that category. The letters **Q**, **J**, **K**, **O**, **P** function in this manner and thus serve as a command "prefix." Issuing one of these prefix commands causes a menu corresponding to its related category to appear. Let's investigate these two-letter commands.

Lesson 2: Getting Help When Needed

The most useful prefix command for beginners is the **HELP** prefix **^J**. Issue ^J now.

You will see a different menu listing several subjects, each preceded by a letter and an equal sign. In the upper left-hand corner of the screen, is a **^J**. This indicates that you've pressed the prefix J and selected the J HELP menu. WordStar is waiting for you to select from this new menu of options.



```
^J        B:TRIAL.TXT  Pg 1 LINE 1 COL 01
___________________________HELP MENU____________________________
H=Display/Set HELP level S=status line     SPACE BAR
B=paragraph reform (^B)  R=ruler line       returns
F=flags in right column  M=margin & tab     you to
D=dot/print commands     P=place markers   main menu
V=moving text

L-----!-----!-----!-----!-----!-----!-----!-----!-----!--R
_
```

The first command listed on the HELP menu is the letter H. This HELP command changes the degree of assistance provided. Pressing the letter H from the HELP menu shows what level of help is currently in effect and allows you to change it.

You have four levels of assistance to choose from (0–3). The HELP setting is usually 3, but as you gain experience using WordStar, you may reduce the amount of information the menus show and eventually eliminate them by choosing lower numbers. Actually, except for the main menu, you can avoid seeing any of the other menus by issuing your commands in rapid succession. When you issue a command prefix, WordStar waits just a moment before displaying your options. If you press a response before WordStar begins displaying these options, you'll "beat the computer" and not see any additional messages. The WordStar HELP system is designed to be there when you need it, not to intrude on everything you do.

Examine the different HELP levels with ^JH (0–3) if you like, but since you are just starting out, leave the HELP level set to 3. We'll return to each subject listed on the HELP menu in its proper context, so don't think you're missing anything.

When you return to the main menu, look at the top of the WordStar screen. You'll see another helpful feature of WordStar, the "status line."

```
B:TRIAL.TXT   PAGE 1  LINE 2  COL 7     INSERT ON
```

is the way the status line should read at this point. Don't worry if the column or line number is different if you've moved the cursor around more than we suggested you do.

Observe that the command currently in progress, if any, appears at the far left of the status line, followed by the name of the file you are working on. The current page number, line, and column are also shown. Also the message INSERT ON should be visible at the right-hand side of the status line; you will need to use the control arrow keys (← and →) to see this message. As described later, **INSERT ON** indicates that WordStar will "insert" text rather than replace it.

To reinforce what you've just learned about the status line, let's have WordStar tell you all about it. You remember the HELP command, right? Issue a ^J, and the HELP menu should appear in a moment at the top of your screen. If you read the menu carefully, you'll find that you have to press S to get more information about the status line. Do so now. A screen full of information should appear for you to read. Press the SPACE BAR when you've read the screen, and another full screen will appear. Continue doing so until WordStar returns you to the main menu and the text you entered earlier.

Lesson 3: Entering Text

Take a minute now to enter a few lines of text. To do so, first position the cursor at the end of the information you've already typed by issuing ^F (for "next word") several times. Remember that the cursor (the bright underline) always appears at the position where the next letter will appear or the next action will occur.

Watch the column number on the status line as you type. Use the left arrow key (← without control) to backspace a single-character position. If you want to erase characters as you backspace, use ^- (control-hyphen) instead of the left arrow. (Note: On the Osborne 1, control-hyphen is the same as what WordStar calls the **DEL** ("delete") key.)

Even though the Osborne 1 displays just 52 characters per line, it allows lines up to 128 characters (letters or numbers) wide. The 52-character display is a window on a longer line. When you enter text beyond the 52nd column, the window automatically shifts to the right, thus extending your view of lines that are longer than the screen.

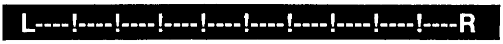
Now let's explore some of the special word-processing functions WordStar incorporates:

WORD WRAP: Normally when you have typed 65 characters, WordStar lets you finish the word you are typing and then jumps to the next line. If possible, the last word you typed will be fit within the 65 columns on the previous line; otherwise the word moves to the beginning of next line. This feature, termed "word wrap," means that you never have to use the carriage-return key except when you wish to force WordStar to move to the next line, as at the end of a paragraph.

You can turn the word-wrap feature OFF and enter text just as you would on a conventional typewriter with the **^OW** command, but it is rare that anyone would want to do this. Issue ^O and look closely at the O-prefix menu. Notice that the W option (for word wrap) tells you whether it is ON or OFF. Right now it should be ON. Press ESC to return to the main menu.

JUSTIFICATION: Have you noticed that the words on a line are spaced unevenly and that all text lines end in the same column? Text is aligned at the right margin by a WordStar feature called "justification." Justification purposely spaces the words to achieve this symmetry. If you want text to appear without being aligned at the right margin, turn the justification OFF with the command **^OJ**. You can also use hyphens to improve the appearance of your text, so don't be afraid to hyphenate a long word if it seems to be going past the right-hand margin.

RULER LINE: A line that appears just above the first line of your text is called the "ruler line":



```
L----!----!----!----!----!----!----!----!----!----R
```

The **L** marks the left-hand margin setting, the **R** the right. Each exclamation point marks a regular tab stop, while a number sign signifies decimal tabs. WordStar margins can be set anywhere between 1 and 240; we'll show you how to set margins and tabs later.

HYPHENS: WordStar has two kinds of hyphens: the "soft hyphen" is used to indicate a syllable break that only appears in your printed text if the break occurs at the end of a line; a "hard hyphen" separates words and phrases no matter where it ends up, and always appears in your printed text.

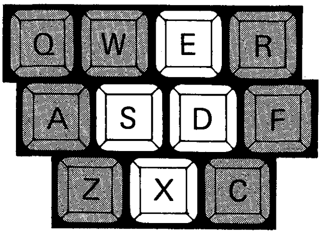
Hyphens are automatically soft because the soft-hyphen feature is automatically ON. As with most commands that concern the format or look of a document, the status of the soft-hyphen feature is shown on the **^O** menu. You turn the soft-hyphen entry ON or OFF with a combination of **^O** and **E**. You can tell if the soft-hyphen feature is ON or not by issuing a ^$\boxed{\text{O}}$ and looking at the menu.

Use hard hyphens whenever a fixed divider is required between characters, words, or phrases. Hyphens are always hard when soft-hyphen entry is OFF, but you can enter hard hyphens even though soft-hyphen entry is ON by issuing **^P** before typing the hyphen.

You distinguish between hard and soft hyphens when you review text on the monitor with the print-control display command, **^OD**. **^OD** turns the display of certain print-control characters OFF or ON. With the print control display turned OFF, soft hyphens will not appear on the screen. The current state of the print-control display is shown on the **^O** menu.

You'll learn more about hyphens in an upcoming lesson.

MOVING AROUND IN A FILE: At this point you've typed some lines of text. Experiment with the one-letter cursor movement commands listed on the left side of the main menu. These cursor-movement commands should be easy to learn because the cursor moves in the direction in which keys are positioned on the keyboard. This cursor command-key arrangement is illustrated here:



Just remember that the cursor moves in the same direction as the keys are positioned in the diamond on the keyboard. Try it. The cursor cannot move beyond the text on the screen in either direction, so you will get a better feeling for these commands if you have a nice chunk of text to play in.

Because the arrow keys also function like these cursor-control keys, whether you use the arrow keys or the control keys is up to you and will depend on your style of typing. Generally, fast typists should use the control keys because the hand does not have to move out of the typing position to use them. Conversely, "hunt-and-peck" typists will probably find the arrow keys more convenient.

If you type enough text, you'll discover another feature called "scrolling." Scrolling occurs when you have more than one screen full of text. Your monitor represents a "window" through which you look at your text, and WordStar automatically moves the document under the window as appropriate. As you add more text, the text in the window seems to move up because you're adding material at the bottom.

You can manually force WordStar to scroll the text by using the scrolling commands, ^Z, ^W, ^C, and ^R. Notice how the first two commands move the screen window one line at a time, while the latter two move the window about a full screen at a time.

DELETIONS: Try deleting some text by positioning the cursor within an existing line of text, and issuing ^G. The **^G** command deletes one character or space at the cursor and draws in the next character from the right. Issuing ^T deletes a whole word to the right of the cursor, and ^Y deletes an entire line. Note that the Osborne 1 does not have a **DEL** key as indicated on the menu. Use **^-** (control-hyphen) if you want the **DEL** function. Most professional typists prefer not to delete characters as they backspace; turning the **INSERT** feature OFF accomplishes this function. See lesson 4, "Editing a File," for more on the **INSERT** feature.

FLAGS: While you were scrolling the screen, did you see some strange symbols to the right of your text? These symbols are called "flag" characters. They indicate the status of text lines. Each line contains a flag character that tells you how WordStar interprets the line. Issue ^JF (our good friend HELP, again) to receive a full description of these flag symbols, then review the summary on flag characters in the Reference Guide.

SAVING A FILE: When you are through experimenting with the skills you've learned so far, you'll probably want to take a break.

Even though the document you have created may not be of permanent value, we will show you how to save it anyway so you learn from the beginning how to keep your work. Follow the directions for saving a file as described below to get a solid understanding of the SAVE commands.

Three methods exist for saving files:

SAVE AND REEDIT: Initiate this command by issuing ^KS. A message at the top of the screen informs you that the file **TRIAL.TXT** is being saved. A copy of the file is saved on diskette, while the original version of the file is left open for further editing. You should be returned to the beginning of the document. You can move your cursor back to the position it occupied before you issued the save command by following the instructions that are now displayed at the top of the screen (issue a ^QP), or simply start at the top of the document by hitting any key. You should get in the habit of periodically saving what you have typed with **^KS**, just in case of some unforeseen calamity. Also, if for some reason you must leave the computer, it is wise to issue a **^KS** command before you depart.

SAVE AND DONE EDITING: Issue this command by typing ^KD. This form of the save command saves the file under the name with which you created it, and returns you to the no-file menu. Use this method when you want to work with additional files or to save a file before printing it.

SAVE AND EXIT: You issue this command with **^KX**. Since you are already at the no-file menu, you cannot try this command without reopening the file. You can, however, simulate what would happen; just press [X], since the **^KX** command both saves the file like **^KD** and additionally exits WordStar like **X**. This method of saving a file is used when you are completely through with WordStar and wish to employ some other program or quit for the day. When WordStar stops executing, you should see the CP/M prompt A> at the bottom of your screen. (Remember that if you change program disks at this point, you should push RESET or issue a **^C** command to bring in the CP/M operating system from that disk.)

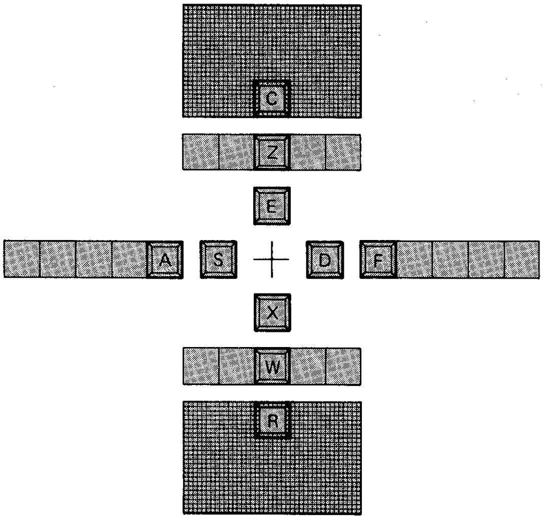
The foregoing was a large, but important lesson. We put just enough of every use of the basic editing functions of WordStar into that lesson that you should be able to create and edit documents on your own now. There's plenty to come, but if you'd like to go back and practice some more editing before moving on, the following chart should help summarize what you've learned.

Getting Started Summary:

no-file menu:	**D**	—to create a document file
	N	—to edit programs or data
	L	—to switch disk drives
	F	—to turn directory ON or OFF
interrupt command:	**^U**	—cancels current command
HELP commands:	**^JH**	—change HELP level
	^JS	—learn about status line
	^JF	—learn about flags in right margin
format commands:	**^OJ**	—turn justification ON or OFF
	^OW	—turn word wrap ON or OFF
	^OE	—turn soft-hyphen entry ON or OFF

^OD—turn print-control display ON or OFF

cursor controls: **^E**—cursor up one line
^X—cursor down one line
^S—cursor left one character
^D—cursor right one character
^A—cursor left one word
^F—cursor right one word
^W—scroll window down one line
^Z—scroll window up one line
^R—scroll window down one screen
^C—scroll window up one screen



deletion commands: **^G**—delete one character to the right
^T—delete one word to the right
^Y—delete entire line
^-—delete one character to the left

save commands: **^KS**—save copy and leave file open
^KD—save file and return to no-file menu
^KX—save file and quit

Lesson 4: Editing a File

You are now ready to move on and explore some more advanced editing techniques and formatting methods. Turn ON the Osborne 1 or if it is already ON push RESET to insure a fresh start for this lesson. Make sure that the diskette holding the file named **TRIAL.TXT** is in drive B and that the diskette containing WordStar is in drive A. Press RETURN but instead of specifying **D** for editing a document, press the X key from the no-file menu to exit to CP/M (like you did at the end of lesson 3).

Let's learn a new way of invoking WordStar. Enter the following in response to the A > prompt:

W S SPACE BAR B : T R I A L . T X T RETURN

This method of creating or editing a file eliminates the need to go through the no-file menu and log onto the B drive. You must indicate the drive in which the file you want to edit or create is located by appending the drive identifier followed by a colon (in this case, B:) to the front of the file name.

Preceding the file name with **B:** is not the same as logging onto the B drive; this procedure only temporarily activates the drive for that particular file.

The file you created earlier should now be opened and displayed on the monitor. Look on the status line for the file name B:TRIAL.TXT to verify that WordStar retrieved the file you specified, and not some other file.

Now it's time to investigate some of the editing features shown in the miscellaneous column on the main menu.

Insertions

The INSERT feature is normally ON. You can determine if INSERT is ON or not by checking to see if the words INSERT ON are displayed in the upper right corner above the menu (status line). Use the control-arrow combination to move the monitor window to make sure that the INSERT ON message is displayed. If it is not, turn it ON by issuing ^V.

To demonstrate the INSERT feature, place the cursor within some existing text and type a few letters. Notice that the existing text moves to the right and accommodates the typed characters, so you can insert a letter or even a whole sentence wherever you want. If you insert enough material, WordStar automatically moves words to the next line—we're assuming you haven't turned the justification feature OFF.

Now watch what happens when INSERT is OFF; issue the ^V command to turn INSERT OFF. The words INSERT ON disappear from the top right corner of the screen. Scroll the screen with the control-arrow keys to ascertain that the words are indeed gone. Now place the cursor within some text and type a few letters. With the insert feature OFF, the characters you enter replace those at the cursor position. Use of the insertion feature should be clear at this point; it either allows text to be inserted to the left of the cursor when ON, or replaces text at the cursor when OFF. To turn INSERT back ON, issue ^V—notice that this is one of those toggle-switch commands we mentioned earlier.

Two other commands need explanation in conjunction with INSERT. Use the INSERTA RETURN command, ^N, when you want a carriage return but you do not want the cursor to move with the return. In other words, the text following your cursor will move down a line, but your cursor will remain stationary. Note that the TAB command, ^I, moves the cursor to the next tab stop; if INSERT is ON, text moves with the cursor.

Reforming Paragraphs

After playing with these editing commands, you may have a disorganized paragraph with some lines extending beyond the right margin, and other lines not quite reaching it. Place the cursor at the beginning of the paragraph you wish to clean up, turn

Hyphen Help OFF with ^[O][H], and issue the paragraph REFORM command, ^[B]. The cursor will advance to the end of the paragraph, leaving the text in its wake perfectly aligned.

Unless you turn the Hyphen Help feature OFF, the cursor will stop in mid-paragraph and you'll get a message from WordStar at the top of your screen asking whether you want to insert a hyphen. For now, if you get such a message, simply issue another ^[B] to continue the reformatting operation.

Do not expect the REFORM operation to eliminate spaces or carriage returns that you have entered. You will quickly appreciate the usefulness of this command to realign margins and shore up spaces between words. The value of REFORM will become even more evident when you see how it can be used to change the format of an entire document.

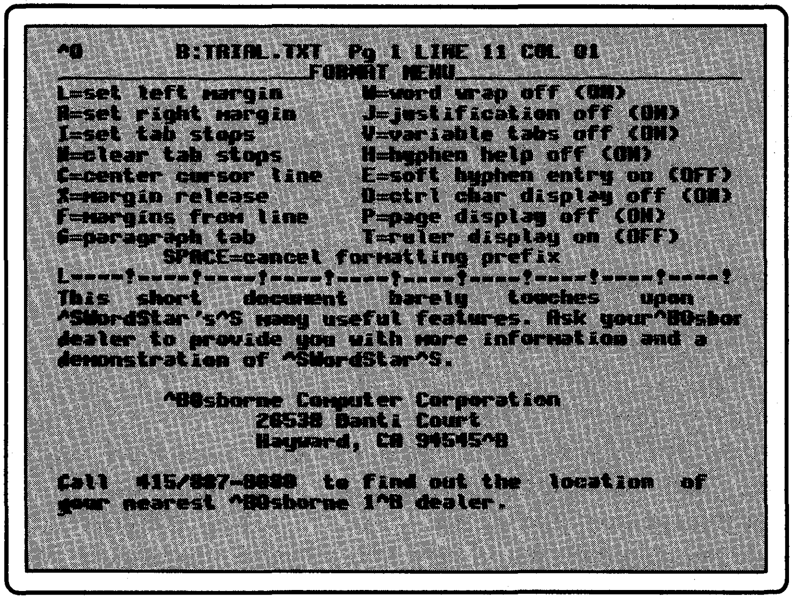
Changing the Format of a Document

One nice feature of WordStar is the ability to change the format (margins, line spacing, etc.) of a document at any time. In order to investigate the formatting procedures, you must have a few paragraphs of text in your file. If you do not have at least a page of text, take a moment and type something—use the text in this lesson if nothing else comes to mind.

It's also not important to be very accurate. After all, you'll be able to correct the mistakes by using WordStar's editing capabilities. So type away, and let the mistakes happen!

A normal WordStar page consists of 66 lines (several of these are reserved for margins in the default settings, so you actually only have 54 lines of text); a hyphenated line with a **P** in the rightmost column separates one page from the next. Use the down arrow to move the cursor to this dotted page-break line. This page divider is called the DYNAMIC PAGE-BREAK LINE.

When you reach this line, use the command ^O P to first turn the line OFF and then back ON again; the **^O** menu shows whether it is ON or OFF. Page length is adjustable, but you will have to wait awhile to see how this is done. If or when you have a page of text, issue ^O to gain access to the ON SCREEN FORMATTING menu:



```
^O        B:TRIAL.TXT  Pg 1 LINE 11 COL 01
_______________FORMAT MENU__________________________
L=set left margin       W=word wrap off (ON)
R=set right margin      J=justification off (ON)
I=set tab stops         V=variable tabs off (ON)
N=clear tab stops       H=hyphen help off (ON)
C=center cursor line    E=soft hyphen entry on (OFF)
X=margin release        D=ctrl char display off (ON)
F=margins from line     P=page display off (ON)
G=paragraph tab         T=ruler display on (OFF)
        SPACE=cancel formatting prefix
L----!----!----!----!----!----!----!----!----!----!
This  short  document  barely  touches  upon
^SWordStar's^S many useful features. Ask your^BOsbor
dealer to provide you with more information and a
demonstration of ^SWordStar^S.

        ^BOsborne Computer Corporation
              26538 Danti Court
              Hayward, CA 94545^B

Call  415/887-8080  to find out the  location  of
your nearest ^BOsborne 1^B dealer.
```

This menu lists commands you can use to format a document and also shows the current state of various features. Most of the features listed in the status section, except HYPHEN HELP and SOFT-HYPHEN ENTRY, should be ON.

You can switch a feature OFF or ON at any time by using the toggle command that precedes it on the **^O** menu. For example, you turn the RULER DISPLAY OFF with ^O T. The main menu returns, and the RULER LINE below the menu should no longer be visible. Turn the RULER LINE back ON with another ^O T. Experiment with each feature on this menu; when necessary, use the Reference Guide to get a full description of the feature.

The rest of the menu is devoted to commands that format documents. For example, enter the words **FORMATTING COMMANDS** on a blank line, and with the cursor still positioned in the same line, use the CENTER CURSOR LINE command ^O C. This command centers the words you just typed midway between your margins. Use the delete command ^Y to delete this title if it interferes with the contents of your file.

The lines in your document are now single spaced because this is the usual setting WordStar assumes when you first start using it. You can change the line spacing easily. Change the line spacing with the SET LINE SPACING command, **^OS**. Issue ^O then S, and WordStar will ask you:

```
ENTER space OR NEW LINE SPACING (1--9):
```

The number you enter in response to this prompt represents line spacing in the document and dictates the number of times the cursor or print carriage moves down at the end of each line. For example, enter 5 and press RETURN. Even though nothing appears changed in your existing document, rest assured that the line spacing is now set to 5. Four blank lines will separate each new line of text you enter from this point on. Text you've already entered will stay the same, unless you reformat it. The message LINE SPACING 5 should show at the upper right of the top line.

Now, to demonstrate how to change line spacing for an existing document, place the cursor at the beginning of a paragraph and REFORM it with a ^B command (you may have to issue ^B several times to get through the paragraph unless you turn Hyphen Help OFF with **^OH** beforehand). The cursor will move through the file, setting off each text line with four blank lines. To reestablish the original line spacing, use the ^O S command and enter 1 as the new line spacing. Go back and REFORM the paragraph with ^B and single spacing will once again be in effect.

Lesson 5: Margins and Tabs

You can change margins just as easily as line spacing. The following exercise will lead you through the process of changing the left margin to 5 and the right margin to 45:

1. To change the left margin, issue the LEFT MARGIN command, ^[O] [L]. When WordStar asks:

   ```
   LEFT COLUMN NUMBER (ESCAPE for cursor column)?
   ```

   enter [5] and press [RETURN].

2. To change the right margin, use the RIGHT MARGIN command, ^[O] [R]. When the following prompt appears:

   ```
   RIGHT COLUMN NUMBER (ESCAPE for cursor column)?
   ```

   enter [4] [5] and press [RETURN].

3. To format the document with the newly specified margins, place the cursor at the beginning of each paragraph and use the REFORM command, ^[B]. Notice that the RULER LINE reflects the new margin settings.

4. To restore the original left margin of 1, try a slightly different approach. Move the cursor to the far left-hand margin; you'll have to use the arrow keys to do so. Notice that the status line shows that the cursor is in column 1. Issue the LEFT MARGIN command, ^[O] [L]. When WordStar requests a column-number, simply press the [ESC] key. Pressing **ESC** will set the left margin at the column where the cursor is located (column 1).

5. Restore the original right margin (65) with the RIGHT MARGIN command ^[O] [R], and enter [5] [2] as the column number. You cannot use the **ESC** key in this case because you haven't learned how to move the cursor beyond the current right-margin setting of 52.

REFORM with ^B to reestablish the original and normally used WordStar margins.

You can temporarily disengage the margins so you can enter text outside of the present settings. To release the margins, use the MARGIN RELEASE command, ^O X. After issuing this command, you can move the cursor beyond the current margins. The message MAR REL will be displayed at the far right of the STATUS LINE while the margins are released. The original margins will again be in effect as soon as the cursor returns within the bounds of the current margins.

Something to consider when you format a document is that standard line spacing, margins, and tabs reset whenever you leave WordStar. This means that text in a file will remain formatted as it was when you saved it, but when you return to edit after leaving WordStar, the standard settings will be in effect.

To clarify this point further, suppose you were creating a document with 65 as the right margin, and you saved the file and left WordStar to use another program. When you returned to edit the file, existing text would still have 65 as its right margin, but entry of any subsequent text or reformatting would have 52 as its right margin.

There is an easy way to set the margins and tabs to match those of a previously composed document. Place the cursor within a full line of existing text and issue the FROM FILE LINE command, ^O F. If your existing text has 65 as the right margin, the **^OF** will reset the current right margin to 65.

Another practical application demonstrates the FROM FILE LINE command. We will create an imitation ruler line that will specify the margin and tab settings for your file.

Place the cursor at column 5 (you can tell by looking at "COL" on the status line) and type **L** as your left margin marker. Now enter hyphens—make sure they're not soft hyphens—all the way to column 52 and type **R** to mark the right margin. You can position exclamation points (!) on the line where you want tabs set; enter a number sign (#) to denote decimal tab stops (an indicator that aligns decimal points in numbers).

Leave the cursor within the imitation ruler line and issue the FROM FILE LINE command, ^OF. This command clears the present margins and tabs and sets those the new ruler line specifies. Observe that the real RULER LINE below the menu reflects the new settings. To prevent the imitation ruler line from appearing when you print the document, insert two periods at the front of the line in columns 1 and 2. These periods are a DOT command, and they inform the printer not to print this ruler line; you'll learn all about DOT commands later on.

The TAB command definitions on the **^O** menu are fairly self-explanatory. **^I** (or TAB) advances the cursor to the next tab stop, moving text with it if INSERTION is ON. The PARAGRAPH TAB command, **^OG**, temporarily sets the left margin to the next tab stop and remains in effect until you type something, issue a margin command, move the cursor to the left of the position in which you issued **^OG**, or press RETURN. Experiment with these tab commands if you like!

Remember that tab stops are shown below the menu on the RULER LINE. WordStar shows regular tabs as exclamation points (!) and automatically sets them at every fifth column. You can set your own tabs in order to format a document or arrange columnar data. The SET TAB STOP command, **OI** (or ^O followed by the TAB key) sets tab stops. Either of these commands prompt you for the column number in which you want the tab set. You can specify a column or press **ESC** to indicate the column containing the cursor as the tab site. Follow this exercise to become familiar with tab arrangement:

1. To clear the tab stops in column 11 use the CLEAR TAB command, ^ON. WordStar will ask:

```
CLEAR TAB AT COL (ESCAPE for cursor col; A for all)?
```

Enter 1 1 and press RETURN. Watch the exclamation point in column 11 disappear from the RULER LINE.

2. To clear the tab set at column 16, move the cursor to column 16 (as shown on the STATUS LINE), issue the CLEAR TAB command, ^O N, and either enter the column number 1 6 followed by RETURN, or press the **ESC** key. Either of these actions clears the tab at the column containing the cursor.
3. To clear all of the tab stops, again use the ^O N command, but this time answer the prompt with A followed by RETURN to clear all of the tab settings. The RULER LINE should now consist of a dotted line of hyphens with **L** on the left and **R** on the right.
4. To set a regular tab, use the SET TAB command, ^O I. This message appears:

```
SET TAB AT COLUMN (ESCAPE for cursor column)? .
```

Enter 3 3 and press RETURN to set a tab at that column. An exclamation point positioned roughly in the middle of your screen on the RULER LINE represents the tab you have set. As before, you could have set the tab at the cursor column by pressing **ESC**. Entering **#** before you enter the column number or before pressing **ESC**, sets a decimal tab.

Columns of numbers may be aligned on the decimal point, or text can be right-aligned through the use of decimal tabs. After you tab over to a decimal tab stop, characters you enter will move to the left, pushing the entire field to the left of the decimal setting. The cursor remains at the tab position until a period is entered, thus terminating that particular decimal tab.

Decimal numbers entered under the decimal tab will align with the decimal point in the column where the tab was set. Text that does not contain decimal points or periods will align with the character one column to the left of the decimal tab stop. Decimal tabbing is only active when VARIABLE TABBING is ON. You can determine whether this feature is ON by looking at the **^O** menu.

The VARIABLE TABBING command, ^O V, switches this feature OFF or ON. This feature is normally ON so that variable tab stops are in effect and you can specify tab settings if you desire. When VARIABLE TABBING is OFF, fixed tabs are in effect. Fixed tabs are not normally used for standard word-processing operations; use fixed tabs when you use WordStar to write computer programs.

If you intend to write programs using WordStar, you need to know that when VARIABLE TABBING is OFF the tab character (^I or the binary number representing 9 decimal) is used in the file and is displayed with fixed stops every eight columns, as opposed to the multiple spaces WordStar enters into the file when VARIABLE TABBING is ON. You should turn this feature OFF when you develop programs for use with the CP/M text editor **(ED.COM)** or Micropro's WordMaster editor.

Since each fixed tab is a single character, it acts differently than the multiple characters in variable tabs. The cursor cannot go within the white space representing the fixed tab, and the cursor advances over the tab. Text inserted before the tab appears in front of it until enough text is entered to force the tab to move to the next tab position.

To refresh your memory on line spacing, margins, and tabs, use the MARGIN and TABS HELP command, ^J M.

Hyphen Help

One last thing on the **^O** menu that we promised to show you and that may interest you is an explanation of the **HYPHEN HELP** feature. HYPHEN HELP is usually ON. Look on the **^O** menu to determine what condition this feature is currently in and if it is not already turned ON, do so with **H**.

With HYPHEN HELP ON, place the cursor at the beginning of a paragraph and issue the REFORM command, ^B. If there happens to be a long word that can be divided between lines and thus improve spacing, WordStar will indicate where this word can be hyphenated. HYPHEN HELP checks that the word can, in fact, be divided by syllables; selects a proper position for the hyphen; and then allows you to decide if you want to insert a hyphen at the indicated location.

The cursor will stop where HYPHEN HELP suggests that you enter the hyphen; though you can move the cursor around and select another place to insert the hyphen if you like. Once you insert a hyphen, the REFORM operation, including HYPHEN HELP, will continue until the end of the paragraph is reached. If a hyphen is not desirable at the suggested location, simply issue another ^B, and the operation will continue. If SOFT-HYPHEN ENTRY is ON, hyphens will be soft. To explicitly enter a hard hyphen while SOFT-HYPHEN ENTRY is on, issue **^P** before typing the hyphen.

Lesson 6: Block Maneuvers

On the **^K** menu are the commands used to save a file, or manipulate a file just as if from the no-file menu. Also shown are the BLOCK commands used to manipulate a specific portion of text. The BLOCK commands are used to select a portion of text so you can move, copy, delete, or even write it to another file. Here is the K menu:

There are two types of blocks: the usual margin-to-margin blocks and special column-to-column blocks. The usual WordStar blocks are used for normal text and encompass everything between the beginning and ending markers. On the other hand, "column blocks" are different in that they extend from the column of the beginning marker to the column of the end marker. Column blocks are usually used to manipulate columns of numbers or tables without disturbing data on either side. To specify a column block, use the COLUMN MODE switch **^KN** whose current state may be determined by looking on the **^K** menu.

To mark a block and, in effect, select that portion of the document to be manipulated, place the cursor at the beginning of the material you want blocked and use the BLOCK START command, ^KB. A < B > will appear on the screen to indicate the position of this beginning marker. Next, move the cursor to the end of the section you wish to block and use the BLOCK END command, ^KK. The text within the block will become dim, making it easily discernible from its surroundings.

The marked block is always the source for any block operations. You can redefine the boundaries of a block by moving the cursor and setting a new beginning and/or ending marker. Block operations that require a destination assume the cursor position at the time of the block command as the location for the new placement of the block.

There is a limit to the size of block you can move or copy. If you see the BLOCK TOO LONG message, reposition the BLOCK END MARKER closer to the beginning and try manipulating the text in smaller segments.

To make a copy of your block, move the cursor to the location in your document where you want the copy transferred. Use the INSERT A RETURN command, **^N**, to make room if you want the copy inserted within some existing text. Then issue ^KC.

The BLOCK COPY command, **^KC**, produces a duplicate of the block and includes the block markers at the destination specified by the cursor. The duplicated portion of the document will become the new block, and the cursor will be set at its beginning. Text at the original site of the block will remain unaltered. If you'd like to return to the place from which you copied the block, use the CURSOR TO SOURCE OF LAST BLOCK command, ^QV.

When you finish making copies, you can use the HIDE/DISPLAY

BLOCK command, ^[K][H], to return the characters to their normal brightness (i.e., "unmark" them). When a block is hidden in this manner it is impervious to any block operations until you redisplay it with another **^KH** command. Try to make a copy while it is hidden; you can't do so.

Now redisplay the block with the ^[K][H] command and we will demonstrate how to move a block. To move the block of text, place the cursor at the desired destination and use the BLOCK MOVE command, ^[K][V]. The block will be moved to the location specified by the cursor. The block markers move with the block and remain displayed. Hide the block with ^[K][H] and then move the cursor with ^[Q][V] to the space that resulted from moving the block. If necessary, use the REFORM command, **^B**, to clean up the margins.

At this point you should note the difference between the BLOCK MOVE command and the BLOCK COPY command. Copying a block duplicates the marked text in a new location—you will end up with two copies of your text. Moving a block physically relocates the marked text to the new location.

If you have been trying these procedures as we described them, you should have at least two duplicate portions of text. Try to delete one of these portions with the BLOCK DELETE command, ^[K][Y]. Nothing happens if your block was still hidden from the last demonstration. If the block was not hidden, then it vanished from the screen along with its markers. Redisplay your block with ^[K][H] and delete it with ^[K][Y]. The block will disappear, leaving the block markers at the sight of the deletion.

You can see why it is a good idea to always hide a block when you are through with it; you don't want to accidentally delete it. When a block is deleted, the markers are hidden and remain at the deletion site. The cursor does not move, but **^QV** can send it to where the deletion took place.

Sometimes you may want to extract a segment of text and save it in a separate file on diskette for later use. Since you deleted your last block of text, mark another block. Next, issue the BLOCK WRITE TO FILE command, ^K W. After you issue **^KW**, you'll get a request for the name of a file where a copy of the block is to be sent:

```
NAME OF FILE TO WRITE MARKED TEXT ON?
```

Supply a valid file name so that WordStar knows where you want the block stored. If you specified the name of a file that already existed, this message would appear:

```
FILE B:name.typ EXISTS ... OVERWRITE? (Y/N)
```

If you pressed **Y** (for yes), the block would replace the contents of the existing file. If you pressed **N** (for no), you would be given a chance to supply another file name. Send the block you have marked to a file named TRIAL.BLK on the B drive.

Enter **B:TRIAL.BLK** in response to the NAME OF FILE prompt. Press RETURN after supplying the file name.

NOTE

*The disk identifier (**B:**) is necessary in this case because of the way you started WordStar at the beginning of these lessons and opened the file to be edited. If you had "logged" the B drive with command **L** from the no-file menu, the drive identifier would not be needed. It is also possible to make the B drive the logged drive from within the file you are presently working on with the **^KL** command.*

Once you have finished writing (copying) the block to the file named TRIAL.BLK, hide the block with ^K H. Now, just to make sure that the block was copied successfully, issue ^K L, enter **B:** and press RETURN to log drive B, then issue ^K F to look at the directory of drive B for the file named TRIAL.BLK. Issue ^[K] [F] again to get rid of the directory.

Place Markers

Before we conclude this section, we want to teach you about the use of PLACE MARKERS. Do not confuse these markers with the BLOCK MARKERS, since they serve distinctly different purposes.

Use PLACE MARKERS to mark up to ten separate locations in your file where you may later send the cursor. You set a PLACE MARKER by positioning the cursor at the desired location and issuing ^[K] followed by a number (0 through 9). The numbered marker will be displayed as a "dim" character at the specified location. The marker remains fixed until you use it somewhere else in the file. When you are through with the marker or no longer want it displayed, hide it by issuing ^[K] and the number that set the marker.

To send the cursor to a particular place marker, simply use the CURSOR TO PLACE MARKER command, ^[Q], and the number of the marker. The Help command, [J] [P], presents more information about the use of PLACE MARKERS.

You're ready to take another break. If you want to save the file you created, do so with the SAVE EXIT command, ^[K] [X]. If you would like to stop editing for a while and don't mind losing this most recently edited version of your file, use the ABANDON EDIT command, ^[K] [Q]. Since you've made changes to the file, WordStar will ask you:

```
Abandon Edited file B:TRIAL.TXT? (Y/N)
```

Reply with a [Y] to indicate to WordStar that you're not interested in permanently saving the changes made since the file was opened. You could probably use a rest before going on to lesson 7.

Lesson 7: Some Finishing Touches Using WordStar

Start over! Put your WordStar diskette into drive A, push the **RESET** button, and press RETURN. Make sure the storage diskette in drive B holds your **TRIAL.TXT** file. You should see the no-file menu displayed on your screen. If not, refer to the beginning of this chapter to see how to start WordStar.

Before you begin editing the file named **TRIAL.TXT**, make sure you've "logged onto" the diskette in drive B. Issue the CHANGE LOGGED DRIVE command, L, and enter **B:** as the active drive. If you have been following along with us, at least three files should be listed in the directory of the diskette in drive B.

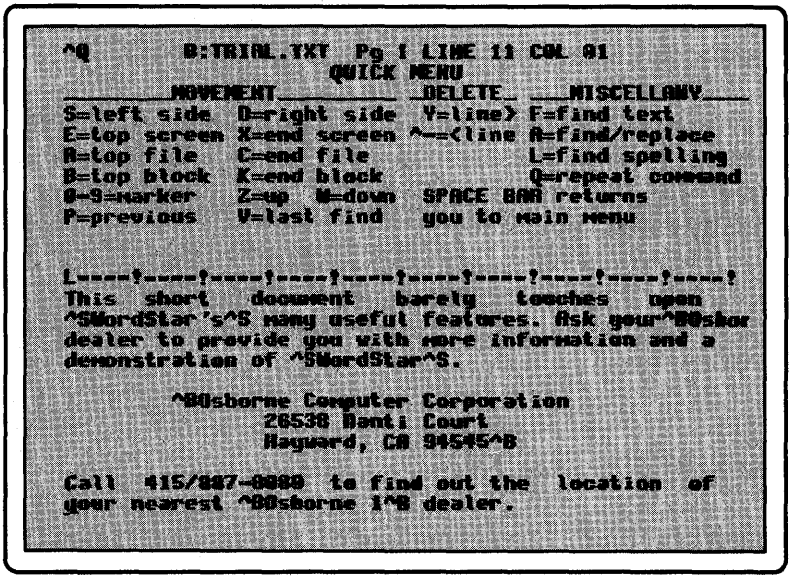
The files listed should include your sample file called **TRIAL.TXT**; the file you named **TRIAL.BLK**, which contains the sample block you saved; and maybe another file called **TRIAL.BAK**. WordStar always creates a "backup" file whenever you edit. The backup file has the same name as the file you edited, but has the file type **.BAK**. This backup file reflects the file's contents as they existed before you began editing it the last time. If you abandoned your last file by issuing **^KQ**, then no backup file will have been made.

WordStar will not erase a previous backup copy of a file until the end of an editing session. As you look at the directory of drive B, the file named **TRIAL.TXT** contains the version of that file before you last saved it. **TRIAL.BAK** contains the version of that file prior to the previous editing session—that is, the file as it was before the last time you edited it. If you now again edit **TRIAL.TXT**, when you save it the old **TRIAL.TXT** file will be renamed **TRIAL.BAK**, the old **TRIAL.BAK** will be erased, and the new version of your file will be saved in a file named **TRIAL.TXT**.

The exception to the above procedure is when you abandon an edit. If you use the **^KQ** command and specify that you wish to abandon the editing you've done, WordStar will replace the files on the diskette exactly as they were before you began the current editing session.

Now that you understand about "backup" files, open the file **TRIAL.TXT** with the create or edit-a-document file command, D. You should know about a few more editing "goodies" before you prepare your document for printing.

All of the commands you are going to use to make last-minute refinements are listed in the QUICK menu, so issue ^Q and you will see the Quick Menu:



```
^Q       B:TRIAL.TXT  Pg 1 LINE 11 COL 01
                   QUICK MENU
________MOVEMENT____________ _DELETE_ ___MISCELLANY____
S=left side  D=right side  Y=line> F=find text
E=top screen X=end screen  ^--=<line A=find/replace
A=top file   C=end file            L=find spelling
B=top block  K=end block           Q=repeat command
0-9=marker   Z=up  W=down  SPACE BAR returns
P=previous   V=last find   you to main menu


L----!----!----!----!----!----!----!----!----!----!
This  short   document   barely   touches   upon
^SWordStar's^S many useful features. Ask your^BOsbor
dealer to provide you with more information and a
demonstration of ^SWordStar^S.

          ^BOsborne Computer Corporation
                26538 Danti Court
                Hayward, CA 94545^B

Call  415/887-8080  to find out the  location  of
your nearest ^BOsborne 1^B dealer.
```

The cursor commands listed in the above menu move the cursor through the file as quickly as possible. Issuing ^Q S, for instance, sends the cursor to the left side of the screen. Issuing ^Q D sends the cursor to the right end of the current line. Issuing ^Q X sends the cursor to the bottom of the screen, and issuing ^Q E sends it to the top of the screen. Try using these commands to move the cursor through your file. ^Q R advances the cursor to the beginning of the file, and ^Q C advances it to the end.

You should notice that the letter following the "Q" prefix follows the same pattern as did the cursor-control diamond presented earlier: S is a leftward movement, D is a rightward movement, E is up, X is down. WordStar is consistent in the placement of its command characters.

The other cursor commands listed in the left column of this menu send the cursor to various markers. **^Q** and a number between 0 and 9 send the cursor to the appropriate PLACE MARKER. **^QB** and **^QK** send the cursor to the beginning and ending block markers, respectively. Issuing **^QV** advances the cursor to the source of the last block or "find" operation; we'll describe "find" in a minute. Last, but not used less often, is the **^QP** command, which sends the cursor to the position it occupied prior to the most recent command.

As indicated at the bottom of the menu, issuing ^QZ will continuously scroll the screen upward, while ^QW will scroll it downward. You press the SPACE BAR to stop the scrolling started by either command.

Also on the **^Q** menu are commands to delete a portion of a line. Issuing ^QY deletes everything in the current line to the right of the cursor.

Lesson 8: Finding Text

The next set of commands we want to introduce you to on the **^Q** menu are used to quickly find or change sections of text. The FIND commands locate words or phrases and optionally replace them. Pick a rarely used word in your trial file and keep it in mind. Use the ^QR command to move the cursor to the beginning of the document. With the cursor at the beginning of the file, issue the FIND command, ^QF. The following prompt will appear on the screen:

```
FIND?
```

Enter the word that you want to find—the word we told you earlier to keep in mind—and press RETURN. You will see this message:

```
OPTIONS? (? for Info)
```

Options allow you to specify such things as matching whole words, ignoring the distinction between uppercase and lowercase letters, or searching backward instead of forward. Entering a question mark (?) will show these options and they are described more fully in the Reference Guide. For the time being, press RETURN to ignore the **OPTIONS?** question. The screen will flicker slightly, and the cursor should move to the word you specified. If the message below appears:

```
* * NOT FOUND: "word" * * * Press ESCAPE Key
```

where "word" is the word you specified, you must have made a mistaken entry, or the cursor was not at the beginning of the document when the FIND operation started. Press the **ESC** key and try the FIND command again. Once the desired word or phrase is located, proceed to the next occurrence of the same word—if there is one—with the FIND AGAIN command, ^L.

Another version of the FIND command allows you to locate a phrase or word and replace it with another phrase or word. The FIND/REPLACE command, ^Q A, locates a word or phrase just as does the FIND command, **^QF**, but it takes the operation a step further by replacing the found word with another. The first step in this command asks for the word to replace; then, after you supply the word you wish to locate and press RETURN, WordStar asks:

```
REPLACE WITH?
```

Respond by entering the replacement text, and press RETURN. Ignore the **OPTIONS?** prompt by again pressing RETURN. After the FIND/REPLACE operation starts, the word will be located, and the following prompt will appear in the upper right of the screen (use the control arrow keys).

```
REPLACE (Y/N):
```

The cursor will blink to indicate that a decision concerning the located word is required. If you want to replace the located word, press [Y], otherwise, press [N]. Then, issue the FIND/REPLACE AGAIN command, ^[L], to continue the search and find the next occurrence of the word you specified.

Repeat Any Command

At the bottom of the Q-menu is a useful command that repeats any valid command as many times as you like. For instance, place the cursor at the start of a sentence and issue the REPEAT NEXT COMMAND command, ^[Q] [Q], and then the CURSOR RIGHT A CHARACTER command, ^[D]. The cursor will move a column at a time to the right, and you will see this message above the RULER LINE:

```
TYPE 1--9 TO VARY SPEED, SPACE TO STOP
```

This message tells you that you can increase or decrease the speed with which the command executes. The slowest speed is 9, and the fastest is 1. If you don't specify a rate, the command will execute at the normal rate of 3. To terminate the command, press the [SPACE BAR]. You can use this command to repeat any other command.

Lesson 9: Print-Control Characters

Before you print your document, you can add special command characters that control your printer and produce special effects. All of these PRINT-CONTROL characters are listed in the PRINT-CONTROL HELP menu (**^JP**) and the Reference Guide, so if you want more information quickly, remember to use these added resources.

You enter print-control characters into the file by first issuing the ENTER CONTROL CHARACTER command, **^P**, followed by the control character. Documents containing control characters will appear disorganized, with lines of text possibly overflowing margins. Do not worry too much about the appearance of your text, because PRINT-CONTROL characters are purely symbolic and will not print or affect the printed formatting of your document. Also, there is a way to make the control characters disappear to see exactly how your document will appear with the **^OD** command.

Many of the print-control features we are about to investigate greatly depend on the capabilities of your printer and the software that directs it. Refer to the section in Chapter 2 in this manual that tells you how to use **SETUP** to match your printer to the Osborne 1. Your dealer should also be able to supply you with the necessary information to help you fully utilize your particular printer. You may also have to use another program, **INSTALL**, to inform WordStar that you have a special printer. See Appendix A for details on the INSTALL program.

In this demonstration you will receive instructions to insert specific text. You may want to create a new file devoted especially to learning the following print controls; save your current file with ^K D, then create a new file called **TRIAL.TST** from the no-file menu. An alternative method is to enter the text and control characters we tell you into your present file, then block it and send it to a file named **TRIAL.TST** with the **^KW** command.

Some of the print-control characters are used like toggle commands: each successive use of the character either turns the feature ON or OFF. You embed print-control characters in your text by using the ENTER CONTROL CHARACTER command, **^P**, followed by the control character. To test your printer's capability, enter each example exactly as shown, preceding each of the printer control characters with **^P**. Number each example so you can keep track of which features work when you print the file.

You can make titles and headings stand out by using the BOLDFACE CONTROL character, **^B**. Boldface type is created when characters overstrike characters offset slightly from one another on DAISYWHEEL and other printers capable of incremental motion. The character may strike numerous times on less expensive Teletype-like printers to create boldface type.

Enter the words:

```
1. This is an example of the BOLDFACE feature.
```

Next place the cursor in front of the word **BOLDFACE**, issue the command ^P , and press the control letter B. Now move the cursor to the end of the word **BOLDFACE** and again issue ^P followed by the control letter B. The top line of your file should look like this:

```
1. This is an example of the ^ BBOLDFACE ^ B feature
```

DOUBLE-STRIKE causes each character to strike twice for extra clarity but is not as dark as boldface. This PRINT-CONTROL character can also produce an extremely sharp impression for an entire document on printers that use carbon ribbons.

Enter the words:

```
2. This is an example of the DOUBLE-STRIKE feature.
```

Surround the words **DOUBLE-STRIKE** with ^P D on each side. The first **^PD** turns the DOUBLE-STRIKE ON, the second **^PD** turns it OFF. The second entry in your file should now look like this:

```
2. This is an example of the ^ DDOUBLE-STRIKE ^ D feature
```

Inserting characters to control the printer is easy. Continue by entering the text in each of the following examples. Remember to use **^P** before each of the control characters:

3. This is a test of the ^SUNDERLINING^S feature. ^STo—underline—an—entire—line—place—underscore—characters—between—words—like—this.^S

4. This is a test of the ^XSTRIKE OUT^X feature used to illustrate the omission of text.

Subscripted text prints slightly below the surrounding text. The subscripted portion is lower by 3/48ths of an inch, and the DOT command **.SR** (described later), can change this amount. Some printers are not capable of fractional line roll, so subscripts print on the line below if it's empty; otherwise, there is no subscript effect at all.

5. This is a test of the ^VSUBSCRIPT^V feature, which slightly lowers the text.

6. This is a test of the ^TSUPERSCRIPT^T feature, which slightly raises the text.

The color of print can change on printers that have a two-color ribbon. Special installation is required to activate this character on nondaisywheel printers.

7. This is a test of the ^YRIBBON COLOR TOGGLE^Y on printers with color selection.

The PRINT-CONTROL characters just described were toggle commands that turned a particular feature ON and OFF. The control characters in the remainder of this exercise initiate a feature at a particular point in the document. Proceed in the same manner as you have, continuing to number your sample entries.

You can enter the control character, **^C**, into a file to tell the printer to stop. Use this character as many times as you want anywhere in the text. When the printer receives the **^C**, printing halts, and the message PRINT PAUSED appears on the STATUS LINE. The no-file command, **P**, or the in-file command, **^KP** —whichever initiated printing—will restart the printing.

8. This is a test of the STOP PRINT ^C CONTROL character, which halts the printing until P or ^KP is issued.

Character pitch can switch between ALTERNATE pitch, **^A** (Elite), which is usually 12 characters per inch, and STANDARD pitch, **^N** (Pica), ten characters per inch on Daisywheel printers. You can further modify these features with the DOT command .CW, described later.

9. ^A. This is a test of the ALTERNATE character feature. ^N. The STANDARD character pitch is returned in this sentence.

After you know about DOT commands and are using .HE and .FO to specify headings and footing, you can use the LEFT-RIGHT HEADING/FOOTING PRINT CONTROL, **^K**. Its effect is to print headings, page numbers, etc., that print on the left-hand side of even numbered pages and on the right-hand side of odd pages so text prints in the appropriate position when the document is collated.

Depending on your printer, the **PHANTOM SPACE** and the **PHANTOM RUBOUT** control a specific code that prints a special character. The exact character depends on the print wheel in use. Try it!

10. This is a test of the PHANTOM SPACE character (^F) and the PHANTOM RUBOUT character (^G), which will be a surprise.

11. This is a test of the NONBREAK feature which causes separated ^Owords ^Oto^Obe ^Okept^Otogether.

The STRIKEOVER CONTROL character is useful for putting accent marks over letters or creating special symbols by printing different characters in the same spot.

12. This is a test of the STRIKEOVER feature, which creates an accent mark over the word "repousse ^H'."

WordStar makes allowance for four user-defined USER PRINT-CONTROL characters. These control characters are for accessing special printer features that vary among printers. Until you define these commands, the control characters **^Q**, **^W**, **^E**, and **^R** serve no purpose. See your dealer for more information on defining these features.

Printing a File

If you have a daisywheel or thimble-variety printer—capable of incremental spacing—and either you or your dealer has properly configured WordStar for use with such a printer, the printed copy you receive will be "microspace" justified. This means that "soft spaces" will disappear. (Soft spaces are those extra spaces between words on your screen that let your margins align.) Microspace justification eliminates gaps and spaces words evenly in printed documents.

If your printer is one of the more popular, inexpensive dot-matrix varieties, soft spaces achieve margin justification in printing.

It's time to check which PRINT-CONTROL features your printer uses by printing your test file, **TRIAL.TST**.

To print the file **TRIAL.TST**, you must first save a copy of this file on diskette. You can initiate printing of the file you are editing, but only if you have saved a copy of the file. Since you haven't saved **TRIAL.TST** yet, it does not exist to be printed. The usual sequence for printing a file is to use the SAVE DONE command, **^KD**, first and then issue the no-file PRINT command, **P**.

An alternative to the aforementioned method is to save the file **TRIAL.TST** with the SAVE REEDIT command, **^KS**, and then issue the PRINT command, **^KP**. After you issue **^KP**, WordStar will inform you that the most recently saved version will print and that you cannot save the current version until printing is completed.

If you transferred your test material by first marking it with block markers and then writing it to the **TRIAL.TST** file with the **^KW** command, you could now use the PRINT command, **^KP**. You can examine the file directory to determine what files reside on disk by issuing the FILE DIRECTORY command, **^KF**. You can also issue **^F** when the prompt asks for the name of the file to be printed. You can always use **^F** like this to see the directory whenever you're manipulating a file from within another file.

To make things simple for this demonstration, save the file you're working on with the ^K D command and return to the no-file menu. Next, issue P to initiate the PRINT FILE operation. Once you have asked to print a file, WordStar will prompt you through each successive step as follows:

```
NAME OF FILE TO PRINT?
```

Supply the name of the drive in which the file is located—if it is different from the currently "logged drive"—and the file's name; in this case, enter:

```
TRIAL.TST
```

and press RETURN. Remember that you can edit your file name as you learned at the beginning of this chapter.

When the file you name is located, PRINT OPTION questions come up in sequence. We will explain these options, but in normal circumstances, you usually press RETURN to ignore each of them. You can completely avoid the PRINT OPTION questions by typing the name of the file you want printed and pressing ESC instead of RETURN to complete the command.

The following option descriptions appear in the order in which they occur on the screen. For now, press RETURN after reading each option.

```
DISK FILE OUTPUT (Y/N):
```

This option allows the contents of a printed file to go to another disk file whose name you specify. Press N or press RETURN to ignore this question. Press Y to tell WordStar to send the formatted output to a disk file instead of to the printer.

```
START AT PAGE NUMBER (RETURN for beginning)?
```

Supply the number of the page at which the printing should begin or press the RETURN key to start printing with the first page of the file. This option continues printing from a specific page number after an interruption, as might be necessary if you ran out of paper or ribbon on your printer.

```
STOP AFTER PAGE NUMBER (RETURN for end)?
```

Supply the page number at which you want printing stopped, or press RETURN to print to the last page of the file. This allows you to stop the print operation at a predetermined page.

```
USE FORM FEEDS (Y/N)?
```

This option allows "form-feed" characters to output to the printer. It is customarily used with nondaisywheel printers to allow for the discrepancies that sometimes occur if your printer uses form feeds to paginate listings. Press RETURN or press N to ignore this question.

```
SUPPRESS PAGE FORMATTING (Y/N):
```

An exact replica of the file as it appears on the screen, including DOT commands, can print, regardless of top margins, bottom margins, headings, and footings. This option is often used to proofread a file and check the embedded print commands without actually using them. Press RETURN to ignore this option.

```
PAUSE FOR PAPER CHANGE BETWEEN PAGES (Y/N):
```

Press Y if you want the printer to stop after printing each page. This option is handy if you are using single sheets of paper, as with letterhead. When the printing stops at the end of a page, insert the next sheet of paper and issue P or ^K P —depending on whether you're at the no-file menu or editing a file, respectively—and printing continues. The message PRINT PAUSED appears on the STATUS line while printing halts. Press N or press RETURN if you have continuous-feed paper in your printer.

```
Ready printer, press RETURN:
```

This message indicates that the printer is ready to begin. Make sure the cable between your Osborne 1 and your printer is connected, that the printer is turned ON, and that the printer's "on-line" light, if it has one, is lit. Press RETURN to initiate printing.

A message at the top of the screen informs you that printing is in progress. If you want to stop printing at any time, issue the same command you used to initiate the print operation: P or ^K P. The PRINT commands, **P** or **^KP**, act like a toggle switch that halts printing temporarily and gives you a choice of three options:

```
"Y" TO ABANDON PRINT, "N" TO RESUME, "U" TO HOLD
```

If you respond with a Y, the print operation in progress will discontinue. Pressing N causes printing to resume. You can also press U to temporarily suspend printing and then use the **P** or **^KP** command to resume printing. This last option comes in handy when you need to readjust the paper or need to leave the computer unattended for a few minutes.

The message PRINT PAUSED will display on the STATUS LINE anytime printing is interrupted. By the way, this message should appear on your screen as a result of the **^C** in example 8 of the file **TRIAL.TST**. As described above, issue a [P] or ^[K][P] to finish printing your file.

You now know how to tell WordStar to start or stop printing a file. You need to know other commands, called "DOT commands," that concern printing, though.

WordStar Commands

Here's a list of WordStar commands and functions, all assembled in one place and arranged conveniently in alphabetic order:

(Press CONTROL key, then type letter)

^A	Cursor word left	**^JS**	Defines status line
^B	Paragraph REFORM	**^JV**	Defines text moving
^C	Scroll up screen	**^K0–9**	Set/Hide place markers
^D	Cursor character right	**^KB**	Mark/Hide block begin
^E	Cursor up line	**^KC**	Block COPY
^F	Cursor word right	**^KD**	Save file
^G	Delete character right	**^KE**	RENAME file
^H	Cursor character left	**^KF**	Directory ON/OFF
^I	TAB advance	**^KH**	Hide/Display block
^J	Prefix HELP	**^KJ**	Delete file
^K	Prefix editing	**^KK**	Mark block end
^L	FIND/REPLACE AGAIN	**^KL**	Switch logged drive
^M	RETURN	**^KO**	COPY file
^N	Insert carriage return	**^KP**	PRINT
^O	Prefix formatting	**^KQ**	Edit abandon
^P	Enter control character	**^KR**	Read a file
^Q	Prefix cursor editing	**^KS**	Save file reedit
^R	Scroll down screen	**^KV**	Block move
^S	Cursor character left	**^KW**	Write block to file
^T	Delete word right	**^KX**	Save file exit
^U	INTERRUPT commands	**^KY**	Block delete
^V	INSERT ON/OFF	**^OC**	Center cursor line
^W	Scroll down line	**^OD**	Display DOT commands

^X	Cursor down line
^Y	Delete line
^Z	Scroll up line
^JB	Defines REFORM
^JD	Print directives
^JF	Defines FLAG characters
^JH	Set HELP level
^JI	Command index
^JM	Defines tabs, margins
^JP	Defines place markers
^JR	Defines ruler line
^OT	Display ruler
^OV	Variable tabs ON/OFF
^OW	Word wrap OFF/ON
^OX	Release margins
^PM	Overprint next line
^PO	Enter nonbreak space
^Q0–9	Cursor to marker
^QA	REPLACE
^QB	Cursor block beginning
^QC	Cursor file end
^QD	Cursor right end of line
^QE	Cursor top screen

^OE	Soft-hyphen entry
^OF	Set margins, tabs as exist
^OG	Paragraph tab
^OH	Hyphen Help ON/OFF
^OI	Set tab stop
^OJ	Justification ON/OFF
^OL	Set left margin
^ON	Clear tab stops
^OP	Display page break ON/OFF
^OR	Set right margin
^OS	Set line spacing
^QF	FIND
^QK	Cursor block end
^QP	Cursor previous position
^QQ	REPEAT next command
^QR	Cursor file beginning
^QS	Cursor screen left
^QV	Cursor source (block, find)
^QW	Downward scroll
^QX	Cursor screen bottom
^QY	Delete to end of line
^QZ	Upward scroll
^Q←	Deletes to front of line

^-	Deletes character left
ESC	Error release
RETURN	Hard carriage return
TAB	Tab
NO FILE MENU-	(When no file is being created or edited)
D	Create or edit document file
E	Rename file
F	File directory OFF/ON
H	Set help level

L	Change logged drive
M	Merge-print
N	Create, edit nondocument file
O	Copy file
P	PRINT, stop print, start print
R	Run program
X	Exit to system
Y	Delete file

Lesson 10: DOT Commands

When you become better versed in using WordStar you will want to have some control of the way your printed document looks. Before preparing a document to be printed you will want to insert DOT commands in your file that tell WordStar what parameters the finished document is to have. You have already learned how to perform basic on-screen formatting; now you need to learn about the special DOT commands that affect page length, margins, headings, footings, page numbering, and generally the way a page is laid out. These DOT commands do not affect the screen display but do affect the way your document appears when it is printed. We will not lead you through the use of these commands, but will simply describe their use; then you can try them at your discretion.

The features controlled by DOT commands are already set to "default" values, the settings used most often. You do not have to enter any DOT commands into your document unless you want to format a document in a specific manner.

A DOT command consists of a period, a two-letter code, and possibly, a number or some other option. DOT commands must always begin in the first column of a line (column 1); WordStar ignores DOT commands that start in any other column (and usually considers it as text). Putting a period in the first column might sound funny—normally you would never place a period in the first column in normal text—but, like the PRINT CON-

TROL characters, these symbols appear on the screen but do not print.

When you enter a period in the first column, a question-mark FLAG appears in the last column to indicate that the Osborne 1 expects a DOT command. The left margin temporarily disengages while you enter the DOT command; if you try to move the cursor to another line before you complete a valid DOT command, the cursor will blink. So, a blinking cursor probably means that a period in the first column was not followed by a valid DOT command.

You can use most DOT commands as many times as you want and locate them anywhere in the file, as long as they begin in the first column. However, WordStar cannot show you where your pages end, unless the DOT commands that control page length and top and bottom margins are at the beginning of the file before any text. This ability to indicate where pages end is called the "dynamic page break display." All DOT commands are in the DOT HELP MENU (^J D) and are summarized in the Reference Guide. DOT commands direct vertical page layout, horizontal page layout, pagination, and so on. Let's examine some of them:

Vertical Page Layout

On daisywheel and thimble printers, you can space lines of text closer together or farther apart by changing the line height. The LINE HEIGHT DOT command, **.LH**, and a number set line spacing in 48ths of an inch on these special printers. The usual setting is 8/48ths which equals 1/6th of an inch, or six lines per inch. Use of this DOT command provides an alternative to the single, double, or triple spacing obtained with the PRINT-CONTROL command, **^OS**. After changing line height, you will have to adjust the margins or the paper to center the text vertically on the page.

LINE HEIGHTS

	Lines per Inch		**DOT Command Used**
	2.0		.LH 24
	2.4		.LH 20
	2.6		.LH 18
	3.0		.LH 16
	4.0		.LH 12
	4.8		.LH 10
	5.3		.LH 9
DEFAULT → →	6.0	→ → → → → →	.LH 8
	6.8		.LH 7
	8.0		.LH 6
	9.6		.LH 5

Paper length is expressed as the number of lines allowed on a page. The default WordStar paper length is 66 lines. You can change this number with the PAPER LENGTH DOT command, **.PL**. The number specified after **.PL** determines the number of lines on a page.

EXAMPLE: **.PL 55** ← **55 lines per page**

To change the number of blank lines reserved as a margin between the top of the page and the beginning of your text, use the TOP MARGIN DOT command, **.MT**. Change the default setting of three lines by specifying the desired number of lines following **.MT**.

EXAMPLE: **.MT 6** ← **6-line top margin**

Change the bottom margin with the BOTTOM MARGIN DOT command, **.MB**, followed by the desired number of lines. The bottom margin usually occupies eight lines. Note that the four vertical-layout DOT commands (**.LH**, **.PL**, **.MT**, **.MB**) just mentioned will properly affect the DYNAMIC PAGE BREAK DISPLAY only if these commands appear at the beginning of the file.

EXAMPLE: .MB 6 ← 6-line bottom margin

Two other DOT commands affect the vertical layout of a page. When you understand how headings and footings are specified, you may want to separate these headings and footings from the body of the text by a predetermined space. The HEADING MARGIN DOT command, **.HM**, or the FOOTING MARGIN command, **.FM**, followed by the desired number of lines will accomplish this formatting. Usually, two lines separate headings and footings from the text.

EXAMPLES: .HM 3 ← 3-line header margin
.FM 5 ← 5-line footer margin

Horizontal Page Layout

The page number usually prints in column 33 or the center column if you specify margins. Position the page number at any column at the bottom of the page with the PAGE NUMBER COLUMN DOT command, **.PC**, followed by the column number.

EXAMPLE: .PC 45 ←page number in column 45

The PAGE OFFSET command, **.PO**, and a number representing columns, sets the number of columns indented from the printer's left margin. This command offsets text from the tractor-feed holes at the left of the paper, allowing use of narrow paper. Usually eight columns of space are indented on each line before printing occurs.

EXAMPLE: .PO 12 ← indent text 12 spaces

Pagination

To indicate the point at which you want a new page to begin, use the UNCONDITIONAL PAGE DOT command, **.PA**, which forces the following text to begin at the top of a new page.

EXAMPLE: .PA ← start new page now

To prevent a new page from starting—as in the middle of a table—or to begin a new page if a section of text cannot fit on the current page, use the CONDITIONAL PAGE DOT command, **.CP**.

EXAMPLE: .CP 6 ← start new page if less than 6 lines left on page

You can specify a heading—a title at the top of each page—or a "footing" on each page. The TEXT HEADING DOT command, **.HE**, begins a line of text as the heading at the top of every page. You can change headings at any time with another **.HE**, but for a heading to print on the first page where it appears, it must precede all text for that page, including blank lines, if you have created any.

EXAMPLE: .HE This is a heading!

The TEXT FOOTING DOT command, **.FO**, specifies footings. Text on a line beginning with **.FO** will print at the bottom of the page.

EXAMPLE: .FO This is a footing!

Page numbers don't print when a footing is in effect. The character **#** can occur in either the **.HE** or the **.FO** command at the place where the page number should appear.

EXAMPLE: .HE This is a heading! Page # Here

You can use the PRINT-CONTROL command, **^K**, in the **.HE** or **.FO** lines to direct printing so that headings and page numbers can print on either side of the page depending on whether the page number is odd or even. This "alternating layout" is useful for creating long documents that have text on both sides of the page.

EXAMPLE: .FO ^KThis is an alternating footing! Page #

When you want to print the page number at the top of the page and no footing is in effect, you must use the OMIT PAGE NUMR DOT command, **.OP**, to prevent the number from occurring again at the bottom of the page. Use the **.OP** command at anytime to suppress page numbering. You don't have to use **.OP** when footing text is present, since it is automatically in effect in that circumstance.

EXAMPLE: .OP ← omits page numbers

WordStar usually numbers pages starting from page 1 in each file, but you can change the page number at any point in a file with the NUMBER PAGES DOT command, .PN. This command turns page numbering back ON following an **.OP** command. Page numbering continues from page 1 unless you specify a page number with **.PN**. The page numbers print at the bottom of the page unless you specify otherwise. The maximum possible number of pages is 65533, and page numbers shown on the STATUS LINE pertain only to the present file.

EXAMPLE: .PN 15 ← start numbering pages at 15

Special Controls

The character width can change on printers with programmable character widths—daisywheel and thimble printers. The CHARACTER WIDTH DOT command, **.CW**, is set in 120th-inch increments. The usual character width is 12/120ths, which equals ten characters per inch.

CHARACTER WIDTHS

	Character Pitch	DOT Command
	5 per inch	.CW 24
	6	.CW 20
	7	.CW 17
	8	.CW 15
DEFAULT →→	10 (pica) →→→→→	.CW 12
	12 (elite)	.CW 10
	15 (compressed)	.CW 8
	20	.CW 6
	24	.CW 5
	30	.CW 4

You specify the amount of platen roll used for producing subscripts and superscripts in 48th-inch increments. The SUB/SUPERSCRIPT ROLL DOT command is **.SR**. The default on daisywheel printers is 3/48ths, or **.SR 3**.

EXAMPLE: .SR 8 ← 1/6-inch roll

"Microjustification" is a feature that spreads letters evenly across a line when a document prints. This feature is usually ON, but you can turn it OFF with the ON/OFF MICROJUSTIFICATION DOT command, **.UJ**. When this feature is OFF, the text will print as it appears on the display, using spaces and carriage returns. Turning microjustification OFF might be useful to print columnar tables aligned as they appear on the screen.

EXAMPLE: .UJ ← toggle justification

Many printers have bidirectional printing capability; the printing continues even as the printhead is going from right to left to return to its original position. Depending on previous setting, the ON/OFF BIDIRECTIONAL PRINT DOT command, **.BP**, either enables the printer to print or prevents it from printing back and forth across the page. This feature is usually turned OFF to make WordStar work correctly with slow printers or with printers whose print tends to "wander" up and down when bidirectional printing is in effect.

EXAMPLE: **.BP** ← **toggle bidirectional print**

All the DOT commands you've learned so far change the "looks" of a document. Sometimes you may want to enter text you don't want to print, such as "comment lines." For inserting such material, use the IGNORE TEXT DOT command, **.IG**. Another method of typing a comment line is to use two periods in the left-hand margin, followed by your comment. This command, you may remember, was used to hide the imitation RULER LINE you created earlier.

EXAMPLES: **.IG** **This is a comment line—ignore it!**
.. **This is also a comment line**

Lesson 11: Managing Files

Manipulating Files

We hope that your printing was successful. If you had any difficulties, you might want to check the special Appendix on modifying WordStar in this manual, or check with your dealer.

Besides the PRINT command, the no-file menu provides a category of "file commands" used to manipulate files. These commands let you rename, copy, and delete the files of your choosing. You can either issue these file commands from the no-file menu or from within your editing session, using the **^K** prefix. Since we're at the no-file menu in our description of WordStar, now is an appropriate time to go through a simple exercise to show how to manipulate files.

Make sure WordStar is at the no-file menu display with the disk drive logged to drive B. Issue the FILE COPY command, O, and WordStar will ask:

```
NAME OF FILE TO COPY FROM?
```

WordStar seeks the name of the file to be copied. Enter the drive identifier and file name B:TRIAL.TXT and press RETURN. If this file is not on the diskette in drive B or if you make a mistake while entering the file name, this message will appear:

```
FILE B:"file name" NOT FOUND
```

This message tells you that WordStar can't find the file you specified. If you enter an invalid file name, WordStar displays the INVALID FILE NAME message.

Once you have successfully entered the file name and the file has been located on the specified drive, you will see this prompt:

```
NAME OF FILE TO COPY TO?
```

This prompt requires that you supply the name of a file where you want the copy sent. Enter the name COPY.TXT and press RETURN. WordStar will create a file named **COPY.TXT**, and store an exact replica of the information from file **TRIAL.TXT** within it. If a file called **COPY.TXT** already existed on the specified drive, the message

```
"file name" ALREADY EXISTS OVERWRITE (Y/N)?
```

would appear. This message forces you to consider the ramifications of destroying the information already in the file specified and replacing it with a copy of another file.

Now that you have two trial files, **TRIAL.TXT** and **COPY.TXT**, delete one of them. Delete **TRIAL.TXT** by using the DELETE FILE command, Y, from the no-file menu. After you issue **Y**, WordStar will ask:

```
NAME OF FILE TO DELETE?
```

Enter B : followed by the file name TRIAL.TXT and press RETURN to delete the trial text file. Observe that the file **TRIAL.TXT** has, in fact, been deleted from the file directory.

You still have a copy of the information that was in **TRIAL.TXT**. Complete the circle by renaming the copy using the original name. Issue the no-file RENAME command, E. WordStar will ask:

```
NAME OF FILE TO RENAME?
```

Enter the name of the file you wish to rename—in this case, COPY.TXT—and press RETURN. WordStar will then ask:

```
NEW NAME?
```

Enter the new name you want assigned to the file. Since this file contains your trial text, change it back to the name that best reflects its contents: TRIAL.TXT. The name **COPY.TXT** should disappear from the directory, replaced with **TRIAL.TXT**. If you did everything correctly, the new **TRIAL.TXT** file should be exactly the same as the original one.

You may not envision any uses for these file commands now, but their value will become apparent as you start working with a lot of files. A typical use for the DELETE FILE command would be to delete backup (.BAK) files when you needed to free up more diskette space. You can accomplish the same task while editing a document, using the **^KJ** command.

WordStar File Size and Diskette Space

Even though WordStar can handle large document files, keep files as small as possible. WordStar requires approximately three times the space occupied by a file if you are to properly edit it. Therefore, if you create a file on one-third of a diskette, the other two-thirds need to be left blank to enable you to safely edit the original file.

When you edit a file, the diskette retains a copy of the file in its original form as a backup; the file is of type **BAK**. Another copy of the file contains the new version of the file created during the most recent editing process. WordStar maintains yet another copy of the file (of type $$$) when you go to the end of the document, then go back to the beginning to reedit.

If you run out of space on your diskette while editing a file, WordStar will inform you of this fact with an error message. It may be possible to move the cursor forward in your document and still salvage your file. If a **DISK FULL** error message occurs while you are saving a file, however, you may lose the current contents of the file. Double-density owners should be able to get up to 80 single-spaced pages onto a diskette, so 30 or 35 pages should be the cut-off point for a working file.

You must keep track of file size and diskette space to prevent the loss of data. WITH SINGLE-DENSITY DISKETTES, YOU SHOULD KEEP FILES UNDER FIVE PAGES LONG, AND UNDER NO CIRCUMSTANCES CREATE A DOCUMENT LONGER THAN 15 SINGLE-SPACED PAGES, if you want to ensure that you'll always be able to save your edited document.

You can use the DELETE FILE command to eliminate backup files when you are editing a series of files on the same diskette. In fact, you can erase any unimportant or otherwise restorable file from the diskette to make more room. This is another reason why you should always make copies of diskettes whenever you change any information stored on them.

An easy method for keeping track of diskette space involves using the CP/M **XDIR** program. This program resides on your WordStar diskette and provides information on specified files and the space they occupy.

To examine how much space is left on your data diskette, make sure that you are at the no-file menu and issue R for RUN A PROGRAM. WordStar then displays:

COMMAND?

If you are logged onto drive B, issue:

A : X D I R SPACE BAR B :

and press RETURN. On the other hand, if you are logged onto drive A, issue:

X D I R SPACE BAR B :

and press RETURN. WordStar will then "run" the **XDIR** program which shows how much space has been used and how much room remains, then waits for you to press any key before returning to WordStar.

Another method for keeping track of a file's size is to turn OFF the PAGE BREAK DISPLAY command by issuing an ^O P . If you're at the end of the file, you'll see the number of bytes currently occupied by the file you are editing at the top of screen immediately following **FC=** ("file count"). Remember, a single-density diskette contains 92,000 bytes.

Lesson 12: MailMerge

Congratulations! You are now an initiated WordStar user and know how to prepare documents and print them. The MailMerge enhancement lets you use your newly gained skills to merge the contents of several files, including data files during printing.

MailMerge provides an easy method of preparing form letters, envelopes, and mailing lists. Insertion of "boilerplate" text and successive printing of multiple files allows further flexibility.

Also, MailMerge lets you print a document with different line spacing, margins, and justification than were specified at the file's creation.

All of these features result from a combination of DOT commands. You've seen some of them; others, unique to MailMerge, will follow, with explanations and demonstrations.

Preparing a Form Letter

Each copy of a form letter conveys the same information, while acknowledging each person individually. The first step in preparing a form letter is to create a document file. Prepare a file for the form letter that contains the information as it will appear for all the letters. As you create this master letter, identify the parts that will vary for each copy—such as names and addresses—by substituting each of these variable pieces of information with an ambiguous "keyword" (a special name).

Keywords replace text that will change from one letter to the next when the master form-letter file later prints. MicroPro calls this process "Merge-Printing," which simply means that you use the special print commands embedded in MailMerge instead of the print commands in WordStar.

A keyword can be up to 40 characters long. You can use the same keyword more than once in a document, as long as it represents the same piece of information each time. To identify a keyword in your text, place an ampersand (&) on both sides of the keyword. When the file containing your form letter prints, the variable information you want inserted at that point replaces the ampersand-enclosed keyword.

KEYWORD EXAMPLE: Now is the time for all &TYPE& people to come to the aid of their party.

&TYPE& varies with each printing of the document.

You can hold information to be inserted for keywords in a separate "data file" or enter it from the keyboard as you print out a document. We will show you how information is supplied for keywords, but first you should prepare a master form-letter file. Create a file named **FORMLET**; you should know how to do this by now. Type the form letter, including keywords illustrated in the following example, and don't forget to include the **.PA** DOT command at the end:

```
&NAME&
&COMPANY&
&STREET&
&CITYSTATE& &ZIPCODE&


DEAR &NAME&:

We at Our Company are extremely happy to allow you
the honor of being one of the first to know about our
new product. Only our most endeared customers are
privileged enough to partake of this once-in-a-lifetime
opportunity. Since &COMPANY& has been such a loyal
customer and because our solid business line is so com-
petitive, we can afford to make you this offer. Thanks to
you, our ability to provide prompt and low-cost service
is possible.

Sincerely yours,
Bill Owner
Title

.PA
```

This sample illustrates how to prepare a master copy of a form letter. An unlimited number of letters with a personal touch can derive from this one master form.

Once the master copy of the form letter exists, you must augment it with certain strategically placed DOT commands. The CONDITIONAL PAGE DOT command, **.PA**, at the bottom of the form letter is one such command. You may have been surprised when you entered this command because the PAGE BREAK DISPLAY (dotted line) automatically appeared on the next line. The appearance of this dotted page-break line is normal since the **.PA** command signifies the end of a page. The **.PA** command makes sure that a new page starts and only one copy of the form letter per sheet of paper prints; meaning the printer advances to the top of the next piece of paper before printing the next form letter.

Other necessary DOT commands tell the computer where variables—such as names and addresses—will come from when the form-letter file is processed for printing. To work correctly, these DOT commands must be at the beginning of the file. Place the cursor at the beginning of the form-letter file and insert four blank lines by issuing ^N four times. If necessary, use ^E to move the cursor to the top of the file, then insert the following DOT commands, pressing RETURN after each one:

```
.. FORMLET Comment identifies file:
. OP
. DF DATAFYL
. RV NAME, COMPANY, STREET, CITYSTATE, ZIPCODE
```

The top four lines in your file should look exactly as shown above. The COMMENT DOT command consists of two periods and indicates an unprinted comment. This comment serves as a reminder, naming the file you are working on. The DOT command **.OP** turns off the page numbering so a number will not print at the bottom of your letter. For a document longer than one page, you can number each page by using the NUMBER PAGES DOT command, **.PN**.

The DATA FILE DOT command, **.DF**, specifies the name of the data file where information to be inserted for the keywords is stored. Our sample form letter includes five keywords representing the name, company, street, city and state, and zip code.

Each time the master form-letter file is processed for printing, MailMerge extracts five values from a data file (explained later) and inserts them in place of the keywords.

The READ VARIABLE DOT command, **.RV**, lists keywords used in the file, in the same order in which they appear in the text. The order in which the keywords are listed in the **.RV** command must also correspond to the order that information is listed in the data file. All of this probably sounds confusing if you have never done any programming, but things will begin to make sense as you work your way through the examples.

Now that you have prepared the form letter, you need to create a data file so that you have something to merge-print. For now, save your document file, **FORMLET**, using the ^K D command.

Data Files

The CREATE A NON-DOCUMENT FILE command, **N**, from the no-file menu is used to create data files. Using **N** to create a file is normally reserved for writing source programs, but is also well suited for creating MailMerge data files. The reason for creating a data file with the **N** option rather than the **D** (document) option is because you don't want WordStar to use any special "soft hyphens" or other word-processing features in your data file. Such features interfere with the data file's highly structured organization. Issue N. When the NAME OF FILE? prompt appears, supply the name DATAFYL and press RETURN. Recall that **DATAFYL** was the name of the file you specified with the **.DF** command in your form-letter file.

A data file consists of lines of information called "records." Each record contains the information necessary to match all of the keywords for one form letter printed from the master. Information is organized within the record lines in the order that the corresponding keywords are arranged in the form-letter file and listed in the **.RV** command.

Each data item supplied for a keyword must be separated from the next by a comma. If the information itself contains a comma, you must enclose the entire clause within quotation marks. Each record line of information for one printed copy of a document, such as a form letter, is "delimited" (ended, or "set off" from the next) with a RETURN. Enter the record lines shown below and remember to end each line by pressing RETURN:

```
Mary Maid,Dairy Queen,411 E 32nd,Boston MA,02174
Helen Mellon,H-Spa,123 Main St.,"Troy,MI",27719
Had Jolly,Goodtime Saloon,,"Nome,AK",00020
```

The data file you just created contains enough information to print three copies from the master form letter. Notice that two of the city-and-state "fields" contain commas and are surrounded by quotes. Also use quotes if you want a blank space before or after a data item:

```
" Space Cadet",Wizardry Inc., Upthere St.,Home,CA,00000
 ^                            ^
space will appear          space won't appear
```

You can omit a data item such as the street address as illustrated in the third record of the example. Leaving an entry blank will cause no problems, as long as you remember to substitute the empty field with a comma. The comma is needed to keep the data selection in "sync" (properly arranged). The missing item will cause a blank at the location of the corresponding keyword. Later in this lesson, we will show you how to keep the blank from appearing in your letter.

Inspect your data file and confirm correct entry of each item in all three records. Now save your data file with the ^K D command and return to the no-file menu.

NOTE

The primary difference between editing a "document" file and editing a "nondocument" file, as you just did, is that WordStar will not insert any special characters on its own or perform any "word wrap," hyphenation, or margin justification. You should note, however, that except for these fancy word-processing functions, all the WordStar commands still function as before.

Merge-Printing a Form Letter

At this point, you should have two new files: one named **FORMLET**, containing your form letter; and another, holding your data records, named **DATAFYL**. It is time to reap the fruits of your labor and perform your first MailMerge operation.

To initiate a MailMerge operation, issue the MERGE-PRINT command, M, from the no-file menu. WordStar will ask you for the name of the file to merge-print:

```
NAME OF FILE TO MERGE PRINT?
```

Enter the name of the file, in this case FORMLET, and press RETURN. As with the regular PRINT command, you'll be provided several options to use during the printing session.

To make things simple, you should probably just press RETURN for each of these options. If you are not using paper that feeds continuously, however, you may want to answer **Y** to the PAUSE BETWEEN PAGES? question. Answering YES to this question halts printing after each page so you can insert individual sheets of paper. Also, if you wish to make more than one copy of each letter, specify the number of copies in response to the NUMBER OF COPIES? question. By now you should know what to do when you see the Ready printer, press RETURN: message. That's right! Press RETURN to start merge-printing.

The message P = STOP PRINT displays in a box in the middle of your screen. You can stop the merge-print operation at any time by issuing the **P** command. You would be prompted with the usual options: Y for abandon printing, N for continue printing, and U to temporarily interrupt the printing. Error and other messages appear below the printing message, and if there is room, the current file directory is shown.

If you did everything correctly, three form letters should have printed. Did your screen display a message preceded by three asterisks? If so, you made a mistake either in the form-letter file, or in the data file. Refer to the error messages in the Reference Guide for a definition of any error message displayed on your screen. Correct your mistakes, then try again.

The text in merge-printed files automatically reformats after each substitution of a data item for a keyword. Look at the form letters you just printed and note that the surrounding text has accommodated the company names perfectly. Even though data items vary in length, a feature called "print-time line forming" monitors these variable lengths and adjusts your text for them. We'll explore the subject of print-time formatting and ways to direct it in depth at the end of this chapter.

Envelopes

You now have three form letters. Wouldn't it be nice to have three addressed envelopes in which to send them? You can print the names and addresses stored in a data file on envelopes by creating a "command file" that specifies the formatting. A command file usually contains no text and only directs formatting and merging of files. Create such a command file with the name ENVELOPE using the [D] option from the no-file menu. You'll use this envelope command file only to format and position the names and addresses on envelopes.

Constructing a command file that directs printing of names and addresses onto envelopes requires more attention to detail than did our form-letter file. You have to use the PAGE LENGTH

DOT command, **.PL**, to specify the size of the envelope to the printer. Also, you must suppress the top and bottom margins so the address will be centered on the envelope.

The **.DF** in the following example tells the computer which data file contains the information to be printed. The **.RV** command lists the keywords and establishes the order in which each data item will print. DOT commands at the beginning of a command file specifying the format for a standard letter-size envelope should look like this:

```
..ENVELOPE
.OP
.PL 26
.MT 12
.MB 0
.DF DATAFYL
.RV NAME, COMPANY, STREET, CITYSTATE, ZIPCODE
^C

      &NAME&
      &COMPANY&
      &STREET&
      &CITYSTATE&
      &ZIPCODE&

.PA
```

This is a rudimentary example of a command file which references a data file to address envelopes. Examine the structure of this envelope file for a moment. The first line is a comment, as indicated by the double periods. These ignored comments help identify the file and are not needed. The page length is set to 26 lines—the size of a standard business letter envelope. Using the **.PL** command, you can set the page length to match any size envelopes you want. The top margin in the example is set to 12 (**.MT 12**) and the bottom margin is suppressed (**.MB 0**). The names and addresses that will print on each envelope will come from the data file named **DATAFYL** as specified by the **.DF** command. You need the **.RV** to define the order in which the data items will be extracted from the data file and printed in place of keywords.

Printing will have to stop after every envelope so you can insert a new one. You can stop printing after each page by answering **Y** to the PAUSE BETWEEN PAGES? question, but an easier way to stop the printer is to put a **^C** at the end of the DOT commands. Enter **^C** by first issuing a **^P**, then pressing **C**. Each time **^C** crops up, the printing will pause. After you've put the next envelope in place, use the **P** command to start printing the next envelope.

Save this file with ^K D, which will bring you back to the no-file menu. Merge-print the envelope file by initiating Merge-Print with M, and supplying the name ENVELOPE and pressing the ESC (for escape) key. You will have to position the envelopes in the printer by hand, so you will need to experiment a bit before you achieve proper alignment.

NOTE

The ESC key is a shorthand way of telling WordStar that you don't want to specify any of the print options. You can use it to avoid the list of questions that appears after you supply the name of the file you want printed. Remember: you usually press the RETURN key.

By slightly changing the DOT commands in your envelope file, you can create a nicely formatted mailing list. A mailing-list command file would not need to suppress the margins or modify the page length; you would leave out the **.PL**, **.MT**, and **.MB** DOT commands. You will still want to make use of the CONDITIONAL PAGE DOT command **.CP**, however.

A number after the **.CP** command will cause a new page to start if fewer than that number of lines are left on a page. Since five keywords represent five data items, entering **.CP 5** will ensure that no name and address will be split between pages. Set up a mailing-list command file as illustrated below:

```
..MAILIST
.DF DATAFYL
.RV NAME, COMPANY, STREET, CITYSTATE, ZIPCODE
.CP 5

&NAME&
&COMPANY&
&STREET&
&CITYSTATE&
&ZIPCODE&
```

then merge-print it.

Is that blank line left in the third address starting to bother you? A missing data item, such as the street address in the third record of your data file, will usually cause a blank line to appear at the position of the corresponding keyword. Remember that you used a comma in the data file to signify a missing data item. WordStar allows you to get rid of this blank line if you wish.

Place a slash (**/**) and the letter **O** after the keyword within the ampersands if you suspect that a piece of information may sometimes be unavailable. The **O**—which stands for omit, will prevent the blank space from appearing in your list when the file prints. For a keyword using **/O** to be effective, nothing else can be on the same line, and a comma must hold a place for the missing item in the data file. Here is how the keywords in a mailing list that accommodates possible missing items look:

```
&NAME/O&
&COMPANY/O&
&STREET/O&
&CITYSTATE/O&
&ZIPCODE/O&
```

Establishing a Constant Data Item

Sometimes you may have a piece of information that can be changed but that you don't want to list in every record of the data file. For example, if your company is really small you might have more than one title reflecting your various duties and would like to easily switch from one to the other. Reedit your form-letter file, **FORMLET**, so we can demonstrate how to do this. Press [D] from the no-file menu and supply FORMLET as the file name.

The SET VARIABLE DOT command, **.SV**, establishes a piece of information (data item) that remains constant and serves as a substitute for a specific keyword. The information that is set can be up to 200 characters long and is normally located at the beginning of the file.

At the beginning of the file, enter:

```
.SV TITLE, Vice Jack of all trades
```

Replace the words "Vice Jack of all trades" with one of your choosing if you like; place this command beneath the existing comment command. Next, change the occurrence of the word "Title," under the name Bill Owner, to a keyword by surrounding it with ampersands. The title Vice jack of all trades, or one you specify in **.SV** will print at the position of the keyword **&Title&**. This shortcut makes it unnecessary to enter a particular title for each letter.

An even more practical example of the use of **.SV** would be in preparing a standard contract between two parties. You could prepare the body of the contract as you did in the form letter, using keywords in place of the party names. Then you could place two **.SV** commands at the top of this contract file; one for each party.

Since you would use the name of each individual many times in the contract, you could use **.SV** to specify the names of the parties involved, just once, at the top of the contract and have them print for all the associated keywords. In this way, you could use the master contract many times with many different individuals. Standard will preparation is one common use of this feature.

An example of this standard contract idea might look like this:

```
.SV PARTY1, John Doe
.SV PARTY2, Ruth Smith
```

Each time &PARTY1& is encountered within the contract, the name John Doe is printed. Every time &PARTY2& is encountered, Ruth Smith is printed.

Inserting Files

Summoning a document file for merge-printing from within another file is simple. The FILE INSERTION DOT command, **.FI**, invokes processing of a named file from within another file. This file-insertion capability allows you to insert boilerplate text that has been stored in a separate file or to print several files in succession. Not only can you accomplish these very fundamental tasks, but, the more experience you gain, the more complex uses you will discover. We will describe the basic file-insertion techniques and allude to some of the other capabilities in the remainder of this chapter.

The name of the file to be inserted and processed is specified following **.FI**, which is located at the point in the main file where processing is to begin. When MailMerge encounters **.FI**, it temporarily interrupts merge-printing of the main file while it processes the file named by **.FI**.

Files processed using **.FI** are treated as though the MERGE-PRINT command invoked them. Processing of the inserted file may even include further merging of referenced data files or even another file insertion. After processing of the inserted file is finished, merge-printing of the main file continues with the line following **.FI**.

Inserting frequently used text that's stored in a separate file is one of the most obvious uses of the file-insertion option. You can prepare a boilerplate file by using the BLOCK WRITE command to save a block of text in its own file, or by creating a file and typing the appropriate text.

A file containing boilerplate text must end with a RETURN, which the flag character, <, indicates. This flag should be present below the last line of text in the boilerplate file. Inserted files that do not end with RETURN will blend with the next sentence of the main file, usually resulting in serious complications.

To demonstrate this type of insertion, create a file called POSTCRPT. Type anything you wish in this file, but remember to press RETURN after the last line of text; then save it. In reality, this example probably does not reflect a practical application of a boilerplate insertion; but this demonstration will give you a chance to see how these insertions are handled.

Next, reedit the file named **FORMLET** and place the cursor after the last sentence in the master form letter.

With the cursor resting below the body of the form letter and before the salutation, press RETURN and issue ^N. On the empty line you have created, enter:

```
.FI POSTCRPT
```

then save the file. Now merge-print the form-letter file. The text you saved in the file named **POSTCRPT** should print in each form letter at the location specified by **.FI**.

Printing Multiple Files

You may find, when creating a large document, that text is spread out over many files. Printing the document one file at a time is painstaking. Using file insertions, you can reference each file to be printed in its proper context from a command file. Usually a command file that references text files in sequence is set up like this:

```
.FI FILE1
.FI FILE2
.FI FILE3
```

The following exercise demonstrates the concept of processing one file after another. Create a command file, named COMBINE that has an **.FI** and a file name on each line—use the names of your form letter, envelope, and mailing list files. This command file will process your form letters, address the envelopes to send them in, and then print a list of the letter's recipients.

If you process this file in its present state, you will probably have a difficult time keeping track of which file is being processed. To make it easier to see what is going on, document the progression of multiple file insertions with messages.

The DISPLAY MESSAGE DOT command, **.DM**, causes display of a specified message on the screen. So you will know when each file is being processed, add some message prompts to your command file. You will also want to make use of the CONDITIONAL PAGE DOT command, **.PA**, to ensure that a new page starts after every file insertion. Set up your command file in the following way:

```
.DM The form letters are being printed.
.FI FORMLET
.PA
.DM WordStar is now typing envelopes for you.
.FI ENVELOPE
.PA
.DM The mailing list is now being prepared.
.FI MAILIST
```

then save it and merge-print it. (Note: it is not really necessary to insert envelopes to observe this task—plain paper will do fine.)

As you can see, this is a simple, yet effective, method of keeping track of what WordStar is doing when it merge-prints files.

Changing Diskettes

If the data file being referenced is on another diskette, or if you're printing a large document such as a book, you can specify a diskette change. To accommodate a diskette change, place the word **CHANGE** after the file name in either the **.FI** or **.DF** command. A **.DF** or **.FI** with the word **CHANGE** after it displays a message prompting a diskette change, and processing stops until the diskette change occurs.

SPECIAL NOTE

On the Osborne 1—with only 92K of data-storage capacity—you'll find the **CHANGE** *command extremely helpful. The early versions of this manual were created in this fashion.*

The name of the file and the drive where it is expected are indicated by a message on the screen. When this message appears, you know that a diskette change is required. Remove the diskette from drive B, and replace it with one holding the file being summoned, and press **RETURN**. If you like, use **.DM** to display a more explicit message prompting for a diskette change. After you press **RETURN**, the Osborne 1 will search for the named file on the diskette you've inserted. If the file is not on either diskette, a message asking for the correct diskette is displayed until the specified file is loaded.

The diskette that holds the command file can be removed while the diskette containing the referenced file is processed. When processing finishes, a message asking for the original diskette is displayed. Reinsert the diskette holding the command file and press **RETURN** to continue the merge-print operation.

Never specify a diskette change for the drive containing the WordStar program files (normally drive A). Diskette changes are limited to the B drive on the Osborne 1. Accidental specification of a diskette change for the drive holding the program files displays the message Cannot change diskettes in drive B. MailMerge will try to access the referenced file on both drives in case there is a mistaken drive identifier. If MailMerge can't find the file on either drive, it displays a message to this effect and ignores the **.FI** or **.DF** requesting the invalid change.

If you are initiating a merge-print from the A drive, place **B:** before the file name in the **.FI** or **.DF** command. The drive identifier is not necessary if you are already logged on drive B and the files to be used are also on that drive.

Entering Data from the Keyboard

Earlier we showed you how to store data to be supplied for keywords in a data file. An alternative method, which involves supplying data from the keyboard, is also possible. The screen can request data, and when supplied, inserts it for keywords in the file being merge-printed. Instead of storing information in a data file, use the ASK VARIABLE DOT command, **.AV**, to ask for the data. The **.AV** command replaces the **.DF** and **.RV** commands that are needed when information is stored in a data file.

A separate **.AV** is needed for every keyword. Each **.AV** is followed by a keyword and placed one per line, in the same order that data for the keywords is inserted when the file is merge-printed. Recall that when using a data file, you need to establish the order of data insertion for keywords with **.RV**. Let's go through an exercise on entering data from the keyboard. Once again edit the file named **FORMLET**.

Place the cursor at the beginning of your form-letter file, issue ^X two times to move down two lines, and delete the **.DF** and **.RV** commands by issuing ^Y twice. Issue a ^N to give yourself a blank line to work in, and enter a separate .AV and RETURN for each keyword in the name and address. This is how the top eight lines of your file should appear:

```
..FORMLET that asks for variables
.SV YOURCOMPANY, CompanyName
.OP
.AV NAME
.AV COMPANY
.AV STREET
.AV CITYSTATE
.AV ZIPCODE
```

Be sure the **.DF** and **.RV** commands have been deleted, then save this file and merge-print it. The disk drives will activate as usual but no printing will take place. Instead, the keyword listed in the first **.AV** will display on the screen, followed by a question mark. Respond by entering the data that you want printed for that particular keyword and press RETURN.

Once you press RETURN, the next keyword will be prompted, and so on until you supply the data for all the **.AV** commands in the file. A copy of the form letter with the name and address you entered at the keyboard will then print.

When dealing with many files, you may want a more explicit message than just the keyword displayed on the screen. You can specify any message you like in the **.AV** command to ask for data. To specify a prompt other than the keyword, follow **.AV** with the desired message enclosed in quotation marks. Then place a comma and the actual keyword after the message. Open your form-letter file to try this.

Move the cursor to the end of the first sentence in your form letter, delete the last three words, our new product, and replace this clause with the keyword &PRODUCT&. Now move the cursor below the currently existing DOT commands and enter these two commands:

```
.AV "Enter the name of the product:", PRODUCT, 10
.CS
```

The prompt message and the data entered for it cannot extend beyond one line or a portion will be cut off. To avoid this problem, you might limit the number of characters MailMerge can accept for an **.AV** prompt. To restrict the number of characters accepted from the keyboard, we have placed the number **10** at the end of our **.AV** command. A number at the end of an **.AV** command represents the maximum number of characters accepted in response to an **.AV** prompt. This feature is used as a data check, or to limit text entry to fit a particular format. No extra characters over the specified limit will display or print.

As its name implies, the CLEAR SCREEN DOT command, **.CS**, causes the screen to clear any information that has accumulated there. It is a good idea to use this command to clear the screen whenever you are prompting for data with a series of messages.

Note that when data is entered from the keyboard instead of being stored in a data file, the file is processed only once. You have available several ways to process the file more than once without going through the whole print cycle each time. One way is to specify the number of times you want the file processed and thus the number of printed copies with the NUMBER OF COPIES option offered when merge-print is initiated. You can also have a file insert itself through the insertion command. Placing **.FI** and the file's name (**FORMLET**) at the end of the form-letter file will, in effect, trick the file into inserting itself over and over again.

Save your form-letter file and merge-print this new version. The last prompt on the screen before the file prints should look like this:

```
Enter the name of the product:
```

The entry in response to this message will print in place of the keyword &PRODUCT&. To see that any characters over the specified limit will in fact be ignored and won't display on the screen, try entering more than ten characters. A copy of the form letter will print and the screen will request data for the next letter.

Should a data item be the same as in the previous letter, you could issue **^R** to restore the same answer you used the last time MailMerge asked the question. The previous answer is displayed on the screen, where you can edit it if necessary. Press RETURN to enter the data as you would for any input.

Generally, you can halt merge-printing of your form-letter file by pressing **P**, but not when MailMerge expects data for an **.AV** prompt. If an **.AV** command expects data, that's how MailMerge will interpret the STOP PRINT **P** command. To circumvent this problem, supply the data necessary for the prompt, then press RETURN and P in rapid succession. The **P** command will issue before you get to the next **.AV** command and will keep the next document from printing.

Before you try any of the examples in the remainder of this chapter, convert the form-letter file so it once again contains **.DF** and **.RV** to reference the data file. Edit your form-letter file and eliminate the **.AV** commands you used to ask for the name and address.

Replace the **.AV** commands by first entering .DF DATAFYL and pressing RETURN. Then, on the next line enter:

```
.RV NAME, COMPANY,STREET, CITYSTATE, ZIPCODE
```

Leave the **.AV** command that asks for the product alone. The final version of your form letter should look like this:

```
..FORMLET
.SV TITLE, Vice Jack of All Trades
.OP
.DF DATAFYL
.RV NAME, COMPANY, STREET, CITYSTATE, ZIPCODE
.AV "Enter the name of the product:", PRODUCT, 10

    &NAME&
    &COMPANY&
    &STREET&
    &CITYSTATE&
    &ZIPCODE&

    Dear &NAME&:

    We at our company are extremely happy to allow you
    the honor of being one of the first to know about
    &PRODUCT&. Only our most endeared customers are
    privileged enough to partake of this once-in-a-lifetime
    opportunity. Since &COMPANY& has been such a loyal
    customer and because our solid business line is so
    competitive, we can afford to make you this offer. Thanks
    to you, our ability to provide prompt and low-cost service
    is possible.

.DM The POSTCRPT file is now being processed
.FI POSTCRPT

Sincerely yours,
Bill Owner
&Title&

.PA
```

Advanced Command Files

Once you know how to insert files with **.FI** and enter data at the keyboard with **.AV**, you can combine these commands. Together, they can structure many imaginative merge-printing operations. Consider a command file that uses **.AV** to ask for the date and that uses **.FI** to insert the file containing the keyword **&DATE&**. A command file that asks for today's date and then merge-prints a file using the supplied date is set up so:

```
.AV "Enter today's date :", DATE
.FI FORMLET
```

For this command file to work correctly, you must put the keyword **&DATE&** at the place in the inserted file where the date should appear. Explore this example and those that follow at your own speed. Insert the keyword **&DATE&** in your form-letter file, then create and merge-print the command file illustrated above and observe what happens.

A whole new realm of possibilities opens up when you consider that you can use keywords within the DOT commands themselves. Say you want to maintain a given document file that can accommodate different data files. You can do it by going back and changing the name of the data file listed in the **.DF** command every time you want to use a different data file. Alternatively, you can create a command file that combines the **.AV** and **.FI** commands so you can enter the name of the data file at the keyboard.

Prepare a command file using separate **.AV**s to ask for the name of the document and data files to merge-print. The same keyword used in the **.AV** asking for the file to merge-print is what **.FI** uses in the command file. **.DF** uses the **.AV** keyword in the document file to be referenced. Here's how to set up such a file:

```
.. Command file for entering file name from keyboard
.AV "Enter name of file to print:", KEYWORD1
.AV "Enter name of data file:", KEYWORD2
.FI &KEYWORD1&
```

A command file set up like this will work only if the **.DF** command in the document file uses **&KEYWORD2&**. **.FI** will insert the name of the document file entered for the first **.AV** for processing. The name of the data file entered for the second **.AV** will be inserted for the keyword in the **.DF** command.

It is possible to place the **.DF** DOT command within the command file to allow reading of the data file from the command file. Also, it becomes unnecessary for the keyword listed in the document file to match the keyword in the file inserted.

A document file that uses a data file referenced from a command file needs the REPEAT PROCESSING DOT command, **.RP**, which causes reprocessing of the file until all data in the data file is used. Without the **.RP** command, processing stops after the first round. Here is an example of a command file that contains **.DF** to reference the data file:

```
.DM This command file can reference any data file
.AV "Enter name of file to merge-print", KEYWORD1
.AV "Enter name of data file", KEYWORD2
```

This information will be printed out before the file &KEYWORD1& is printed using data in the file &KEYWORD2&.

```
.FI &KEYWORD1&
.DF &KEYWORD2&
```

When a command file such as the one above merge-prints, the name of the document file and the data file name are requested. The named data file supplies data for variables in the document file. Files referenced in this manner must contain the **.RP** command (i.e., the &KEYWORD1& file must contain a **.RP**). A number following **.RP** causes processing of the file that many times.

A command file can process more than one file, providing the same data file is used for each document. For example, the following command file would print the form letters, then use the same data file to print the envelopes:

```
.DM Prints form letters and envelopes
.AV "Enter name of file to merge-print",KEYWORD1
.AV "Enter name of data file",KEYWORD2
.FI &KEYWORD1&
.FI ENVELOPE
```

The preceding examples are only a few of the possible command files you can structure by combining commands and using keywords. As you become used to these MailMerge DOT commands, you will find that with a bit of forethought you can handle most applications.

MailMerge Formatting Commands

Recall how text surrounding keywords shifted to make room for data of varying lengths. Text in files to be merge-printed automatically reformats following every insertion of a data item for a keyword. Consider this automatic text formatting when preparing files for merge-printing. Data printed for keywords sometimes exceeds the length of the ampersand-enclosed keyword, so a printed document might look quite different than it did when you created it.

Sometimes automatic reformatting may cause your document to print in an undesirable manner. For instance, consider a line ending with a RETURN, where a long data item replaces a keyword. The long insertion might cause text to extend beyond the point you intended.

One way to accommodate an expected long data item is to place spaces on both sides of the keyword. For example, enter the keyword **&PRODUCT&** where a long description that would alter the format is required. Advanced users may want to further control the way documents with long-data-item's format.

Formatting is the province of the PRINT-TIME LINE FORMER. The line former consists of two functions: the "input scanner" and the "output formatter."

The input scanner examines text in the file being merge-printed and locates the keywords. It takes margins, line spacing, justification, and word wrap into account. When the input scanner detects insertion of data for a keyword, it directs the output formatter to reorganize text: the scanner interprets the DOT commands and relays them to the output formatter, which performs the automatic print formatting.

The output formatter does not change text until the input scanner encounters a keyword. After insertion of the data item, lines of text reformat until to the next carriage return, line feed, form feed, or end of file is reached. This type of line forming is termed "discretionary" line forming. The PRINT-TIME LINE FORMING DOT command, **.PF**, can turn line forming ON or OFF.

Whether a keyword turns line forming ON or you activate it by entering **.PF ON:**, you can direct formatting in a variety of ways. Left and right margins, line spacing, and justification are all subject to the control of MailMerge formatting commands. You can either turn these features ON or OFF, or leave them to the discretion of the input scanner and output formatter.

Printing a document with different margins than those specified at the file's creation requires use of the RIGHT MARGIN DOT command, **.RM**, and the LEFT MARGIN DOT command, **.LM**. Line forming has to be ON for these margin commands to be in effect. You already know how to set the margins for a whole document, but if you want to change the margins for a particular section of a file, use one of these margin commands.

Suppose you want to indent a specific paragraph with a left margin of 20 and right margin of 40. First, turn ON line forming at the beginning of the paragraph. Next, specify the left and right margins. After the intended section of text is formatted, you have to direct line forming to act in its normal manner. These formatting commands would appear like this in your file:

```
.PF ON:
.LM 20
.RM 40
```

These commands would be placed before the portion of the file to be affected. The PRINT-TIME LINE FORMER would be returned to its discretionary setting by the .PF DIS: command located after the affected paragraph as illustrated here.

```
.PF DIS:
```

Margins can change in the middle of a paragraph: place the cursor at the beginning of the line above which you'll use the DOT command. Issue ^N, and enter the desired command. A soft RETURN will precede the command; a hard RETURN will follow it. MailMerge won't interpret the hard RETURN as a paragraph terminator. When detected, the DOT command will be invoked at the specified point in the file. If, subsequently, you need to use the REFORM command on the text containing the mid-paragraph command, you'll have to remove the DOT command. If the line spacing or justification for a specific portion of text has to change, follow the same procedure as for changing margins.

You can also use a command file to change the line spacing or some other format for an entire file. All you have to do is create a command file that turns line forming ON, specify the formatting, and insert the file to be formatted. Below is an example of a command file that would cause a file to print double-spaced, even though the file was created with single spacing. Use the LINE SPACING DOT command, **.LS**, in this case, and set the command file up like this:

```
.PF ON:
.LS 2
.FI POSTCRPT
```

To cause text to print with a ragged-right margin, even though justification is ON, use the OUTPUT JUSTIFICATION DOT command, **.OJ**. In the same vein, the INPUT JUSTIFICATION DOT command, **.IJ**, can direct the input scanner's interpretations. **.IJ** is not meant to format your files, but is there in case some unusual text confuses the input scanner. Try some of these commands on your file named POSTCRPT. Look in the Reference Guide for a complete list of these MailMerge formatting commands.

Another Section Completed

You've completed another chapter and learned almost everything in both WordStar and MailMerge. Through the use of examples, you've attempted practical applications of every command.

After one reading of this chapter, you probably won't remember everything about WordStar and MailMerge. For that reason, we suggest that you familiarize yourself with the reference material for WordStar in the Reference Guide.

In addition, we suggest that you learn to use the built-in HELP facilities of WordStar. Remember that **^J** is the secret command that can bring you additional information about almost every topic in WordStar, even while you're in the middle of editing a document.

On top of everything else, practice. We have attempted to make the use of a computer as easy as possible, but you'll never learn more than a few rudimentary ways of manipulating it unless you put some effort into the process. Neither author of this manual has been using word processors for long, but it did not take long to opt for their exclusive use. We know that you, too, can learn to use a word processor. We only ask that you remain patient during the process and remember the words of author Franz Kafka: "All human error is impatience, a premature renunciation of method, a delusive pinning down of a delusion."

CHAPTER 5

SuperCalc

In this chapter, you'll learn how to use SuperCalc, a program that lets you work with numbers in much the same way WordStar lets you work with words.

Introduction

The SuperCalc program supplied with your Osborne 1 is an important tool for those who want to use their Osborne computer to deal with numbers.

SuperCalc is easy to learn to use, partly because of built-in prompting messages that are concise and useful. In this chapter, you'll learn how to use SuperCalc.

What Is SuperCalc?

The SuperCalc program changes the Osborne 1's memory into a large worksheet. Since the worksheet can be as large as 63 columns by 254 rows, the screen on your computer is a "window" through which you can view a portion of this worksheet at a time. You can look at information, alter it, delete it, or replace it, all with a few easy-to-learn commands. Subject to your control, the program will recalculate any values affected by new, updated, or modified data you enter.

In a way, SuperCalc operates like a business or scientific calculator. The primary difference is that SuperCalc has an extremely large memory and can remember both equations and data you enter.

Suppose for a moment that you have entered several complex equations that predict the amount of cash your business will have if the interest rate is 20 percent. SuperCalc lets you change the data in your calculations so you can perform the same calculations with an assumed interest rate of 15 percent, 25 percent, or whatever figure you desire.

Perhaps, instead, you have a set of data for your business for the last year: monthly sales and expenses, for example. With SuperCalc you can keep the data the same while changing the calculations you perform. With a few simple changes to the equations you enter, you can perform a cash-flow analysis, a return-on-sales analysis, or even a prediction of what next year's sales may be, based on the pattern of sales already achieved.

SuperCalc is invaluable. You can use conditional expressions —for example, you might apply one formula to a set of numbers if a certain condition were met, and another if that condition were not met.

See notes on single and double density, page 758.

The Worksheet

We've already noted that SuperCalc turns your Osborne 1's memory into a large worksheet. Let's examine this worksheet more carefully.

The SuperCalc worksheet is organized as a large grid of rows and columns of information. Letters designate columns, while numbers designate rows. Since 63 columns are possible, and the alphabet has only 26 letters, double letters represent some of the columns:

A–Z → first 26 columns
AA–AZ → second 26 columns
BA–BK → final 11 columns

Numbers from 1 to the maximum, 254, show rows.

Each grid location in the worksheet is called a "cell." The cell in the upper left-hand corner of the worksheet is cell A1; the lower right-hand corner is BK254.

upper									
left	**A1**	**B1**	**C1**	**D1**	<....>	**BH1**	**BI1**	**BJ1**	**BK1**
corner	**A2**	**B2**	**C2**	**D2**	<....>	**BH2**	**BI2**	**BJ2**	**BK2**
	A3	**B3**	**C3**	**D3**	<....>	**BH3**	**BI3**	**BJ3**	**BK3**
		↑			↑		↑		
		:			:		:		
		↓			↓		↓		
	A252	**B252**	**C252**	**D252**	<....>	**BH252**	**BI252**	**BJ252**	**BK252**
	A253	**B253**	**C253**	**D253**	<....>	**BH253**	**BI253**	**BJ253**	**BK253**
	A254	**B254**	**C254**	**D254**	<....>	**BH254**	**BI254**	**BJ254**	**BK254**
									lower
									right
									corner

You enter data into "active" cells. Only one cell is active (current) at any moment. This active cell is immediately available for use. When you type in data, it goes into the active cell, and the row and column that contain the active cell are called the current row and current column. We'll use this nomenclature consistently throughout this manual.

When you first start the SuperCalc program, cells are only potential locations of data or formulas and contain no entries. Empty cells to the right and below the worksheet space you use take up no space in your computer's memory. You bring a cell into existence by "using" it in some way, by putting some information into it, by formatting it, or by using it to enter an equation. In addition, empty cells which lie between two used cells will have a bit of memory, called a "stub," assigned to them. We'll cover each of these functions in upcoming lessons.

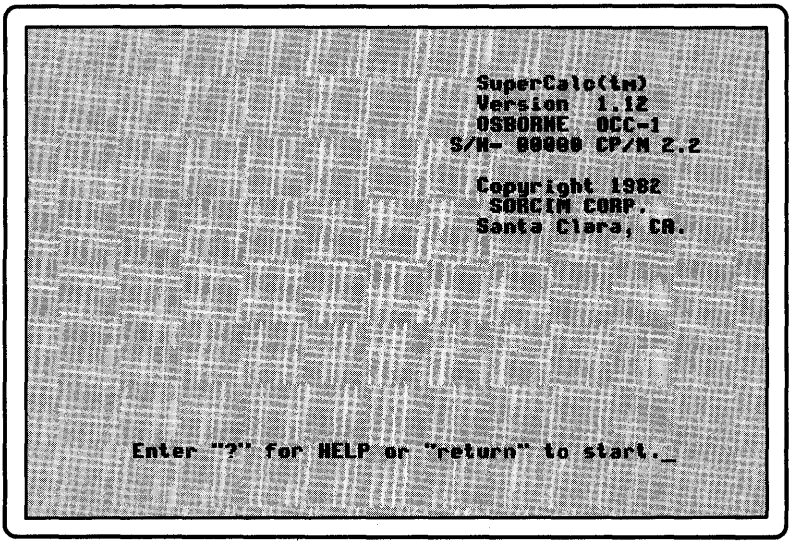
Getting Started

You're now ready to begin your exploration of SuperCalc. We assume that you've read at least the first three chapters of this manual and understand the information in them. We also assume that you followed our advice and created a copy of the

SuperCalc diskette, along with all the others you were provided. If you haven't already done so, do it now.

The normal way you'll start using SuperCalc is as follows:

1. Press the RESET button on the front of the Osborne 1.
2. Place the SuperCalc diskette in drive A and press RETURN as instructed.
3. The Osborne logo will appear and tell you that SuperCalc is loading. After a few moments, you'll see the SuperCalc sign-on message.



```
                           SuperCalc(tm)
                           Version  1.12
                           OSBORNE  OCC-1
                         S/N- 00000 CP/M 2.2

                           Copyright 1982
                            SORCIM CORP.
                           Santa Clara, CA.




     Enter "?" for HELP or "return" to start._
```

For now we'll employ only the SuperCalc diskette, but when you begin using SuperCalc for real format a blank diskette with the COPY program, as described in chapter 2, and put this diskette into drive B so you have a diskette on which you can permanently save your data and formulas.

You're now ready to work through the nine lessons that follow. As you start out, set aside some time to work your way through all the lessons before you attempt to use SuperCalc for important calculations.

Lesson 1: Moving Around the Worksheet

Imagine that you are examining a map through a magnifying glass. When you use the SuperCalc program, think of the monitor screen as your magnifying glass; through it you can view any area of your map, or SuperCalc worksheet.

The concept of the monitor as a magnifying glass looking at a larger area is similar to the "window" concept of the screen that you learned in chapter 1. Instead of using the control-arrow keys to control what area of the grid you see, SuperCalc requires that you learn a different method of specifying what section of the spreadsheet to be viewed.

In the same way that you use latitude and longitude measurements to designate a location on a map, you locate and enter data on the SuperCalc worksheet in positions specified with reference to alphabetically designated columns and numerically designated rows. Every location on your worksheet has a unique letter-and-number combination assigned to it. Let's learn how to move around the worksheet.

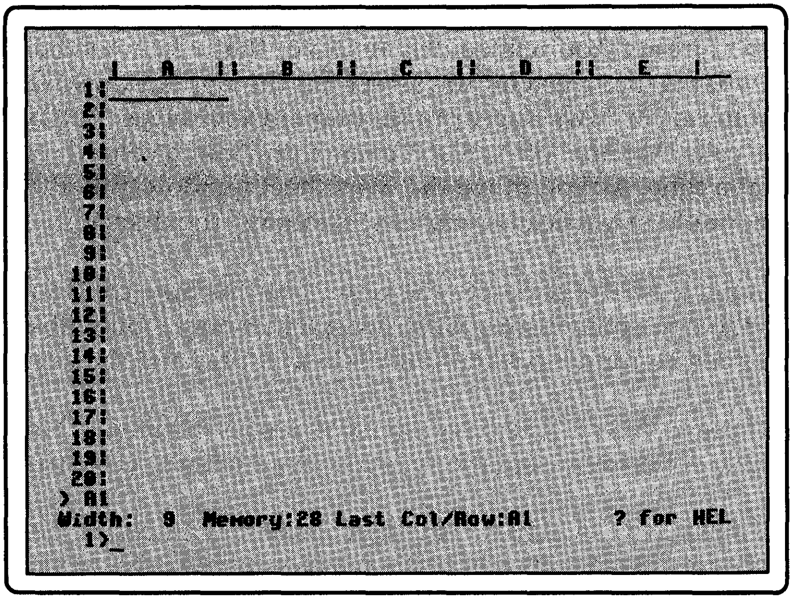
The SuperCalc sign-on message should still be on your monitor. At the bottom of the screen is a line that reads:

```
Enter "?" for HELP or "return" for start.
```

Press the RETURN key.

Examine the screen (figure next page) and you'll see columns A through E and rows 1 through 20. Notice also that one set of column and row coordinates (in this case, **A1**) is on display at the lower-left side of the screen.

A bright line is located at column A, row 1. This is the "worksheet cursor," and it designates the active cell, much like the WordStar cursor designates the active character position. The active cell is the destination of any data or information you enter.



```
   |  A  ||  B  ||  C  ||  D  ||  E  |
 1|
 2|
 3|
 4|
 5|
 6|
 7|
 8|
 9|
10|
11|
12|
13|
14|
15|
16|
17|
18|
19|
20|
> A1
Width:  9  Memory:28 Last Col/Row:A1       ? for HEL
  1>_
```

Any coordinate—for instance, A1, B3, B6, E19—is called a "cell" because it represents a unique position on your worksheet. You can position the worksheet cursor at any cell on the screen. One way to accomplish this is to press any of the arrow keys at the right side of your keyboard. Using these arrow keys, try moving the worksheet cursor.

Alternatively, you can use the cursor-control keys you learned in the WordStar chapter (**^E** for up, **^X** for down, **^S** for left, **^D** for right). When you issue one of these control characters, the worksheet cursor will move in the specified direction. We'll make all of our movement references using the arrow keys, but if you feel more comfortable with the WordStar cursor-movement controls, by all means, use them instead.

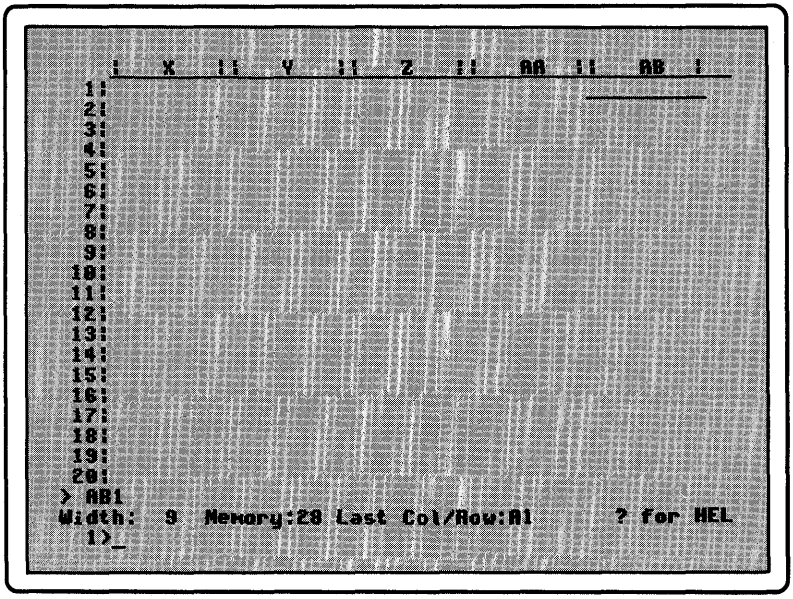
Scrolling

What happens if you try to go above row 1 or to the left of column A? Nothing. You have reached the boundary of your worksheet and cannot make the worksheet cursor go any further in those directions.

But what about moving to the right or down? Try it, if you haven't already. You will quickly discover that when you move past what is displayed on the screen (to the right or downward), the columns or rows appear to renumber themselves. Actually, those cells that are "off-screen" come into view, a column or row at a time. As you move the worksheet window either horizontally or vertically, you are "scrolling" the display.

Try moving off the screen to the right, but this time continue to hold the key down instead of striking it just once. You will see the screen continue to scroll until you stop pressing the key.

Continue to "scroll" the screen until you pass column Z. Note that two-letter representations mark the remaining columns.



```
    !    X    !!    Y    !!    Z    !!    AA   !!    AB   !
  1!
  2!
  3!
  4!
  5!
  6!
  7!
  8!
  9!
 10!
 11!
 12!
 13!
 14!
 15!
 16!
 17!
 18!
 19!
 20!
> AB1
Width:  9  Memory:28 Last Col/Row:A1        ? for HEL
  1>_
```

Illustration points out status line, prompt line, and entry line which are discussed starting on the next page.

The Status, Prompt, and Entry Lines

Now look at the three lines at the bottom of your screen. The top line is called the "status line." It tells you the current active cell and the direction the worksheet cursor will move.

The first character—a pointer (<, >, ^, v)—indicates the direction the cursor will move when you press the RETURN key. To change the direction of cursor movement, press the arrow key that points in the direction you wish it to move. Note that the arrow on the status line changes.

The next entry on the status line is the "address" of the current active cell. The status line allows you to read from your worksheet the location of the active cell more conveniently than you can by trying to estimate the column and row of its location, a problem you might encounter when you're working with narrow columns.

If the current active cell is empty, you'll see nothing else displayed on the status line. If the active cell contains text, numbers, or formulas, however, the content of the cell will appear as you originally entered it. It will be displayed like this:

```
Form = (contents of cell)
```

Now move the worksheet cursor around, this time watching the status line as the active cell and direction indicators change.

The second line at the bottom of the screen is the "prompt line" and secondary status line. This line will display the current cell width, and the amount of memory available to use, and indicate the last cell used for your current application. When you are in the command-entry mode, which you'll learn about soon, the message displayed here will change depending on what command you are currently using. The prompt message also lists your options, if any, at any given moment.

The last line at the bottom of the screen is the "entry line." It displays 1 > at the left margin. This line lets you communicate with the SuperCalc program. As you type at the keyboard, any information or command will appear on this line. The entry line is much like a scratch pad, in that it allows you to check and edit the data and text you wish to enter before you commit it to the worksheet. As you type in characters, the entry-line cursor moves to indicate where the next character will appear. At the left-hand margin, the number 1 will change to 2, 3, and so on, as the cursor moves from character 1 of your data to character 2, and so on.

GOTO: Getting Around Quickly

If you're working with a large worksheet, you'll want to be able to "jump" around from one position to another without having to scroll across many columns or rows. SuperCalc lets you make the worksheet cursor move from one active cell to another immediately by using the **GOTO** command.

The GOTO command is simply the equal sign (=). Press = and the prompt line changes to read Enter cell to jump to. Now type: M 3 1 or m 3 1. Either will work, as SuperCalc accepts lowercase or uppercase letters for any command, with two exceptions: you can't substitute a lowercase **L** for the numeral 1, or the uppercase **O** for the numeral 0, as you can on a manual typewriter.

Notice that nothing has happened, with the exception of the changes on the entry line. For any action to occur, you must press the **RETURN** key to inform SuperCalc to process your command. It's a good habit to check your work first, by reading the entry line, before pressing RETURN.

Now press RETURN, if you haven't done so already.

If you did everything right, you should see your worksheet cursor appear at cell M31, which has replaced cell A1 as the top leftmost cell of your display window. Try to use the `=` command to find out how large the worksheet is. When you are finished, **GOTO A1** again by pressing `=`, then answering `A` `1` and pressing `RETURN`.

Let's learn an additional feature of the GOTO command. Move the active cell to anywhere near the middle of the screen, say to E8. Press `=` but specify no cell; just press `RETURN`. Notice how the active cell remains **E8**, but that the window moves to make E8 the top-left corner of your display window.

You've now used the arrow keys and the GOTO command to move around the worksheet. For most situations, you process any entry you make in SuperCalc by pressing the RETURN key, which also affects the direction the cursor will move for the next entry.

Press `RETURN` a few times and notice that the position of the active cell advances to the next cell. The direction—left, right, up, or down—depends on which arrow key you last used.

Press the down-arrow key (`↓`) once, and then `RETURN` a few times . . . now the left arrow (`←`) and `RETURN` several more times. The arrow keys set the direction, and the RETURN key advances the worksheet cursor cell by cell in that direction. Remember that the status line always indicates in which direction the worksheet cursor will move if you press RETURN.

The QUIT Command

Moving around is fun, but it's time to move on.

Press `/` to initiate a command sequence. You will see that the prompt line changes. It now reads:

```
B,C,D,E,F,G,I,L,M,O,P,Q,R,S,T,U,W,X,Z,?
```

The prompt line is telling you that these letters represent the only meaningful actions you can take, now that you have typed the command prefix /.

Each letter designates a command option. Whenever you wish to examine this command-option list in its expanded form, press ? and an annotated list will display on your screen. To return to your worksheet display, press RETURN. We will explore all of the commands soon, but for now you should know about one in particular.

Make sure you are in the command mode with /, then watching the bottom lines of your screen carefully, press the Q key. What happened? First, the SuperCalc program automatically interprets the / Q so that it appears as /Quit on the entry line. Second, the prompt line has changed. It now reads:

```
EXIT SuperCalc? Y(es) or N(o).
```

If you want to stop here and continue the lesson later, press the Y key; otherwise, press N.

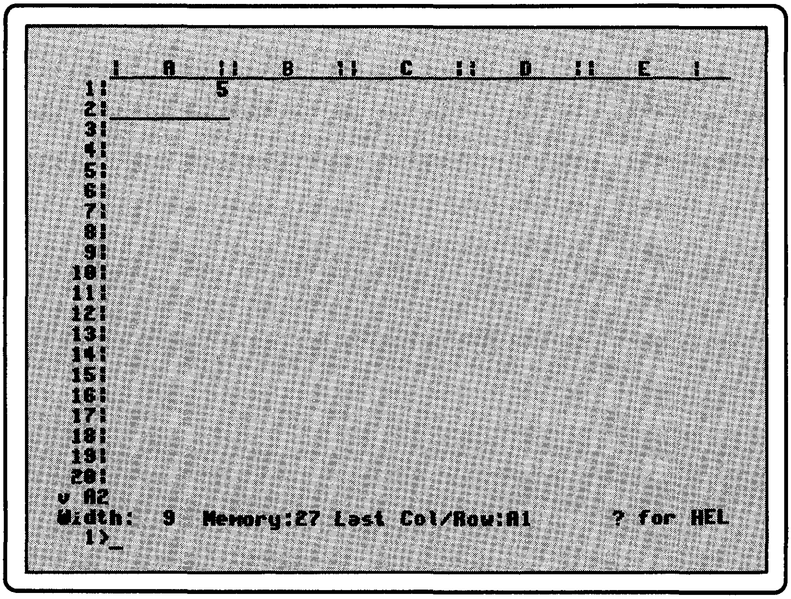
Lesson 2: Data Entry

Now make some entries on your worksheet.

In this exercise you will enter numbers down the column, so you should set the worksheet cursor to move "down" by pressing the down-arrow key. Now use the **GOTO** command to place the worksheet cursor at cell A1.

Enter the number 5 on the entry line. Do not press RETURN yet. You may cancel an entry anytime by pressing the CTRL and Z keys simultaneously. If you start to do something but then change your mind, ^Z will allow you to start over without affecting your worksheet.

You have **5** on the entry line. Now press RETURN; this action will enter whatever is currently shown on the entry line: the characters you have typed go to the active cell, and the entry line will clear. In our example, the data item 5 should now appear on the screen in cell A1.



```
     |  A  ||  B  ||  C  ||  D  ||  E  |
    1|        5
    2|
    3|
    4|
    5|
    6|
    7|
    8|
    9|
   10|
   11|
   12|
   13|
   14|
   15|
   16|
   17|
   18|
   19|
   20|
 v A2
 Width:  9  Memory:27 Last Col/Row:A1        ? for HEL
    1>_
```

Notice that the worksheet cursor moved to A2. Enter 6, but do not press RETURN yet. Before you typed 6 a 1 > was at the left edge of the entry line, now there is a 2 >. This number increases each time you type a character on the entry line. The number you see is always one more than the number of characters you have typed before pressing RETURN. For now, this information helps you fit your data into the column width you have—remember the 9 on the status line. In a later lesson, this character count will become even more helpful.

Now press RETURN, and cell A2 will contain the value of **6**. Cell A3 becomes the active cell.

Let's try another entry, this time 12. Type [1] [2] now, followed by a [RETURN].

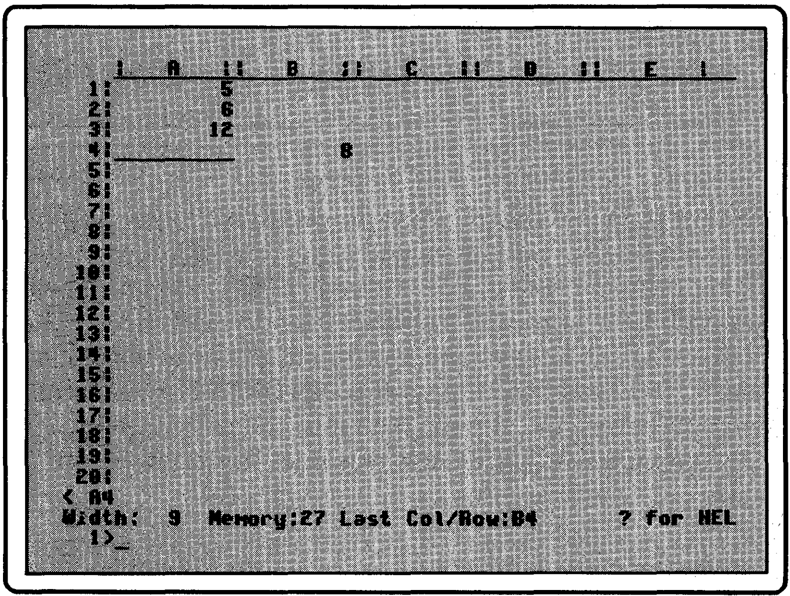
The worksheet cursor again progresses down one column, anticipating your next entry, while the data, 12, enters into cell A3.

Press the right-arrow key ([→]) once, then enter [5] [6], and press [RETURN]. What happens?

The number **56** appears in B4, and the worksheet cursor moves to cell C4. Remember that, after each entry followed by a RETURN, the worksheet cursor moves in the direction indicated on the status line depending on the arrow you last pressed.

Press the left-arrow key ([←]). Enter [8], and press [RETURN].

An **8** should replace the **56**. In addition, the worksheet cursor now moves to cell A4.



```
    |  A   ||  B  ||  C  ||  D  ||  E  |
  1|        5
  2|        6
  3|       12
  4|________         8
  5|
  6|
  7|
  8|
  9|
 10|
 11|
 12|
 13|
 14|
 15|
 16|
 17|
 18|
 19|
 20|
< A4
Width:  9  Memory:27 Last Col/Row:B4      ? for HEL
   1>_
```

Try entering different letters and numbers as data, changing direction with the arrow keys. Take a few minutes to make sure you fully understand this entry procedure.

Depending on how adventurous you were, you may have made some discoveries. Generally, you can make two kinds of data entries: text and numbers. SuperCalc considers your entry a number unless you press the double quotation mark (") as the first character, in which case subsequent entry is treated as text. A single quote preceding an entry causes it to be repeated. Later in this chapter we'll show you how to enter formulas.

Headings, labels, and explanatory notes are all examples of text entries. In a mathematical sense, they simply have a value of zero. If you forget to lead these entries with double quotation marks, the computer will respond with an error message. The quotation marks do not appear on your screen; they simply signal the computer that you are making a text entry, just like / signals a command. You do not have to close the quotation marks.

You can obtain repeated text by starting your item with a single quotation mark. For instance, typing ' - will result in a series of hyphens appearing across the remaining positions in that row. We'll come back to this concept in a bit.

The ZAP Command

Let's try some more examples, but first let's start with a fresh screen.

Remember that we used the **QUIT** command to exit from SuperCalc in lesson 1. Now we'll use another command, **ZAP**.

Press / and note that the prompt line again displays a listing of all possible commands.

Press Z. The prompt line now reads:

```
Y(es) to clear everything, else N(o)
```

The entry line shows:

```
/Zap-ENTIRE-Worksheet?
```

The effect of the ZAP command is to clear the entire worksheet and return everything to its original state, just as it was when you first loaded the SuperCalc program. Because the command's effect is so drastic, the program uses the prompt line to remind you that the entire worksheet will empty if you follow through and actually execute the command.

Since you do indeed want to clear everything, press the Y key. SuperCalc will clear the worksheet and replace the worksheet cursor at cell A1. Whatever you entered on the worksheet is now gone, permanently.

Characteristics of Text and Numeric Entries

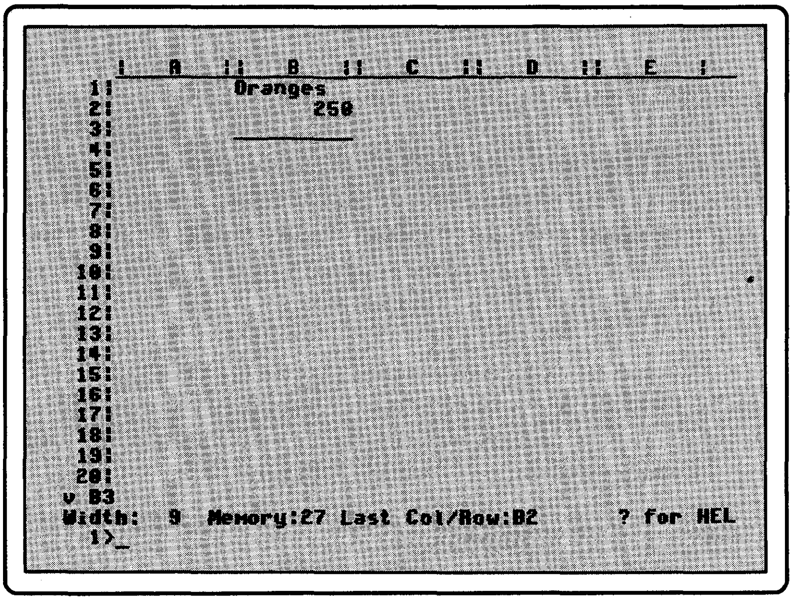
Now enter:

```
"Oranges
```

in cell B1 and

```
250
```

in cell B2. Note that you must lead off **Oranges** with a quotation mark to inform SuperCalc that you're entering text. When you've entered these two values, you should notice that the text is left-justified while the numbers are right-justified within column B.



```
    !    A    !!    B    !!    C    !!    D    !!    E    !
  1!           Oranges
  2!                  250
  3!
  4!
  5!
  6!
  7!
  8!
  9!
 10!
 11!
 12!
 13!
 14!
 15!
 16!
 17!
 18!
 19!
 20!
v B3
Width:   9  Memory:27 Last Col/Row:B2        ? for HEL
  1>_
```

Now move the active cell back to B2 and watch the rightmost display of the status line. FORM = 250 is how it should read. Move the active cell to B1; the same display now reads:

TEXT = "Oranges.

How wide are the columns; how large a number can we enter? How much text?

Remember we mentioned display width earlier when looking at the status line. Note again the 9 on the status line under the width heading.

The **9** tells us that the column currently accessed (the column with the active cell) is set to display nine characters. Nine is the standard, or default, value that SuperCalc uses for the display width of all columns unless you specify otherwise. You will soon learn how to specify display widths. Text may contain 47 characters visibly, or up to 115 that can be reaccessed with the EDIT command. You can also enter as many as 16 numeric characters per cell. The maximum column width is 127.

Move the cursor to cell B3 and enter:

```
"Alberta peaches
```

This piece of text is longer than nine characters, but SuperCalc allows display of your text to extend over neighboring cells, but only if those other cells are unused. Now move the worksheet cursor to cell A1, press ["], and type [A] [L] [B] [E] [R] [T] [A] [SPACE BAR] [P] [E] [A] [C] [H] [E] [S] again.

Your entry does not display in full because B1 is occupied. But the entire entry is accepted in cell A1 even if only a portion of it —the first nine characters—is displayed. The status line with the worksheet cursor at A1 should indicate:

```
Text =  "Alberta peaches
```

Move to cell B4. Enter, without commas,

```
2500000000
```

(that's two-and-a-half billion). The number is too large to display. SuperCalc converts it to scientific notation, a more compact form of representing a number, and displays it as 2.5e9—which means that the number in B4 is 2.5 times ten to the ninth power.

SuperCalc provides many different display and format options. You've learned just a few; we'll introduce more later. If exponential numbers (scientific notation) are new to you, here is a quick look at what they are and how SuperCalc displays them. Exponential numbers are displayed as "powers of 10." You will soon see what this means.

GOTO cell C1 and set column C for exponential display. You use the **FORMAT** command to do this. Type / F, for the command, and C, for "column formatting." When the prompt asks you what column to format, you can just press the , key, because you are at column C, and pressing the comma key indicates SuperCalc is to get the information it needs from the current location of the worksheet cursor. Next, press E, for exponential, and press RETURN to complete the entry.

Press the down-arrow key (↓) to set the current direction as down. Now enter 1776 and press RETURN. Cell C2 shows 1.776e3. What does this mean? **e3** means exponential 3, or "10 to the power of 3," or 1000; 1.776 times 1000 is 1776.

Try entering 1000. Is **1e3** what you expected? What will represent 100? Try it. Now enter 2000, and then enter .002. Notice that 2000 is **2e3** and that .002 is **2e-3**. If **e3** is thousands, **e-3** is thousandths. What is **-2000**? Try it and see.

What happens if you enter a number in exponential notation? Try it. Enter:

```
567e13
```

Are you surprised to see it display as 5.67e15? SuperCalc prefers to put the decimal point just after the first digit and will adjust the exponential value to do so.

Explore on your own—use numbers in both decimal notation and in exponential form. Try to guess beforehand what the display will be.

When you feel comfortable with exponential notation, give SuperCalc a little job to do.

Press the down-arrow key to reset the cursor direction, then GOTO (=) cell D1.

In cell D1, enter:

```
93000000
```

and press RETURN. That is 93 million, the number of miles between the earth and the sun. Now in cell D2, enter:

```
5280*D1
```

and press RETURN. The value displayed—4.910e11—is the number of feet in 93 million miles. What about inches? Enter:

```
12*D2
```

in cell D3. The result—5.892e12—is the number of inches in 93 million miles.

What 5.892e12 tells us is that there are about 5.9 times 1,000,000,000,000 inches between here and the sun. Only the first two digits of 5.892 are significant, because only the 93 was significant in 93 million miles.

We bring this up because the reason for use of scientific notation is to let you quickly grasp the essential points of a number and discard the unessential. Scientific notation splits very large and very small numbers into two distinct parts—both of which are easily scanned—the significant digits and the general magnitude of the value.

You used SuperCalc to perform calculations in the above example, but that's getting ahead of the game, so let's return to more formatting information.

In-Line Editing

Right now let's investigate SuperCalc's in-line editing feature. If you have used the exponential notation section of the lesson, **ZAP** your worksheet and reenter:

```
"Oranges

250
```

and

```
"Alberta peaches
```

in their original positions.

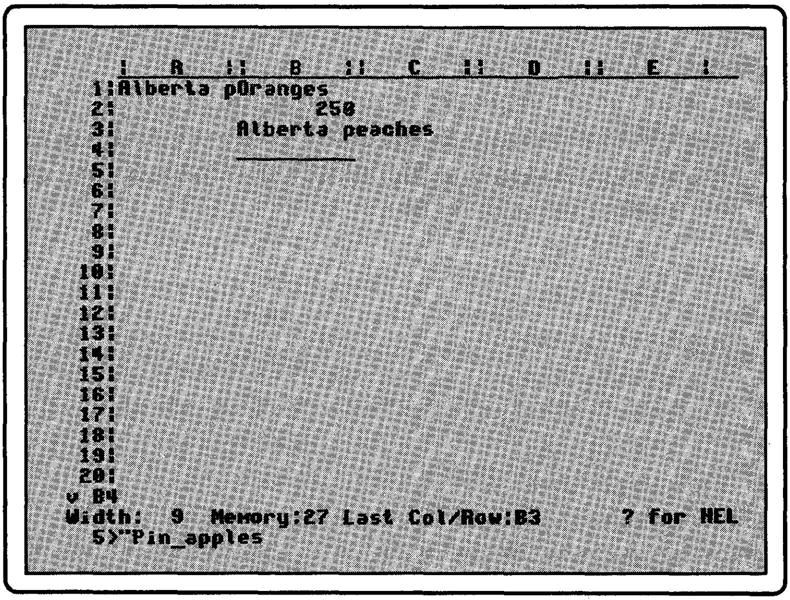
Move the active cell to B4. Type in the text entry, using the quotation mark to tell SuperCalc that text is coming next, with the following incorrect spelling but do not press RETURN:

```
"Pinapples
```

As you know, you could use a left-arrow key to backspace and retype from the point of the error. The right-arrow key moves your cursor one space to the right.

Using the left and right arrows, move back and forth across your text, but take care not to backspace beyond the leftmost character. Notice that nothing changes except for the position of the cursor on the entry line. Now locate the cursor on the **a** in **Pinapples**. Notice also that the number 5 appears at the left of your entry line to tell you that you're on the fourth character in the entry.

Press the up-arrow key ([↑]) and see what happens.



```
     !    A    !!    B    !!    C    !!    D    !!    E    !
    1!Alberta pOranges
    2!              250
    3!         Alberta peaches
    4!
    5!         ___________
    6!
    7!
    8!
    9!
   10!
   11!
   12!
   13!
   14!
   15!
   16!
   17!
   18!
   19!
   20!
 v B4
 Width:  9  Memory:27 Last Col/Row:B3        ? for HEL
    5>"Pin_apples
```

SuperCalc has created a space for you just ahead of the **a** so that you can insert a one-character correction without having to retype any text. Enter [e], and your entry line now reads "Pineapples. What if you had needed to insert several characters, or to delete some?

Press the up-arrow key continuously and create a large gap in the text. Press the down-arrow key once and notice that the gap reduces by one character. Hold the key down and watch the blank spaces delete. Go ahead and enter **"Pineapples** and then make up other examples. Practice with the editing keys until you are confident with this in-line editing feature. Try it with numeric entries, too.

Regardless of where the cursor is on the entry line, all the visible text or numeric values go into the active cell when you press RETURN.

The arrow keys have more than one use. They move the active cell around the worksheet until you type a character on the entry line. Then SuperCalc recognizes that you have begun to enter data. The function of the arrow keys changes in this data-entry mode; they are now for editing.

The EDIT Command

You've seen how you can edit data before it is entered into a cell. You might now want to know if there's a way to edit already processed data. You could enter the data again in its entirety—a new entry replaces an old one in SuperCalc. There is a better way, though. You use a new command, the **EDIT** command (**/ E**).

Make B4 the active cell—use GOTO or move the worksheet cursor. Enter / E to inform SuperCalc you wish to edit. The prompt line now reads:

From? Enter cell.

SuperCalc is asking where to find the material you want to edit. Because you want to edit the contents of the active cell, you need only press RETURN and SuperCalc will bring the cell's contents to the entry line.

Make some changes using the arrow keys, as we previously discussed. For instance, delete a few characters from the entry. When your change is complete, press RETURN, and your modified entry replaces the old one in B4. If you haven't done this, try it now.

Sometimes you may wish to edit the contents of a cell and put them into another cell. For example, position the active cell at B5, your destination cell. Issue / E to enter the edit mode. In response to the prompt, answer B 4, your source cell, and press RETURN. The contents of cell B4 will be shown at the entry line. After you make your change, press RETURN to send the edited version of your entry to the current active cell, B5.

Note that no matter where it comes from, the "new" or "edited" data on the entry line always goes into the active cell. In our first example, the original contents of B4, the active cell, were modified and replaced by our edited version because B4 was the active cell. In the second example, the contents of B4 didn't change. The edited material went into cell B5, the active cell, and the source material remained unchanged in cell B4.

If you want to stop here, use the QUIT command as you learned previously. Otherwise, proceed on to the next lesson.

Lesson 3: Blanking, Protecting, and Saving Your Work

In lesson 2, you expanded your knowledge to include the fundamentals of data entry for the purposes of creating text or for entering numeric data to use in actual calculations. In this lesson, you'll gain more experience entering data. You'll learn to blank, protect, unprotect, and save your data. You will also learn to use the **/ G** command to make some general or "global" changes in your worksheet display and to use the **/ F** command to make certain formatting changes.

If you are continuing directly from lesson 2, issue a **ZAP** command (/ Z) so that you start with an empty screen. Otherwise, load the SuperCalc program in accordance with what you learned earlier.

Use the down arrow to set the current direction. Use the **GOTO** command to go to cell A1. Enter the following following data in column A:

```
"Apples
5
8
3
11
```

4
9
6
12

Note: You should end up with the active cell as A10, with data in cells A1 through A9.

In lesson 2, you learned how to modify a cell's contents, to edit. What if you just want to completely erase, or blank, the contents of a cell?

You do that with a new command, the **BLANK** command. The **BLANK** command blanks out or erases data you have already entered on your worksheet. You can blank an individual entry or cell, partial or complete rows or columns, or entire blocks (rows and columns) of cells. We'll try an example of each in this lesson.

Press / and note the prompt line. Now press B. SuperCalc fills in the rest of the command, so you should now see /BLANK on the the entry line. The prompt line should read:

Enter range

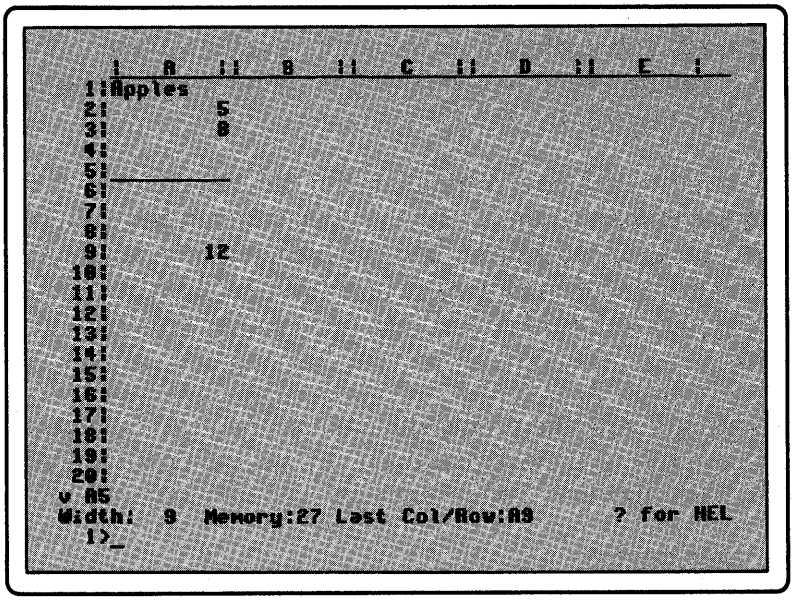
You must now specify the portion—or range, if you will—of the worksheet you want to blank.

Enter **A4:B4** and press RETURN. The contents of A4 through B4 will disappear. You can blank a single cell by placing the worksheet cursor on the cell you want to blank, typing / B and, with no cell reference, press RETURN. Try doing this with cell A5. When working regularly with SuperCalc, use whichever method is more convenient for you. Remember that since the cursor can only point to an individual cell, you affect only an individual entry when you press RETURN with no cell coordinates.

Type / B again. Now, in response to the Enter range prompt, specify A6 through A8 by typing:

A 6 : A 8

Press [RETURN]. The colon in the above example is how you tell SuperCalc that you are specifying a range of cells. The two cells you identify are the first and last cells of the range.



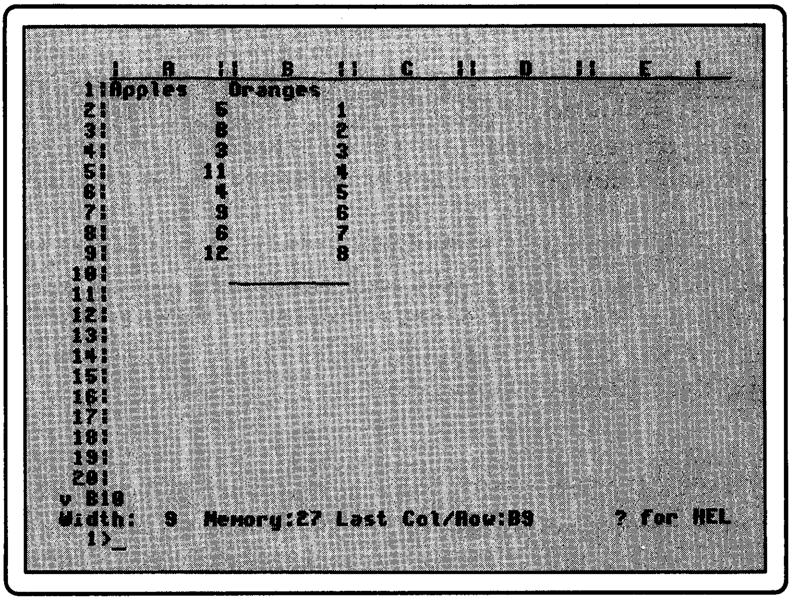
```
   |    A    ||    B    ||    C    ||    D    ||    E    |
  1|Apples
  2|        5
  3|        8
  4|
  5|_________
  6|
  7|
  8|
  9|       12
 10|
 11|
 12|
 13|
 14|
 15|
 16|
 17|
 18|
 19|
 20|
v A5
Width:  9  Memory:27 Last Col/Row:A9        ? for HEL
  1>_
```

The PROTECT Command

Reenter the numbers in column A that you just blanked out. Create a new column of numbers in column B. Label it Oranges:

```
"Oranges
1
2
3
4
5
6
7
8
```

Your display should now look like this:



```
     |    A    ||    B    ||    C    ||    D    ||    E    |
   1|Apples   Oranges
   2|        5         1
   3|        8         2
   4|        3         3
   5|       11         4
   6|        4         5
   7|        9         6
   8|        6         7
   9|       12         8
  10|         ________
  11|
  12|
  13|
  14|
  15|
  16|
  17|
  18|
  19|
  20|
v B10
Width:  9  Memory:27 Last Col/Row:B9        ? for HEL
   1>_
```

Now let's use the **/ P** command to protect a cell. Issue / P. Use the **PROTECT** command exactly the same way you just learned for the **BLANK** command: enter a cell number or a range using the colon as the range indicator. For example, answer A 5 and press RETURN. Move the cursor to cell A5 and notice that a P now appears next to the Form display on the status line. This indicator tells you that the active cell is "protected." You'll also note that the information in cell A5 has "dimmed"—it is not displayed at the same intensity as the rest of the data you entered.

Continue by protecting a range of cells. Type / P, and answer A 8 : B 8, then press RETURN. This will protect entries in columns A and B of row 8.

What is the significance of what you've just done?

Remember we said that **/ B** could blank out an entire block of cells. Let's attempt to blank out a block of cells from row 2 through row 8 for both columns A and B. Guess how we specify this.

Type / B, followed by A 2 : B 8 and a RETURN. You define the range for a block of cells as a diagonal with the top leftmost cell followed by the lower rightmost cell in the block.

Row 1, with titles, remains because it lies outside the range of the block definition we used with the **BLANK** command. Cell A5 and row 8 remain because they are protected. Row 9 remains, not because it is protected, but because it, too, is beyond the range we blanked out.

Try to change the contents of cell A9 to 17. Now try the same thing with cell A5 or cell B8. Because the latter two cells have been protected, you can't change or blank them. This feature can provide you with a large measure of safety when you work around information you've taken a long time to develop and that you cannot afford to lose accidentally.

The **UNPROTECT** command, **/ U**, can unprotect the information in a cell, a partial row or column, or a block of cells. You can use the command twice to unprotect cell A5 and row 8, but can you do it with just one **/ U** command?

Yes. How would you unprotect a block, rows 5 through 8 of columns A and B? What is the proper range specification? Did you say **A5:B8**? That's correct. Now go ahead and do it.

Formula and Numeric Display Options

Move the active cell to A2. Enter 3 + 5 RETURN. What happened? The value of the expression, 8, was placed in cell A2. If the worksheet cursor is not at A2, move it there and examine the status line. The rightmost display will read:

```
Form = 3 + 5
```

Has SuperCalc actually stored the **3+5** or the **8**?

However complicated the expression, SuperCalc calculates the result and displays it. This lets you use the entry line as a scratch pad. For instance, you may be adding two columns of numbers, but you're only interested in their total value.

Again at A3, enter 1 + A 2. SuperCalc recognizes this as a formula referring to cell A2 and quickly calculates and displays the value based on the value in A2. Further, if you change the contents of cell A2—to 5, for instance—you should observe that the new value of A3 will be recalculated as well. Try it!

The active cell should be A3. The screen displays 6, the current value, there, while the status line displays Form = 1 + A2. Apparently, SuperCalc is keeping track of both expressions. In cell A4, enter A 3 * . 6 5. The * means multiply and is the equivalent to the **x** sign in conventional notation. Division is represented by / .

Locate the active cell at A10. Enter:

```
SUM(A2:A9)
```

followed by a RETURN.

SUM is a built-in function of SuperCalc. Many such functions are built into SuperCalc, including:

SQRT	square root
AVERAGE	mathematical mean
NPV	let present value
SIN	trigonometric function (sine)
COS	trigonometric function (cosine)
TAN	trigonometric function (tangent)

For more detail on these functions, see the Reference Guide.

For SUM, you specify a list of ranges, as you've just done, or cells, as in **SUM (A8,B9,A2)**.

Now change the value of cell A2 from 5 to 3 and watch SuperCalc recalculate the sum.

The GLOBAL Command—Formatting Options

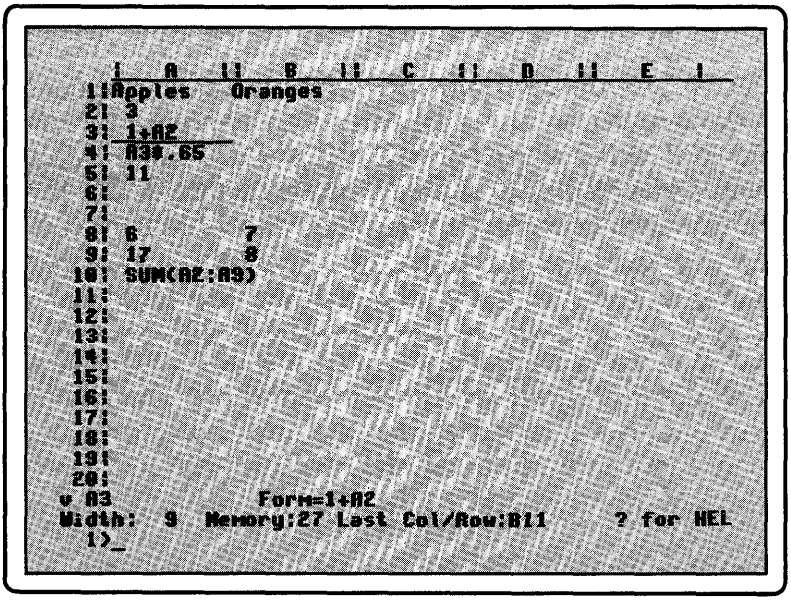
Earlier we determined that SuperCalc kept track of formulas, although it only displayed their current values on the worksheet. To review all the original formulas, issue / G.

Notice that SuperCalc's prompting fills the **/ G** to read /Global. What does this mean? The prompt line now reads:

```
F(orm.),N(ext),B(order),T(ab),R(ow),C(ol),M(an.),A(uto)?
```

You can think of the **/ G** command as a way to make overall, or "global," changes to the worksheet, rather than specific, or "local" changes. It is as if you had a map of California and could, at will, transform it into a topographical map, a population-density map, or a tourist-attraction map.

Our concern here is with formulas, so press F and see a display of your formulas as well as numerical data.



```
   :    A    ::    B    ::    C    ::    D    ::    E    :
  1:Apples    Oranges
  2: 3
  3: 1+A2
  4: A3*.65
  5: 11
  6:
  7:
  8: 6         7
  9: 17        8
 10: SUM(A2:A9)
 11:
 12:
 13:
 14:
 15:
 16:
 17:
 18:
 19:
 20:
v A3            Form=1+A2
Width:  9  Memory:27 Last Col/Row:B11     ? for HEL
  1>_
```

To return to the other style of display (cell values), simply repeat the sequence [/] [G] [F]. SuperCalc will alternate, or switch between the two display modes.

Determining Column Width

Enter [9] in cell B10.

Make sure the formulas are displayed. When SuperCalc displays formulas, you will notice one problem. The SUM formula in A10 has two characters more than the column width, which is only 9. Let's widen the column to accommodate longer entries.

Type [/] [F], for FORMAT. The prompt line will respond with:

```
Enter Level: G(lobal), C(olumn), R(ow), or E(ntry).
```

The G in this case is not the same as the **/G** command. Here it simply qualifies the **/F** command; but its meaning is similar: "for all" or "every." Now press [G]. The prompt line displays:

```
Define Formats: (I,G,E $ ,R,L,TR,TL * ,D,column width)
```

As you can see, the **/ F** command has many possible options; for now, though, define a new column width by typing **12**, followed by RETURN. If necessary, move the cursor to column B and note the status-line display. It should indicate that the width is now 12. Notice that all columns have changed to a 12-character width. You could have specified the new width for just a single column by typing a **C**, for column, instead of **G**, for global.

You are now using commands with several levels of prompts. Another use for the left-arrow key—one that you may have discovered already—is that backspacing with the left arrow always takes you back to the prior "step" in a command, to a less specific level of the command.

For instance, type / F G 1 2 again. Now backspace once with the left arrow. Backspace again, and see that the prompt changes to its earlier message, "Enter Level...."

You can, if you want, specify a level other than global and continue with the command sequence. Instead, backspace again.

You will see the list of / commands on the prompt line. Backspace again. Now you should finally have backed all the way to the original prompt.

Of course, no matter how far you have gone in specifying some command, range, or option, you can always use **^Z** to abort —cancel—the operation. Typing ^Z always returns you to the original prompt. You can use this technique, for example, if you start to enter data on the entry line and then notice that the active cell is not positioned where you want it.

If you haven't done so already, return to the display mode that displays cell values rather than formulas by typing / G F.

The SAVE Command

You will want to save the work you've done in this lesson so you can use it for later lessons. Save your worksheet by using the **SAVE** command, **/ S**. This command makes a copy of the entire worksheet and stores it on a diskette located on either drive A or drive B, depending on which you specify.

Issue: / S and a prompt requests:

```
Enter File Name (or < RETURN >  for directory)
```

You can respond to this in one of several ways, depending on where you want to store your file. If you wish to save it on a formatted diskette in drive B, type **B:WORK1** and RETURN. Or omit the B: if you want the file stored on the SuperCalc diskette in drive A. Do not leave any blank spaces in your file name. The computer will not accept **TIME 1**, but **TIME1** is acceptable. If you are unsure about file names in CP/M, review Chapter 3 before proceeding.

After you have entered the file name, the prompt line inquires further:

```
A(ll), V(alue) or P(art)?
```

Since you wish to save both your formulas and your values, press A for all. Your disk drive will whirr and click contentedly for a few moments.

We will use this newly created file to "load" your work back into the system when we resume with lesson 4, so keep the diskette handy.

Now issue / Q and exit from the SuperCalc program; all your work disappears. It is gone irretrievably, unless you specifically save it with the SAVE command before exiting as we have just showed you.

Lesson 4: COPY and REPLICATE

In lesson 3 you began to learn about the "power" of the SuperCalc program—in particular, its ability to recalculate automatically all values that depended on the values in other cells. In this lesson you'll gain more insight into SuperCalc's versatility. You'll learn to use the **LOAD** (**/ L**), **COPY** (**/ C**), and **REPLICATE** (**/ R**) commands, and the current-cell key (**ESC**). **/ L**, **/ C**, **/ R**, and **ESC** are basically time-saving commands.

The LOAD Command

Continue using the worksheet you began in the last lesson. If you used the **/ QUIT** command in the last lesson, then you will have to start SuperCalc over again. To start SuperCalc from the CP/M A > prompt, type S C and press RETURN. Use the **LOAD** command, / L, to retrieve the file you created at the end of lesson 3. If the file is not on the diskette that contains the SuperCalc program, be sure to insert the diskette with the file into the B disk drive.

Issue / L. How you respond to the prompt message Enter File name depends on where you stored the file. Within the context of these lessons, enter:

```
B:WORK1
```

then press RETURN.

If the file is on the system drive, you do not need to include the drive identifier B:. You can examine the file directory of either drive by pressing **RETURN** before supplying a file name; this is so indicated on the screen.

The disk drive will respond with some clicking, and the prompt line will change to read A(ll) or P(art)?. Press A, for All, and the material you saved from your last effort will be extracted from the diskette and appear on the screen.

The COPY Command

Now that you've restored your work from the previous lesson, it's time to investigate another command, **COPY** (**/ C**). The COPY command is easy to use. You can copy a single cell, a partial row or partial column, or a block of cells.

In this first example, copy the data in column A into column C. Issue / C. The prompt line responds with:

```
From? (Enter Range)
```

In response, answer A 1 : A 1 0, followed by a RETURN. This time the prompt asks:

```
To? (Enter Cell), then Return; or "," for Options
```

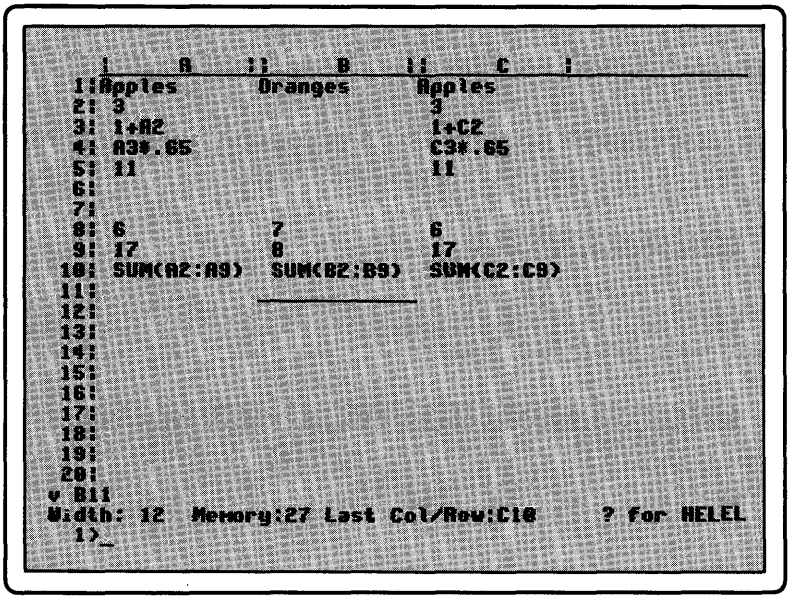
We just want a "standard" (exact) copy this time—we'll look at the options later. So specify C 1 and press RETURN.

Now use the COPY command to copy the contents of cell A10 to cell B10.

Next, change the display to show formulas (/ G F), and look at the contents of cell B10 and column C. The formulas you entered into column A have all been adjusted relative to the column in which they appeared. All cell references have changed to reflect the new location of the formulas. If you had moved to a new row, as well as a new column, the relative row designation would also have been adjusted. (Figure next page.)

Generally, this automatic adjustment is exactly what you want. But other options are open to you. For instance, you can specify that there be no adjustment, or you can tell the SuperCalc program to ask whether each occurrence of a cell reference should be adjusted or left alone. We'll try this soon.

The COPY command makes a one-to-one copy of its source material into a destination of the same type and size: cell to cell, row to row, column to column. But suppose you want to repeat a series of values and formulas many times, perhaps to compare alternative cases.



```
    !     A     !!     B     !!     C     !
  1!Apples       Oranges      Apples
  2! 3                          3
  3! 1+A2                       1+C2
  4! A3*.65                     C3*.65
  5! 11                         11
  6!
  7!
  8! 6           7              6
  9! 17          8              17
 10! SUM(A2:A9)  SUM(B2:B9)   SUM(C2:C9)
 11!
 12!
 13!
 14!
 15!
 16!
 17!
 18!
 19!
 20!
v B11
Width: 12  Memory:27 Last Col/Row:C10      ? for HELEL
  1>_
```

The REPLICATE Command

You can use another powerful command, **REPLICATE** (**/R**), for repetitions. It makes a "one-to-many" copy of a cell, a partial row, or a partial column and distributes these copies over a destination range that is larger than the source range. If you haven't already done so, change the display to show formulas (**/GF**).

Try replicating a single cell, A10.

Type / R. For From?, answer A 1 0 followed by a RETURN. For To?, specify the range D10 through F10 by typing D 1 0 : F 1 0, and follow that with another RETURN. Note how the command performs.

Replicate the partial column A3 through A4 into D3 through E3. These columns, D through E, now have data in row 10.

The REPLICATE command has the same formula-adjustment options as the COPY command, so try one of them.

Enter into cell A12 the formula A2+A2.

Now type the sequence / R A 1 2 , B 1 2 : E 1 2. After you specify **E12**, enter an additional comma (,) to receive a list of options. They will display on the prompt line:

```
N(o Adjust), A(sk for Adjust), V(alue)
```

Press A, for ask for adjustment.

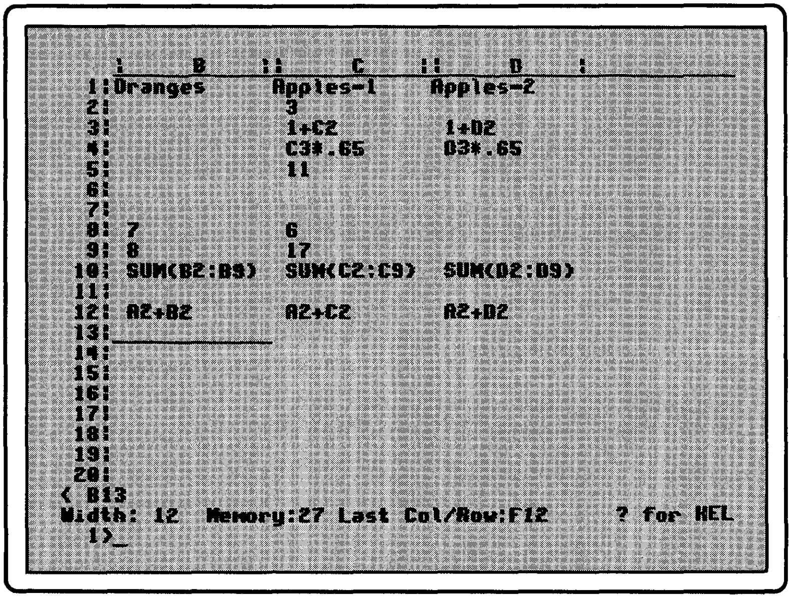
The prompt line changes to read:

```
Source location A12. Adjust A2?
```

and the first A2 is highlighted on the entry line. Respond with N for no adjustment. Now the second reference to A2 is highlighted on the entry line. Respond with a Y. You see that the first part of your formula remains unchanged, while the second adjusts, according to your responses. In this way, you can specify that one component of a cell holds constant, while other components adjust relative to their new location.

Replicate cell A1 to cells D1 through E1. Then use / E to edit the contents of C1 through E1 so they will be **Apples-1, Apples-2**, and so forth. (Figure next page.)

It's important to save the work you have completed up to this point. We'll use it again in lesson 5. Save it on the storage diskette in drive B, type / S, then B:WORK1 followed by a RETURN. Otherwise, specify the drive that has your destination diskette in it. If you need to jog your memory about how SAVE works, press the ? key to receive HELP or go back to lesson 3 for a detailed reference.



```
    :     B      ::     C      ::     D      :
  1:Oranges      Apples-1     Apples-2
  2:               3
  3:               1+C2         1+D2
  4:               C3*.65       D3*.65
  5:               11
  6:
  7:
  8: 7             6
  9: 8             17
 10: SUM(B2:B9)    SUM(C2:C9)   SUM(D2:D9)
 11:
 12: A2+B2         A2+C2        A2+D2
 13:____________
 14:
 15:
 16:
 17:
 18:
 19:
 20:
< B13
Width: 12  Memory:27 Last Col/Row:F12     ? for HEL
  1>_
```

To help protect your work, SuperCalc checks to see if you already have a file with the same name on your destination diskette. If you do, SuperCalc then asks you:

```
File already exists: C(hange name), B(ackup), O(verwrite)?
```

You now have the option of retaining the existing file either by renaming the file **C**, or by backing up the existing file with **B**. **O** may be used to specify that the existing file be deleted and replaced with a copy of the current worksheet.

When the **B**ackup option is selected, SuperCalc will rename the existing file by changing its extension to .BAK. The current worksheet is then saved with a .CAL-type extension which allows you to always have the two latest versions on disk.

Using the overwrite option causes the original file to be deleted before the new one is saved. This can be a potentially dangerous option, especially if your disk is almost full. If the current worksheet is larger than the one before it and storage on the disk is limited, the old file is deleted but the new file is not saved.

However, the SuperCalc program will spot this potentially dangerous situation and point it out beforehand.

If you no longer need the original version, reply with an **O**; otherwise, specify **B** to rename the original version with a **.BAK** file type, or **C** to specify a different name to save the file under. Since we no longer need the old WORK1 file, you may overwrite it by pressing O, then press A for A(ll). Normally, you would use the backup option.

Try replicating a row, or rows, or a block. If a practical application of your own comes to mind, begin a simple example on the screen. If you want to save this first effort of your own, be sure to use a different name—for example, TRIAL or MYTRY.

SuperCalc offers many command options, which makes it powerful and versatile. We can't introduce you to all the options in this tutorial, so we encourage you to investigate them on your own. You should find it easy to make the best possible use of the SuperCalc program by combining what you learn here with information available in the Reference Guide and through the HELP function (**?**) built into SuperCalc.

The Current-Cell Key: ESCAPE Key

This is a good place to become acquainted with the "current-cell" key. You can use it to boost the efficiency of certain kinds of data manipulations that use the COPY and REPLICATE commands. The ESC key serves as the current-cell key.

Whenever SuperCalc requires a cell or range, you can place the active-cell coordinate on the entry line by simply pressing the ESC key.

Set up an example to learn how to use this feature. Start with a fresh screen by ZAPping the worksheet.

Enter 123 into cell A1. Use the **REPLICATE** command to fill every cell on the visible screen with **123**. Try to do this before you look ahead.

Here are two ways you could have done it:

1) **/ RA1,B1:E1 RETURN**
2) **/ RA1:E1,A2:A20 RETURN**
or
1) **/ RA1,A2:A20 RETURN**
2) **/ RA1:A20,B1:E1 RETURN**

You should see **123** everywhere on your screen.

Type / B for BLANK. SuperCalc now wants you to specify a cell or a range to blank. Let's start with a single cell.

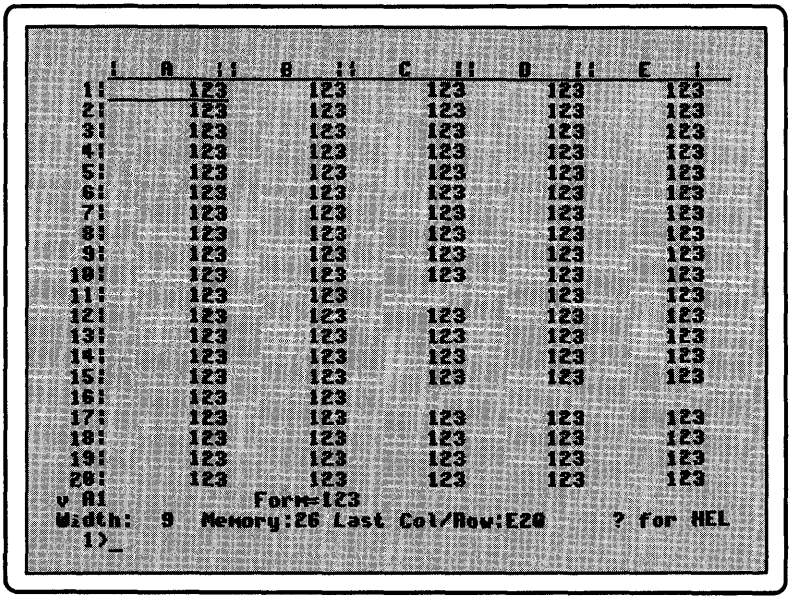
Press the ESC key. The address of the active cell will appear on the entry line. Use the arrow keys to move the worksheet cursor to another location—for example, C11. Notice the active-cell address on the entry line change as you move.

Now press RETURN. Observe that the latest active cell was blanked and that the active cell location has returned to its original place. Again, type / B for BLANK and press ESC.

Use the arrow keys to make cell C16 the active cell. The entry line now reads /Blank,C16. You can use this line to begin a range specification. Just press :. The line now reads /Blank,C16:C16. Move the worksheet cursor to cell E16.

Notice that the second address of the range increments as you go. Press RETURN. The cells in the range C16 through E16 blank.

In brief, this is what happens. Once you have set the ESC function, the arrow keys move the worksheet cursor and set the cell location on the entry line. A colon (:) generates a limiting location (the starting point) for a range specification. Pressing the ESC or RETURN key terminates the ESC function and allows you to use the arrow keys again for editing. The ESC movement of the active cell is only temporary; when you terminate the ESC function, the active cell returns to its starting place.



```
    |    A    ||    B    ||    C    ||    D    ||    E    |
  1|      123        123        123        123        123
  2|      123        123        123        123        123
  3|      123        123        123        123        123
  4|      123        123        123        123        123
  5|      123        123        123        123        123
  6|      123        123        123        123        123
  7|      123        123        123        123        123
  8|      123        123        123        123        123
  9|      123        123        123        123        123
 10|      123        123        123        123        123
 11|      123        123                   123        123
 12|      123        123        123        123        123
 13|      123        123        123        123        123
 14|      123        123        123        123        123
 15|      123        123        123        123        123
 16|      123        123
 17|      123        123        123        123        123
 18|      123        123        123        123        123
 19|      123        123        123        123        123
 20|      123        123        123        123        123
v A1              Form=123
Width:  9  Memory:26 Last Col/Row:E20      ? for HEL
  1>_
```

Here is another sample. Type [/] [B]. Press [ESC]. Move the worksheet cursor to cell D4 and press [:]; then move again to cell E14 and press [RETURN]. You have blanked cells in the block from D4 to E14.

By using the ESC key and placing the active cell at the appropriate points, you can let SuperCalc define your coordinates. At first, this may seem difficult, but with some practice you will begin to find it increasingly useful. This feature allows you to modify your screen by simply pointing with the active cell to the boundary of the range of cells you wish to modify.

Don't bother to save any of this work. At this point you may quit or continue with lesson 5.

Lesson 5: Move, Insert, and Delete

You've learned to use the GOTO command, the ESC key, and many important / commands. You can save and load your worksheet. Next we'll introduce some new commands and techniques that can greatly simplify the development of a complex worksheet.

If you are continuing directly from lesson 4, use the ZAP command so that you will begin with an empty worksheet. Otherwise, start SuperCalc as you learned in lesson 1.

Let's continue to develop the worksheet you saved in lesson 4. Use / L to LOAD the file **WORK1**. You can use ? to get HELP, or check back to lesson 4 if you want a refresher on how to use LOAD.

Suppose that column B, labeled Oranges, really belongs to the right of Apples-3, which is in column E. With what you know already, you can use the COPY command to "move" it there and then use the BLANK command to erase the original Oranges column. You can do that, but there is a better way.

Type / M for move, and notice that the prompt reads:

```
R(ow) or C(olumn)?
```

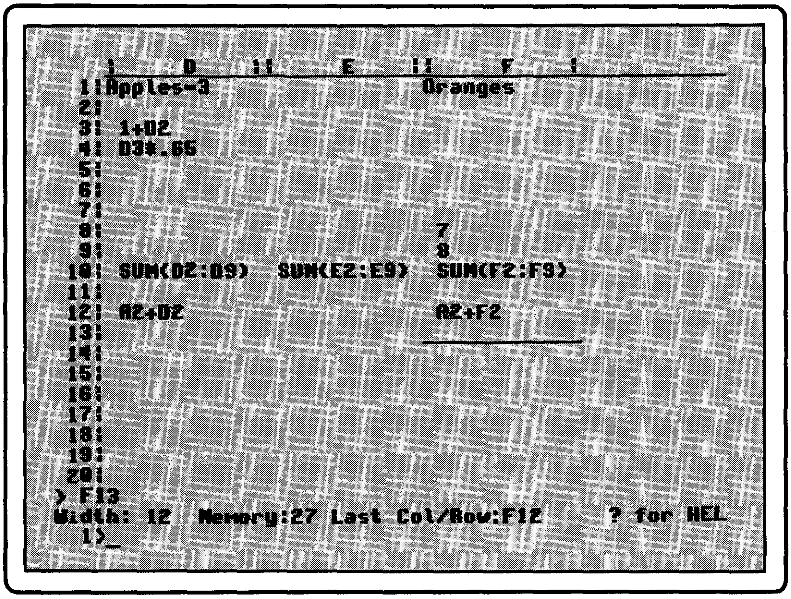
press C (for column), and the prompt changes to:

```
Enter column letter.
```

Since you want to move column B, press B, followed by a RETURN. The new prompt, To?, asks where you want the material to go. Specify F (for column F).

Wait a second, isn't column F already occupied? Press RETURN anyway and note what happens.

Your column has moved and the formulas it contained have adjusted. The "gap" you might have expected column C to leave behind is filled. SuperCalc moved your entries for former columns C through F one column to the left, in effect vacating column F and making it available for use. The program has neatly moved all the columns and adjusted all the formulas to reflect their new locations.



```
      :     D      ::     E      ::     F     :
   1 :Apples-3                  Oranges
   2 :
   3 : 1+D2
   4 : D3*.65
   5 :
   6 :
   7 :
   8 :                             7
   9 :                             8
  10 : SUM(D2:D9)  SUM(E2:E9)  SUM(F2:F9)
  11 :
  12 : A2+D2                     A2+F2
  13 :                          ______________
  14 :
  15 :
  16 :
  17 :
  18 :
  19 :
  20 :
> F13
Width: 12  Memory:27 Last Col/Row:F12      ? for HEL
  1>_
```

INSERT and DELETE Commands

Two other complementary commands that can create or erase intermediate columns and rows are INSERT (/ **I**) and DELETE (/ **D**).

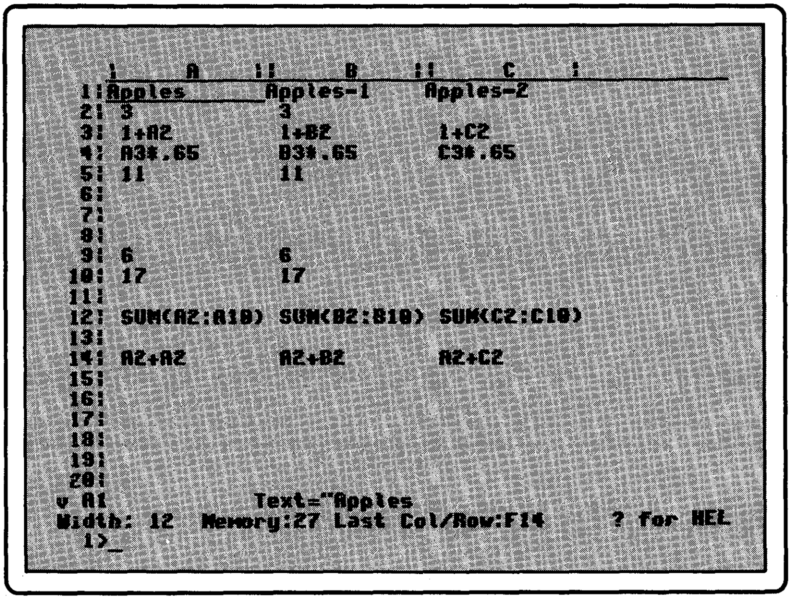
Insert a row between rows 9 and 10.

Type / I, followed by R, for row. Respond to the next prompt by typing the number 1 0, press RETURN and a "new" row appears on your worksheet.

Look at the formulas in row 11; you will see that they are unchanged. SuperCalc can't know if you want to include the new row in the SUM equations—you will have to enter the new row separately if you desire to include it in the summations.

Now insert another row at 7. Type / I R 7 and press RETURN.

Look at the SUM formulas for row 12. They have been adjusted, automatically extended from **A2:A9** to **A2:A10**, because the row we just inserted fell within the range we previously described.

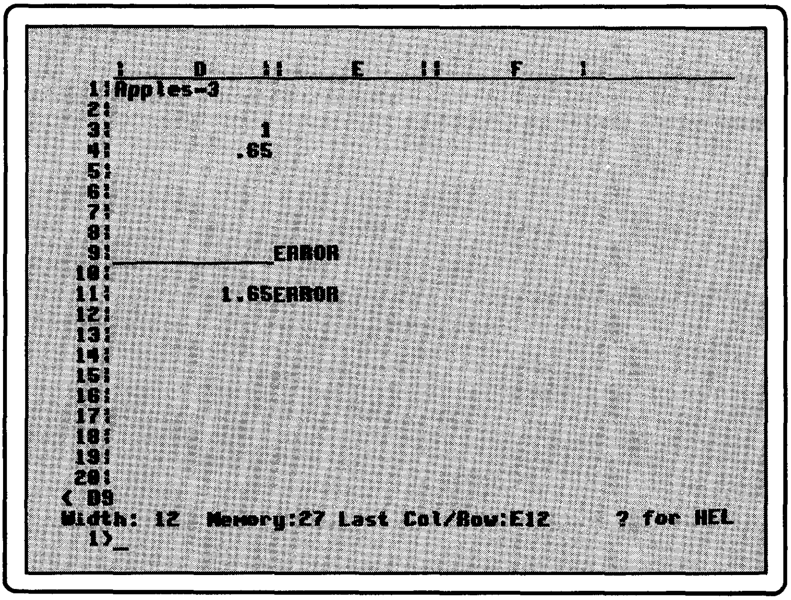


```
    \     A     ||     B     ||     C     |
  1|Apples      Apples-1    Apples-2
  2| 3           3
  3| 1+A2        1+B2        1+C2
  4| A3*.65      B3*.65      C3*.65
  5| 11          11
  6|
  7|
  8|
  9| 6           6
 10| 17          17
 11|
 12| SUM(A2:A10) SUM(B2:B10) SUM(C2:C10)
 13|
 14| A2+A2       A2+B2       A2+C2
 15|
 16|
 17|
 18|
 19|
 20|
v A1           Text="Apples
Width: 12  Memory:27 Last Col/Row:F14     ? for HEL
  1>_
```

Now type / D R. For row number, type 1 4 followed by RETURN. Row 14 is deleted. If you delete row 7, will the SUM formulas adjust back to **A2:A9**? Try it and see.

Delete a column and try an experiment in the process. You want to find out what happens to a value that depends on one you delete. Enter into cell E9 the equation F8—the value contained in cell F8 will repeat in E9. Change the display to show cell values instead of equations.

Now type [/] [D] [C] [F] followed by a [RETURN]—you're deleting column F, and the column entitled Oranges empties. Cell E9 now displays ERROR. SuperCalc has nothing to use in calculating the value of E9 and warns you of this fact with the error message. Once a cell is in error, any reference to it will display a similar error message. As you see, the SUM value also indicates ERROR.



```
   :     D     ::     E     ::     F     :
 1:Apples-3
 2:
 3:           1
 4:         .65
 5:
 6:
 7:
 8:
 9:____________ERROR
10:
11:       1.65ERROR
12:
13:
14:
15:
16:
17:
18:
19:
20:
< D9
Width: 12  Memory:27 Last Col/Row:E12     ? for HEL
  1>_
```

If cell E9 does, in fact, have **F8** in it, you can simply enter that formula again, and everything will be set right. Now put a number or **F8**, whichever you wish, into cell E9. Notice that the error display in the SUM value also goes away; the recalculated value replaces it. You may force recalculation by pressing **!** if necessary.

If you delete row 11, will this affect your range specification for the SUM formulas in row 11? No, because row 11 is beyond the range. Delete row 10.

What will happen if you now delete row 9? Try it.

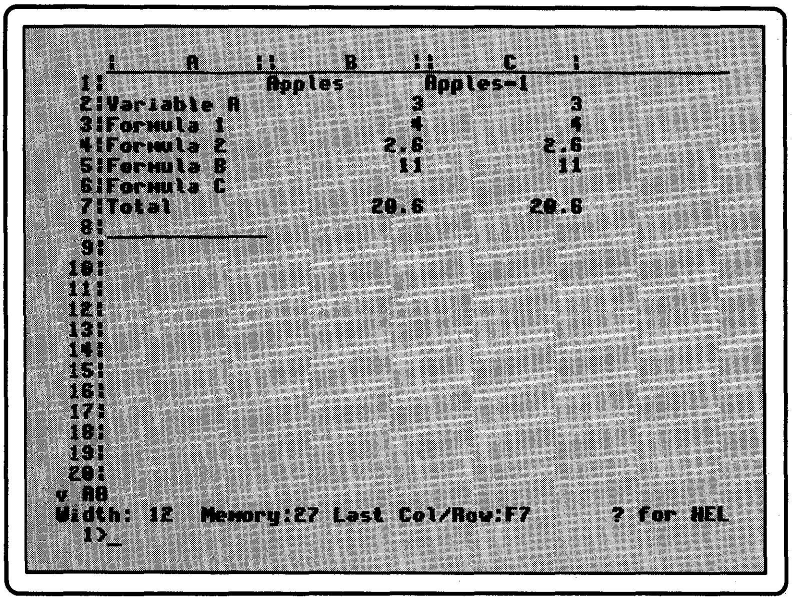
The deletion produces an ERROR in the SUM formula.

The general rule is not to delete either of the boundaries specified in a range like the one in our example. Our example was **SUM(A2:A9)**. Deleting either A2 or A9 will cause an ERROR condition because SuperCalc cannot guess your exact intentions. These warnings help you avoid inadvertently leaving references to nonexistent cells after a DELETE command.

Use the BLANK command to blank out the block from cell A7 to cell E9. Now reenter SUM(A2:A6) in cell A7, and then use REPLICATE to place it in cells B7 through E7. Use the INSERT command to create a new column at A for labels. Next, enter the following text in column A:

cell	text
A2	"Variable A
A3	"Formula 1
A4	"Formula 2
A5	"Formula B
A6	"Formula C
A7	"Total

Here's what you should now see:



```
     :     A     ::     B     ::     C     :
   1:            Apples      Apples-1
   2:Variable A              3           3
   3:Formula 1               4           4
   4:Formula 2             2.6         2.6
   5:Formula B              11          11
   6:Formula C
   7:Total                20.6        20.6
   8:
   9:
  10:
  11:
  12:
  13:
  14:
  15:
  16:
  17:
  18:
  19:
  20:
 < A8
Width: 12  Memory:27 Last Col/Row:F7        ? for HEL
   1>_
```

At this point, use the SAVE command to save your work. This time, let's call it **LESSON5**. You'll use it later.

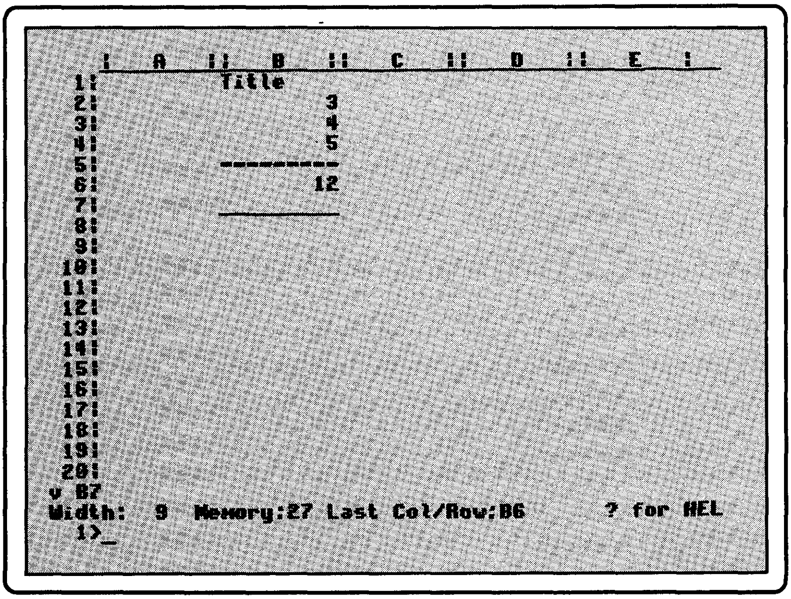
Now that you've saved the worksheet, try something new. You'll need to start with a blank worksheet, so use the ZAP command.

As you've seen, performing insertions and deletions at the boundaries of specified ranges creates problems. Because you will often want to add or delete from lists—including, naturally, the beginning or end of the list—here is a useful suggestion.

Make the following entries to the worksheet:

cell	entry
B1	"Title
B2	3
B3	4
B4	5
B5	"---------
B6	SUM(B1:B5)

You should now see this:



```
    |   A   ||   B   ||   C   ||   D   ||   E   |
  1|         Title
  2|                 3
  3|                 4
  4|                 5
  5|         ---------
  6|                12
  7|
  8|
  9|
 10|
 11|
 12|
 13|
 14|
 15|
 16|
 17|
 18|
 19|
 20|
v B7
Width:  9  Memory:27 Last Col/Row:B6      ? for HEL
   1>_
```

Notice that the range specification includes the title line and the ledger line (“---------). This inclusion is harmless because, in a mathematical sense, SuperCalc regards text as having a value of zero and, therefore, has no effect on the calculation.

Now you can insert or delete with impunity. Insert a new row 5 and enter some numerical value in cell B5. Now delete row 2. Delete row 4. You can add or remove entries without worrying about the top and bottom of your column.

By the way, here is an easy way to put in lines of repeating characters, such as "--------- which you entered in cell B4. SuperCalc has a function to repeat text. Go to cell B4 and press ' - followed by a RETURN.

The single quote (') causes the display of hyphens to repeat and fill the cell display, and, in fact, to continue displaying to the right until it meets a cell with something in it . . . not bad for three keystrokes. Take a look at the contents of cell B4. You will see: **Rtxt=** **'–** which describes the contents as **R**epeated **t**ext.

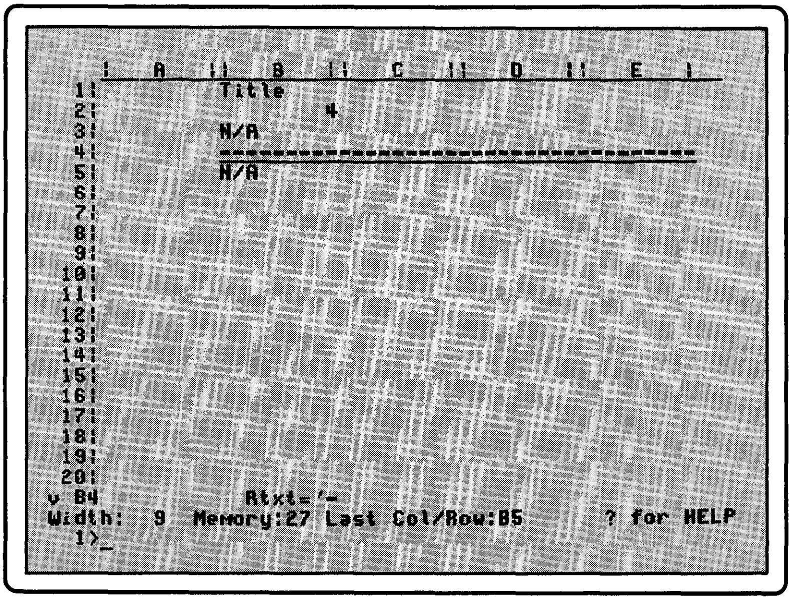
Experiment with this one a bit. Find some open space on the worksheet and try:

```
'123
'abcd
'*
'*
```

Try your name.

Sometimes you may be working on a complex worksheet with many values that are functions of other values. Because your data may be incomplete, you may mistakenly view some totals or values as significant when, in fact, they are not yet complete.

Here's another hint: using the example you started above, enter NA into cell B3, for instance. This action tells SuperCalc that you intend to put a value into that cell in the future, so the value of the cell should be considered as "not available," as opposed to zero. You will see that as soon as you enter **NA**, cell B6, which uses cell B3 in a SUM equation, is also flagged as **N/A**.



```
   |   A   ||   B   ||   C   ||   D   ||   E   |
  1|        Title
  2|                  4
  3|        N/A
  4|        -----------------------------------
  5|        N/A
  6|
  7|
  8|
  9|
 10|
 11|
 12|
 13|
 14|
 15|
 16|
 17|
 18|
 19|
 20|
v B4           Rtxt='-
Width:  9  Memory:27 Last Col/Row:B5        ? for HELP
  1>_
```

N/A and **ERROR** behave similarly; one difference is what is displayed, **N/A** or **ERROR**. By using **NA**, you inform yourself of the ramifications of any incompleteness or oversights. Another difference is that when you replace an NA with a value or formula, SuperCalc will use that new value to recalculate the worksheet; if you replace **ERROR** with a value, nothing else on the worksheet is affected.

You may either quit here or continue to lesson 6.

Lesson 6: Formatting

By now you've learned most of the basic instructions you need to use SuperCalc. It is now time to build on this knowledge and refine it.

When we introduced the FORMAT command (**/ F**) in lesson 3, we used it to change the display width of all the columns on the worksheet. But the prompt line indicated that other options were available with this command. In this lesson, we will examine these other options more closely.

If you are continuing directly from lesson 5, use the ZAP command now so that you will start with a fresh worksheet. Otherwise, start SuperCalc as you learned previously.

Now use / L to load the file LESSON5.

Integer Format

Look at your worksheet. Is it displaying formulas? In this example we will want to look at cell values, not formulas. Use the GLOBAL options command (**/ G**) if you need to change the display.

Look at cell B4. If it does not contain a decimal fraction (for example, 6.4), enter one.

Type / F (for FORMAT) and note that the prompt line now reads:

```
G(lobal), C(olumn), R(ow), or E(ntry).
```

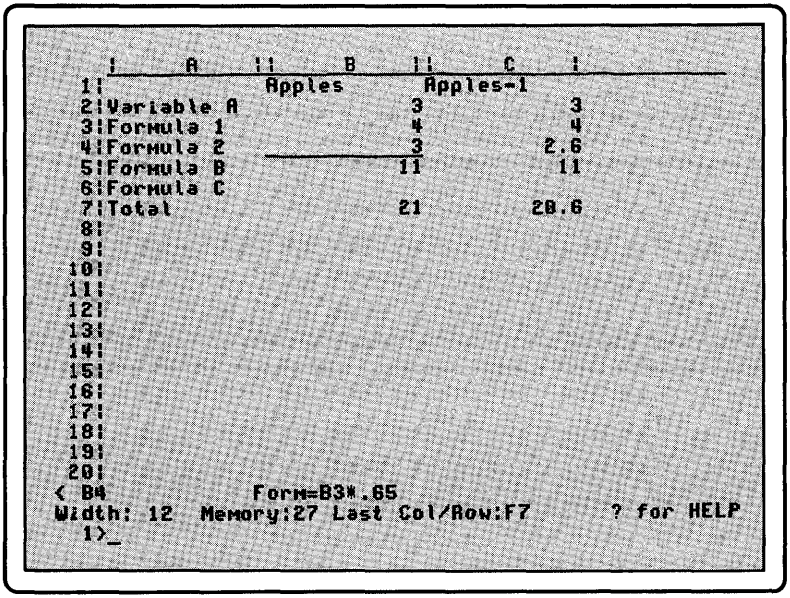
This line means that you can specify whether the format change will affect all cells, a column only, a row only, a cell only, or a range of cells.

Press C for column. The prompt line now asks you to specify which column to affect. Use the column letter—in this case B—and press RETURN.

Now the prompt gives you a great variety of choices:

```
Define Formats: (I,G,E,$,R,L,TR,TL * ,D,column width).
```

We'll examine each of these options eventually, but for now specify I (for INTEGER) and press RETURN. Look at the entries on the display and see what happened to the fractional value you entered earlier—only the integer portion of the values is displayed. Integer format rounds values to the nearest whole number.



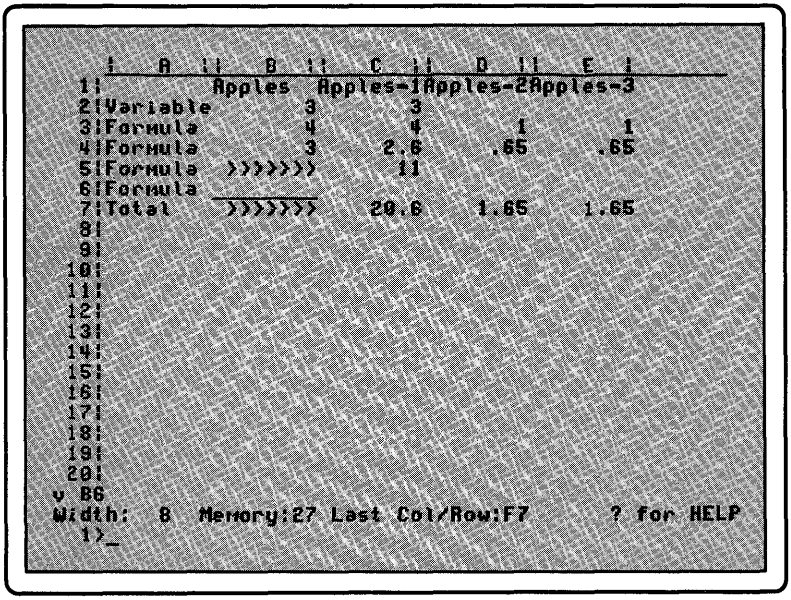
```
    !     A     !!     B     !!     C     !
  1!             Apples       Apples-1
  2!Variable A                3          3
  3!Formula 1                 4          4
  4!Formula 2                 3        2.6
  5!Formula B                11         11
  6!Formula C
  7!Total                    21       20.6
  8!
  9!
 10!
 11!
 12!
 13!
 14!
 15!
 16!
 17!
 18!
 19!
 20!
< B4           Form=B3*.65
Width: 12  Memory:27 Last Col/Row:F7       ? for HELP
  1>_
```

Until now we have always used SuperCalc's standard, or "default," format to display numbers. That format is called the GENERAL format, represented by the letter G in the prompt-line display.

You already know that any number too large to display in ordinary notation will display in scientific or exponential notation. In INTEGER format, numbers too large to display will appear as a series of greater-than signs (> > > >). In fact, whatever the format, > > > >s will appear whenever a number cannot be shown.

Enter 123456789 at cell B5.

Now reduce the display width to 8: / F G 8 RETURN.



```
   |   A   ||   B   ||   C   ||   D   ||   E   |
  1|        Apples  Apples-1Apples-2Apples-3
  2|Variable       3       3
  3|Formula        4       4       1       1
  4|Formula        3     2.6     .65     .65
  5|Formula >>>>>>>>      11
  6|Formula         ________
  7|Total   >>>>>>>>    20.6    1.65    1.65
  8|
  9|
 10|
 11|
 12|
 13|
 14|
 15|
 16|
 17|
 18|
 19|
 20|
v B6
Width:  8  Memory:27 Last Col/Row:F7        ? for HELP
  1>_
```

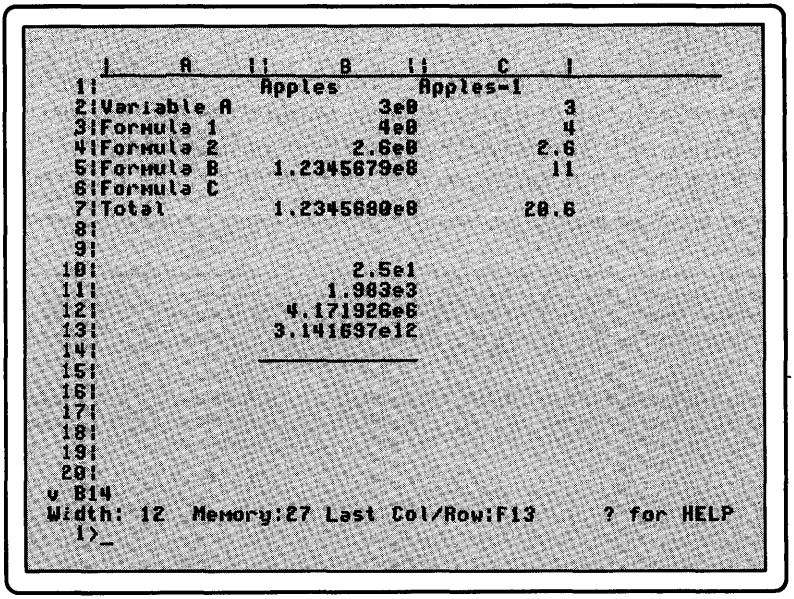
Notice the greater-than signs. Now change the column width back to 12 by typing / F G 1 2 RETURN.

Again type / F C B followed by a RETURN. This time, specify G for the GENERAL format. Notice that the fractional portion of your data values reappears. It is important to note that SuperCalc always maintains the number you enter and merely changes the displayed value based on your requested format. Changing the format does not change the value SuperCalc maintains, only the number it displays.

Exponential Notation, Again

For scientific or exponential notation, try / F C B RETURN E RETURN. This format displays numbers as a power of 10. For example, 1776 is 1.776E3, or 1.776 times 10 to the third power. You should remember this from lesson 2.

Look at your worksheet. As you see, SuperCalc converts all your data in column B to exponential notation, as per your request. If the data does not look familiar to you, you may wish to experiment a bit. Enter some ordinary numbers in column B and watch how SuperCalc displays them.



```
    |     A     ||     B     ||     C     |
  1|           Apples      Apples-1
  2|Variable A          3e0           3
  3|Formula 1           4e0           4
  4|Formula 2         2.6e0         2.6
  5|Formula B  1.2345679e8           11
  6|Formula C
  7|Total      1.2345680e8         20.6
  8|
  9|
 10|                  2.5e1
 11|                1.983e3
 12|            4.171926e6
 13|           3.141697e12
 14|
 15|
 16|
 17|
 18|
 19|
 20|
v B14
Width: 12  Memory:27 Last Col/Row:F13     ? for HELP
  1>_
```

Dollars-and-Cents Format

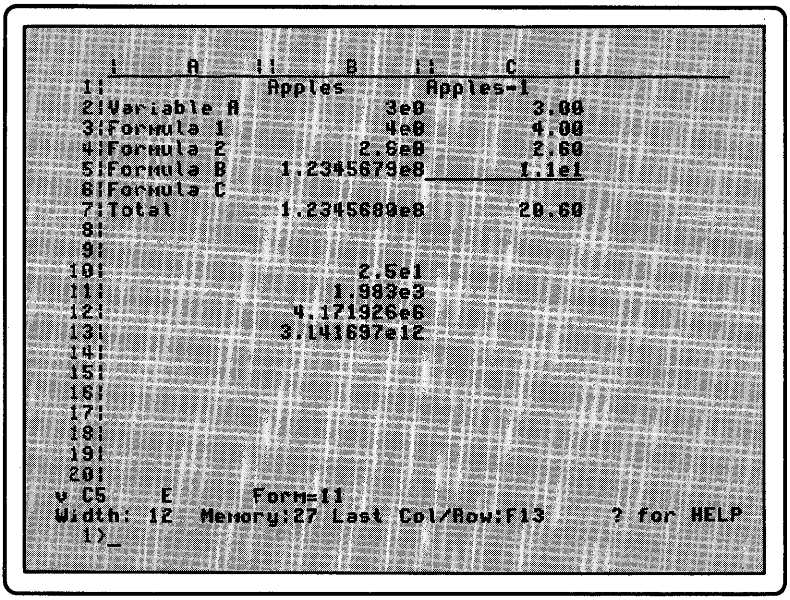
The next format option is a familiar one: [/] [F] [G] [$] [RETURN]. Your data now formats as dollars and cents. Displayed numbers are rounded to the nearest cent. Note also that SuperCalc adds **.00** to whole numbers, but does not insert a leading dollar sign.

Except those cells in which you changed formats, everything changed. Why? When GLOBAL is indicated, SuperCalc changes all formats, except those that you have already specified using column, row, or entry options. It leaves these formats untouched because you intentionally set them individually.

Now change the format for a single cell. Move the worksheet cursor to cell C5, making that the active cell. Type [/] [F] [E]. The prompt line reads Enter Range.

You can specify a range of cells—that is, a partial row or a partial column—or you can simply specify a single cell. Change the format of cell C5, the active cell. You can type **C5** to do this, but instead press RETURN and see what happens.

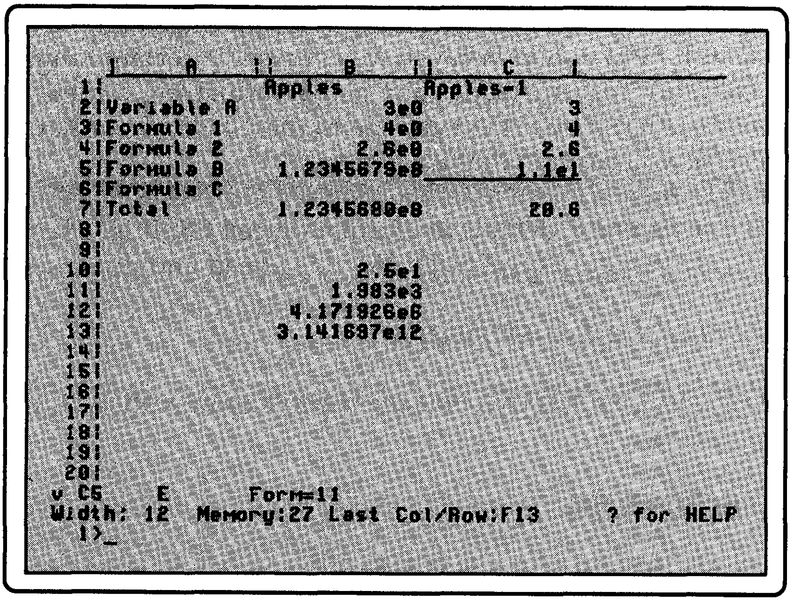
If the result surprises you, think back to when we introduced the action of the ESCAPE key; at that time we said that RETURN would generally fill in a part of a command with the number of the active cell. Finish off the entry by pressing E (for EXPONENT), followed by another RETURN. Note the change on your worksheet.



```
    !     A      !!      B      !!      C      !
  1!             Apples        Apples-1
  2!Variable A            3e0          3.00
  3!Formula 1             4e0          4.00
  4!Formula 2           2.6e0          2.60
  5!Formula B     1.2345679e8________1.1e1
  6!Formula C
  7!Total         1.2345680e8         20.60
  8!
  9!
 10!                    2.5e1
 11!                  1.983e3
 12!               4.171926e6
 13!              3.141697e12
 14!
 15!
 16!
 17!
 18!
 19!
 20!
v C5     E        Form=11
Width: 12  Memory:27 Last Col/Row:F13       ? for HELP
  1>_
```

Now suppose you want to convert all the display back to the GENERAL format. Can you make a GLOBAL change? Sure—just issue the command / F G G RETURN (/FORMAT, GLOBAL, GENERAL, RETURN).

Everything changes, except the cells in which you've changed the formats.



```
     |      A      ||      B      ||      C      |
   1|             Apples       Apples-1
   2|Variable A             3e0               3
   3|Formula 1              4e0               4
   4|Formula 2            2.6e0             2.6
   5|Formula B    1.2345679e8             1.1e1
   6|Formula C
   7|Total        1.2345680e8              20.6
   8|
   9|
  10|                     2.5e1
  11|                   1.983e3
  12|                4.171926e6
  13|               3.141687e12
  14|
  15|
  16|
  17|
  18|
  19|
  20|
v C5      E         Form=11
Width: 12  Memory:27 Last Col/Row:F13      ? for HELP
  1>_
```

So that GLOBAL changes include any column, row, or cell you formatted individually—column B, for example—you must "undo" the individual formatting.

Position the active cell anyplace in column B. Type / F C RETURN, then D (for default), and another RETURN. Notice that column B changes to GENERAL format except for its width, which may remain 12 characters wide.

When a format setting that refers to a column or row defaults, it changes back to the original setting, which is text left-justified and numbers right-justified. Format settings function in a hierarchical fashion, meaning that an entry-level format, entered as a cell or a range of cells, is the "highest" level. The full list of levels looks like this:

highest level →	entry (individual entry)
	row
	column
lowest level →	global

After the default command, column B formatting changed to the default format, and because we specified a certain range (the column) this supersedes any global formatting commands. Remember that if you don't change to the GLOBAL format, SuperCalc will assume that it's general.

See if you can now default the format on cell C5. It is, of course, / F E C 5 RETURN D RETURN. Go ahead and do it.

Right and Left Justification

Type / F R 1 RETURN. You see the following options on the prompt line:

```
...R,L,TR,TL...
```

With them, you can change the setting to right or left justification. The standard, or default, values are left-justified text and right-justified numbers. Shift the text on row 1 so that all text entries are right justified. Hint: **TR** stands for "text right."

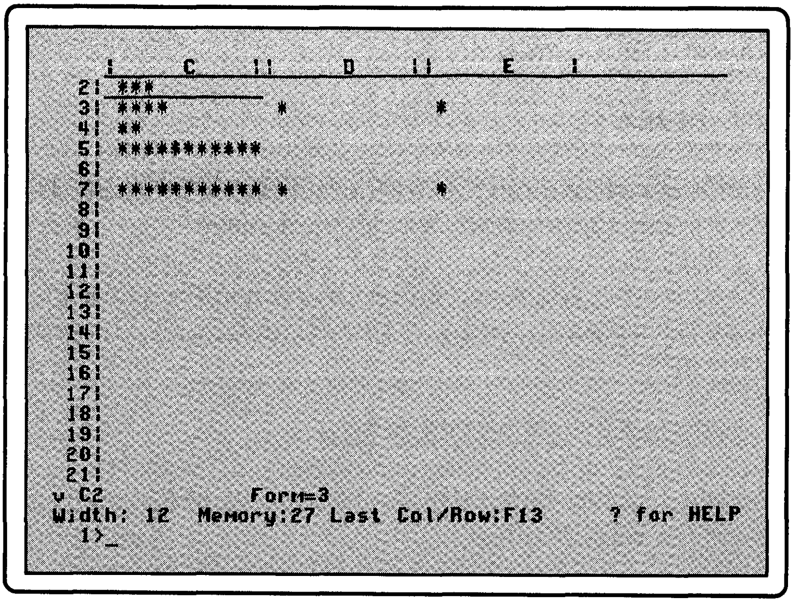
Now that you've done that, try another one.

Type / F G L RETURN. All numbers are now justified to the left.

You can combine format entries. / F G R $ RETURN right-justifies numbers in the dollars-and-cents format, for example. Try experimenting with various format combinations.

GRAPHIC Display

There's one last FORMAT option to try, the GRAPHIC display option. Place the active cell at C2. Type = RETURN to place cell C2 at the upper left of your screen. Then type / F G * RETURN. The * specifies GRAPHIC display.



```
    |     C      ||     D      ||     E      |
   2| ***
   3| ****        *             *
   4| **
   5| ***********
   6|
   7| *********** *             *
   8|
   9|
  10|
  11|
  12|
  13|
  14|
  15|
  16|
  17|
  18|
  19|
  20|
  21|
v C2          Form=3
Width: 12  Memory:27 Last Col/Row:F13      ? for HELP
   1>_
```

Depending on what you enter, you may have entries that no longer fit in the current column size. If this troubles you, you can specify a column width of anything up to and including 126 characters, although you will not be able to see all of these in the current version of SuperCalc.

We'll come back to the graphic display in lesson 8 and show you a way to make the graph fit within a column width of convenient size. For now, there is no need to save the work you did in this lesson, so you may either quit or continue with lesson 7.

Lesson 7: TITLE LOCK and WINDOW

You know enough about SuperCalc and its many commands to put it to practical use. This lesson adds two more advanced commands to your store of tools.

One of the commands, TITLE LOCK, is useful if you want to keep any portion of the worksheet locked in place while you scroll the rest of the screen. The other command, WINDOW, lets you "split" your screen and look at different parts of your worksheet at the same time.

TITLE LOCK

If it is not already running, start the SuperCalc program. Use the ZAP command to obtain a fresh worksheet.

Now load the file you saved under the name **LESSON5**. Place the active cell at A1. Issue / T (for TITLE LOCK). The prompt line now asks:

```
H(oriz), V(ert), B(oth), or C(lear)?
```

SuperCalc wants to know which titles you want locked in place. Press V for vertical titles.

Now scroll the screen and move off the screen to the right. You will see the titles at screen left "locked" in place, while the rest of the screen scrolls as usual. The position of the active cell when you enter the TITLE LOCK command determines how much of the screen will lock in place. For example, if the worksheet cursor is in row 8, everything in row 8 and above will be locked similarly; if the location of the worksheet cursor is column 4, everything in column 4 and to the left will lock.

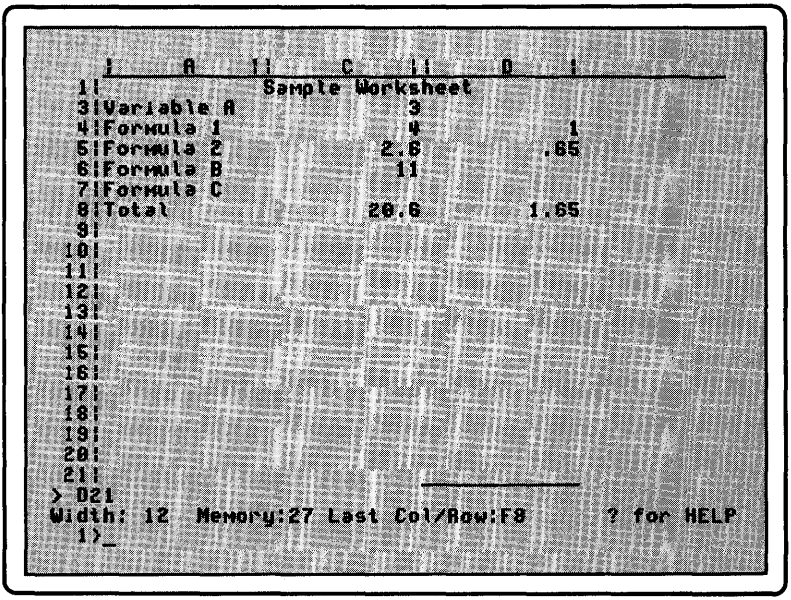
With the **H** option, lock the top row of titles in place.

Move the worksheet cursor down the screen and watch the information scroll up while the row 1 titles stay in place. Next go back to cell A1 by using the GOTO command (=).

Clear the locked row. Type / T, and then C. You are telling SuperCalc that you do not want to lock anything. Use the INSERT command to create a new row 1 for an additional title. At cell C1, type:

```
"Sample Worksheet
```

This time lock both the horizontal and vertical titles with one command. Position the active cell at A2. Type $\boxed{/}$ $\boxed{T}$, followed by $\boxed{B}$ (for BOTH). This locks column A and rows 1 and 2.



```
     |     A     ||     C     ||     D     |
   1|               Sample Worksheet
   3|Variable A              3
   4|Formula 1               4           1
   5|Formula 2             2.6         .65
   6|Formula B              11
   7|Formula C
   8|Total                20.6        1.65
   9|
  10|
  11|
  12|
  13|
  14|
  15|
  16|
  17|
  18|
  19|
  20|
  21|                       _______________
> D21
Width: 12  Memory:27 Last Col/Row:F8       ? for HELP
   1>_
```

Move the worksheet cursor down and to the right and verify that the display scrolls in both directions while rows 1 and 2 and column A stay put.

WINDOW—Split Screen

What if you want to view two widely separated areas of your worksheet at the same time? The WINDOW command allows you to do this. We will use one of the sample worksheets on your SuperCalc diskette to demonstrate the WINDOW command.

Before loading the sample worksheet, make sure you've erased everything from the current worksheet by using the ZAP command.

Now load the sample worksheet. Type the LOAD command ($\boxed{/}$ $\boxed{L}$) and use the file name BALANCE. **BALANCE** is a complete sample SuperCalc worksheet. For now, just scroll to column N and notice that you see columns representing months and a "total" column for the entire year.

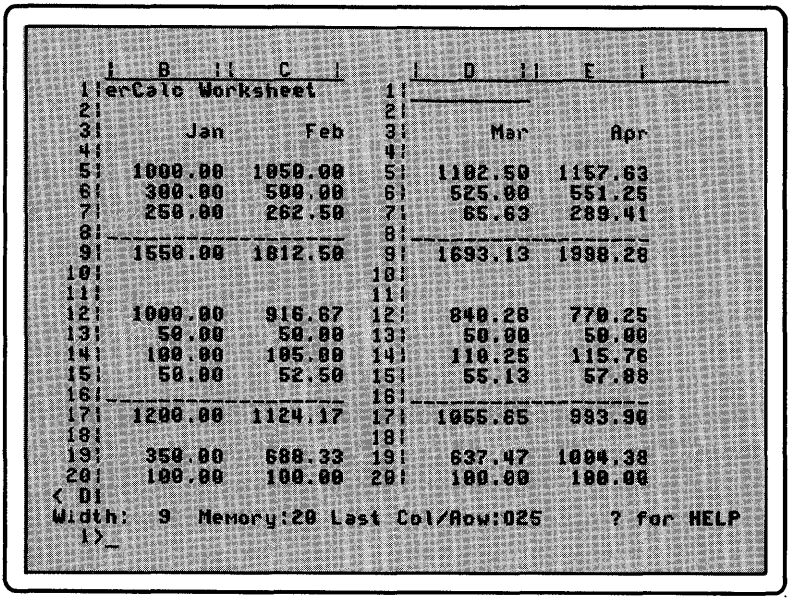
Go back to cell A1 (use GOTO) and scroll down to row 25 to see Net Income. Next go to cell A1, then move the active cell to column D and scroll the screen to position column D in the center; do this by hitting the right arrow once, then the left arrow once. This will designate the place to "split" the screen.

Type [/] [W] (for WINDOW). The prompt now reads:

```
H(oriz), V(ert), C(lear Split), S(ynch), or U(nsynch)
```

You are going to split the screen vertically into two separate display windows, so press [V] (for VERTICAL).

Notice that, starting at column D, there is a second set of row numbers. This is the left-hand border of the second display window. The worksheet itself hasn't split, though, you have simply created two display windows through which to view it. Either window may now scroll independently.



```
    |    B    ||    C    |     |    D    ||    E    |
   1|erCalc Worksheet        1|_________
   2|                        2|
   3|      Jan      Feb      3|      Mar      Apr
   4|                        4|
   5|  1000.00  1050.00      5|  1102.50  1157.63
   6|   300.00   500.00      6|   525.00   551.25
   7|   250.00   262.50      7|    65.63   289.41
   8|------------------      8|------------------
   9|  1550.00  1812.50      9|  1693.13  1998.28
  10|                       10|
  11|                       11|
  12|  1000.00   916.67     12|   840.28   770.25
  13|    50.00    50.00     13|    50.00    50.00
  14|   100.00   105.00     14|   110.25   115.76
  15|    50.00    52.50     15|    55.13    57.88
  16|------------------     16|------------------
  17|  1200.00  1124.17     17|  1055.65   993.90
  18|                       18|
  19|   350.00   688.33     19|   637.47  1004.38
  20|   100.00   100.00     20|   100.00   100.00
< D1
Width:  9  Memory:20 Last Col/Row:025       ? for HELP
   1>_
```

Scroll the display and notice that the left-hand window remains still.

Now press the ; key which will transfer you to the "other" window, regardless of which window you are working with.

Instead of splitting the screen vertically into right and left halves, you can split it horizontally. First clear the current split: type / W C. Now set the active cell at the point you wish the screen to split horizontally. For example, move the active cell to row 15, then issue / W H and, if necessary, RETURN.

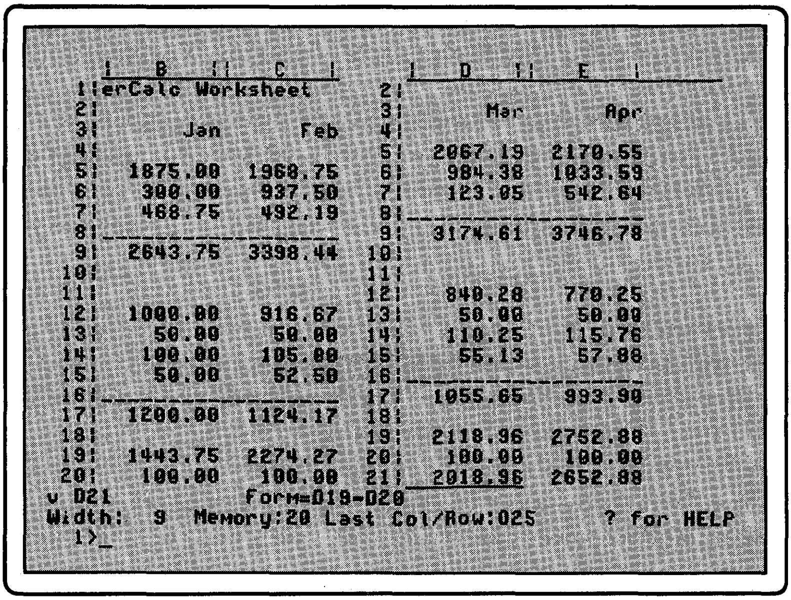
You'll notice that the lower screen starts at row 15, at your cursor. Scroll down to see the Net Income entry. Press ; to place the cursor in the upper screen, and move to cell B5. Change the value there. Watch as the recalculation takes place. Within moments you will see the net-income figure change in the lower window. When you wish to remove the split screen, type / W C and, if necessary, RETURN.

The **S** option indicates to the SuperCalc program that you wish to scroll both windows in a "synchronized" fashion (simultaneously). Try it.

Split the screen vertically at column D again, but now use / W S. The displays scroll together. Verify this, then "unsynchronize" them by using / W U. (Figure next page.)

With the split screen in effect, each window has its own "global" identity for both the GLOBAL options and FORMAT commands. For instance, you could specify FORMULA display in one window and CELL VALUE display in the other. Similarly, you could use FORMAT to specify GENERAL format in one window and INTEGER format in the other. You could even look at the same data in two different formats at once.

Again, type / W U RETURN. Scroll both displays to show January through April. Now change to display formulas for one side of the screen. Type / W S RETURN. Now you can scroll through the data in one window and compare it to the formulas as you go.



```
    |   B   ||   C   |      |   D   ||   E   |
  1|erCalc Worksheet     2|
  2|                     3|      Mar      Apr
  3|      Jan      Feb   4|
  4|                     5|  2067.19  2170.55
  5|  1875.00  1968.75   6|   984.38  1033.59
  6|   300.00   937.50   7|   123.05   542.64
  7|   468.75   492.19   8|------------------
  8|------------------   9|  3174.61  3746.78
  9|  2643.75  3398.44  10|
 10|                    11|
 11|                    12|   840.28   770.25
 12|  1000.00   916.67  13|    50.00    50.00
 13|    50.00    50.00  14|   110.25   115.76
 14|   100.00   105.00  15|    55.13    57.88
 15|    50.00    52.50  16|------------------
 16|------------------  17|  1055.65   993.90
 17|  1200.00  1124.17  18|
 18|                    19|  2118.96  2752.88
 19|  1443.75  2274.27  20|   100.00   100.00
 20|   100.00   100.00  21|  2018.96  2652.88
v D21          Form=D19-D20
Width:  9  Memory:20 Last Col/Row:025      ? for HELP
  1>_
```

The WINDOW and TITLE LOCK commands affect the way your worksheet is displayed. The effect is temporary, and you can always reverse it. When you save your worksheet on a diskette, the TITLE LOCK and WINDOW information is included; if you reload the worksheet, it will look exactly as it did before.

If you want to take a break now, use QUIT to exit from SuperCalc; otherwise, continue with lesson 8.

Lesson 8: Graphic Formats and Recalculations

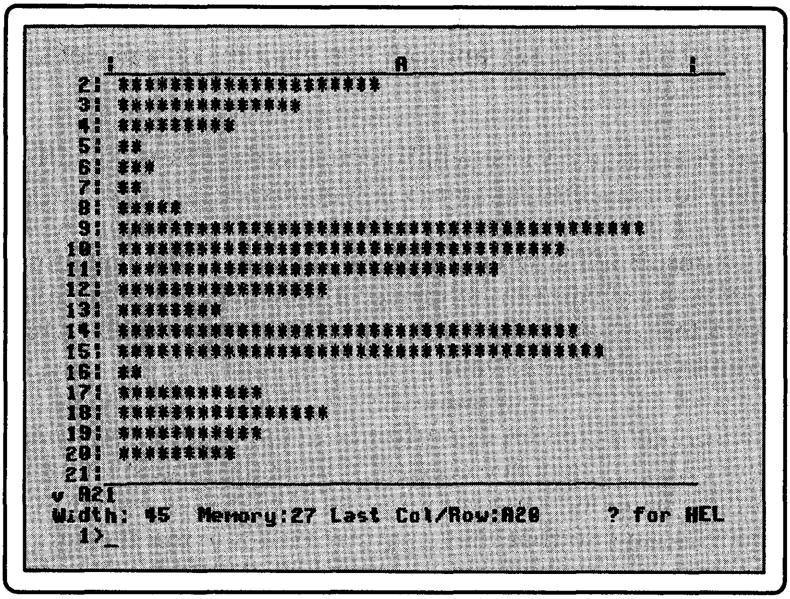
In this lesson we will look more closely at some of the options available with two of SuperCalc's most powerful commands, FORMAT and GLOBAL OPTIONS.

Start with a fresh screen—use ZAP to clear the screen if you are continuing from a previous lesson.

We touched briefly on graphic representation of data in lesson 6 and promised, in time, to tell you more about it. That time has arrived. After working more with graphic representation, you should feel confident enough to use it with your own data.

Start by entering some numbers in column A, from row 2 to row 20. Use any numbers between 1 and 45.

Begin with this sequence: [/] [F] [C] [A] [RETURN] [*] [4] [5] [RETURN] (this stands for /Format,Column,A,RETURN,*,45,RETURN).



```
   :                   A                    :
 2: **********************
 3: ***************
 4: **********
 5: **
 6: ***
 7: **
 8: *****
 9: *********************************************
10: **************************************
11: ********************************
12: ******************
13: *********
14: ***************************************
15: *****************************************
16: **
17: ************
18: ******************
19: ************
20: **********
21:
v A21
Width: 45  Memory:27 Last Col/Row:A20      ? for HEL
  1>_
```

You have accomplished two things: you've changed to graphic display (by using the *), and you have increased the width of column A to 45 characters.

Say you now want to have the number display, in addition to a graphic representation. Try this: [/] [F] [C] [A] [RETURN] [D] [RETURN] and [/] [G] [F]. At cell B1 enter the FORMULA A1. Then type [/] [R] [B] [1] [,] [B] [2] [:] [B] [2] [0] [RETURN]. Finally: [/] [F] [C] [B] [RETURN] [*] [4] [5] [RETURN] and [/] [G] [F].

Move the cursor to column B to check your results.

We'll explain what you did. First you returned column A to the DEFAULT format. Then you asked SuperCalc to show FORMULAS. Next, you entered the FORMULA **A1** in cell B1 and replicated that FORMULA through the rest of column B. Last, you told SuperCalc to show graphic representation in column B. Essentially, you duplicated column A in column B, then formatted column B to show graphic representations of the numbers in column A.

You now have a one-for-one graphical display in column B of the numbers in column A. But what if the values you wish to display are as large as 600 or 1000? Do you have to make the column widths 600 or 1000, and can you even make sense of such a display? No.

So that your largest value will be equal to the column width, put a scaling formula into column B. If you divide any value in column A by the maximum value within your sample from A1 to A20, the result will express its SIZE (relative to the maximum). And since the maximum, whatever it is, will be represented by 45 characters (*) of display, you can multiply the size by 45 to determine the SCALED value.

This action gives you an opportunity to use another built-in function: MAX. The value of MAX will be the largest value within the specified range or list. Use MAX to scale your graphic displays so that they are relative to the maximum value. Here is how the formula for cell B1 should look:

```
A1 * 45/MAX(A1:A20)
```

We're using cell A1 in the formula because we'll use the REPLICATE command to fill in the rest of column B. Type / G F RETURN, move the cursor to cell B1 and enter:

```
A1*45/MAX(A1:A20)
```

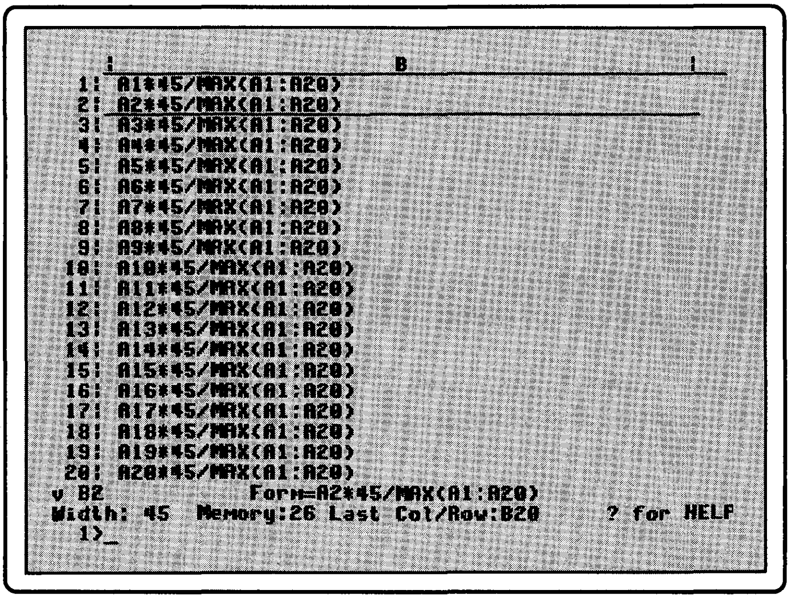
Now use the REPLICATE command with the **A** (for ADJUST) option:

/ R B 1 , B 2 : B 2 0 , A

wait, then type Y N N

Here you've responded **Y** to adjust A1 automatically, and **N** to keep the rest of the formula unchanged.

Use / G F to see your formulas:



```
      !                        B                        !
    1! A1*45/MAX(A1:A20)
    2! A2*45/MAX(A1:A20)
    3! A3*45/MAX(A1:A20)
    4! A4*45/MAX(A1:A20)
    5! A5*45/MAX(A1:A20)
    6! A6*45/MAX(A1:A20)
    7! A7*45/MAX(A1:A20)
    8! A8*45/MAX(A1:A20)
    9! A9*45/MAX(A1:A20)
   10! A10*45/MAX(A1:A20)
   11! A11*45/MAX(A1:A20)
   12! A12*45/MAX(A1:A20)
   13! A13*45/MAX(A1:A20)
   14! A14*45/MAX(A1:A20)
   15! A15*45/MAX(A1:A20)
   16! A16*45/MAX(A1:A20)
   17! A17*45/MAX(A1:A20)
   18! A18*45/MAX(A1:A20)
   19! A19*45/MAX(A1:A20)
   20! A20*45/MAX(A1:A20)
v B2           Form=A2*45/MAX(A1:A20)
Width: 45  Memory:26 Last Col/Row:B20     ? for HELP
  1>_
```

Typing / G F will return to the graphic display. Your graph looks the same so far—it should, since no value exceeds 45. Now change the value of any cell in column A to, say, 75. Notice that all the other lines are scaled relative to 75. Now enter 150.

You may wish to save this example for your own use later. Use SAVE and call the file **GRAPH**, or something easy to remember.

Now change your formula to scale from the minimum to the maximum value in A1 through A20. GoTo cell B1 and use EDIT. You want to insert new information into the formula so that it reads:

```
(A1-MIN(A1:A20) * 45/MAX(A2:20)
```

Replicate this new formula for cells A2 through A20, using the ASK-FOR-ADJUST option. Be careful to adjust only the first cell reference (**A1**) in the formula.

Type / G F. Notice how the results of this formula differ from those of your first formula. Try different values in column A to test and verify how this new formula works.

Recalculation Options

If you enter a new value in column A, notice that the program takes a long time to go through all necessary recalculation of formulas. It may take even longer with a larger worksheet because SuperCalc recalculates automatically every time you enter a new number.

You can suspend that automatic recalculation. Type / G M. Now try entering new numbers in column A for the graph. As you can see, the time required for their entry is reduced.

This is fine, but what does MANUAL recalculation mean? Certainly you don't want to do it yourself with pencil and paper, you want SuperCalc to do it. You haven't yet learned the option that performs recalculation, but it's a familiar mark of punctuation.

Besides serving its usual exclamatory function in textual material, the ! key forces a recalculation. Try it.

Manual mode allows you to make periodic recalculations at your convenience. When you wish to reestablish automatic recalculation, type / G A (for AUTOMATIC).

Order of Recalculation

When SuperCalc recalculates, it does so in a certain order, which you can change. Usually, the order of calculation won't affect the results on your worksheet, and you can ignore it. But there are times when it can make a difference. Observe.

First, use ZAP to get a fresh worksheet.

Enter 4 into cell A1, 6 into cell A2, and

```
SUM(A1:A2)
```

into cell A3. Next enter A3 into cell B1.

Look at the values. Everything seems fine. Cells A3 and B1 both display the value of 10. Now change the value in cell A1 to 3.

Cell B1 does not yet contain the new result, **9**, and that is because SuperCalc normally recalculates row by row: first row 1, then row 2, then 3, and so forth. Obviously, A3 was still **10** when cell B1 referenced its value during recalculation.

Now type / G. The prompt line reads:

```
F(orm.),N(ext),B(order),T(ab),R(ow),C(ol.),M(an.),A(uto)?
```

In the example, recalculation should proceed column by column, so press C to change the order of recalculation.

Enter 5 at cell A1. Now everything seems to work because SuperCalc is proceeding down columns as it recalculates. Both cell A3 and cell B1 display 11.

It is possible to create a situation in which neither order of calculation can give current values in all cells. Here is an example:

ZAP the worksheet
in cell A1 enter 5
in cell C1 enter A1
in cell A3 enter A1
in cell B2 enter C1+A3
in cell A1 enter 4

Can you see the problem? Cells C1 and A3 display 4, and are correct, but cell B2 has a **9** in it.

Issue / G C; in cell A1 enter 6.

In cells A3 and C1 you see **6**, but cell B2 shows 10. One of the cells in the formula for B2 has the correct value of **6**, while the other has a leftover value of **4**.

Press !. Now cell B2 has the correct value of **12**. You forced a second recalculation to get the correct result.

This example is unrealistic and improbable—still, you should know that you can create situations involving out-of-order references that give misleading values.

Cases of out-of-order references, such as this, are called "forward references" because the reference is forwarded to a value not yet recalculated. They can occur in actual worksheets, perhaps because a worksheet is especially complex or because you've changed it from its original design.

A real-life example of a forward reference might happen like this: you build a worksheet with a table of expenditures by category (columns) and locations (rows). You sum the rows and columns to get totals. Everything works fine. Later, you add a table comparing various category and location totals. Everything still works fine, because you know where the second table should be. Then, someone else adds new material to the worksheet and moves one of your tables to a new location. Now the comparison table shows incorrect values, but they might seem reasonable.

One way to check for such cases is to press $\boxed{!}$ and see if any value changes. If so, it is time to redo the worksheet.

Another case you want to avoid is the "circular reference." Here is an example:

> ZAP the worksheet
> in A1 enter 1+B1
> in B1 enter 1+A1

Press $\boxed{!}$ a few times and watch the values in the two cells increase. They will never stop changing because there is no logical place to stop calculating.

When you're satisfied you understand the circular reference, you can quit or ZAP the circular references and go on to the next lesson.

Lesson 9: OUTPUT

You've now worked with all of the SuperCalc commands except for OUTPUT. This command makes it possible to generate printed reports from your worksheets.

The OUTPUT command makes a copy of your worksheet and sends that copy to any of three places, depending on your specification. You can send the "output"—the copy of all or part of your worksheet—to a printer, which will print it out immediately; or you can send it to the screen, where it will temporarily replace the usual SuperCalc display; or you can send the output to a diskette. In this last case, the output will be "saved" as a special sort of disk file, different from the ones you created in the past with the SAVE command.

NOTE

Look in the Appendix for information on installing your printer. Also consult the section on the System Setup program in chapter 2.

Try this new command. First, be sure that you have a fresh worksheet. Next, load the file that you created in lesson 5 (**LESSON5**).

Type / O (for OUTPUT). Now the prompt line reads:

```
D(isplay) or C(ontents) report.
```

"Display" means that the output will reproduce exactly what you see on the screen, including the column and row numbering. So try that first. Press D.

You see that the prompt line requests the range of material you want as output. Specify the range as usual. In this case, specify A 1 : F 8, which describes the boundaries of your work. Press RETURN to complete the command.

The prompt now reads:

```
Enter Device: P(rinter),S(etup),C(onsole),or D(isk)
```

Press C to send the report to the console screen. There may seem to be no reason to do this, but sometimes you may want to check your output before printing it. By the way, if you have several pages of output, SuperCalc shows them to you one at a time; you tell the program when to show the next one.

Press any key to return to the normal SuperCalc display. If you have a printer, try sending output to it—print just part of your worksheet. Be sure that the printer is turned on and connected properly to the Osborne 1, then type:

/ O D A 1 : D 8 RETURN P RETURN

So far, the examples have used only the **D**, or DISPLAY, option. Try **C**, for CONTENTS output. Type:

/ O C A 1 : E 7 RETURN P RETURN

The **C**(ontents) option gives you a list of all cell contents—what is actually in each cell. Because of display formatting, the contents may be quite different from what you see on the worksheet. You are familiar with the idea of cell contents because the contents of the active cell regularly display on the status line.

The **D**(isk) option, in some ways, is similar to the SAVE command. The resulting disk file is different, however; you cannot use the LOAD command to reload a Disk file in SuperCalc. The files created with this OUTPUToption automatically receive a special file type of **.PRN** (stands for "print").

Print files can be useful. You can use them to mix SuperCalc's output with WordStar—as when you are preparing a report in which you need calculations—so you may use SuperCalc to create the numeric sections of any text report you are preparing. Use the **D** option to save the worksheet in a form WordStar can use. When you get to the point in WordStar where you want to insert the worksheet, simply use the **^KR** option to read the print file you created with SuperCalc.

The **S**(etup codes) option is used to output select codes that affect the printer being used. When you press S from the OUTPUT mode, a list of parameters that can be controlled as well as their current settings are shown as follows:

```
L  =  Change page length              (now 66 lines)
      (length zero for continuous form)
W  =  Change page width               (now 80 chars)
S  =  Manual setup codes
P  =  Print report
      CTRL-Z to cancel /O command
```

To alter the page dimensions, use **L** for length or **W** for width and specify the desired number of characters. The Setup codes are those recognized by your printer and described in your printer documentation. These codes perform such print functions as compression and graphic displays.

Lesson 10: Creating Command Files

Command files contain instructions used by the **/X** (execute) command. The character strings contained in the file represent exactly the characters that would be typed at your terminal while in the SuperCalc program.

This is how a command file is created:

Command files can be created using SuperCalc or through the "N" command in WordStar. Each line of the file contains exactly the keys you would press to execute a specific command within the SuperCalc program. Every operation available to you in the SuperCalc program is also available for use in an executable command file. This includes cursor movement (represented by ↑, ↓, →, ←, for up, down, right, and left), and data entry. One exception to this is the **/E** (edit) command which should only be used as the last command in your command file.

Now let's create a file of commands.

The first thing you might like to do in your command file is to **/ZAP** the worksheet. If we were to enter this command in the SuperCalc program, the entry line would look like this:

```
24 > /ZAP-ENTIRE-Worksheet? Y
```

The keys that were actually pressed to accomplish this were [/] [Z] [Y]. Let's try another. If we were to format column A to be 20 characters wide, the entry line would look like this:

```
21 > /Format, Column, A,20
```

The keys that were actually pressed were / F C A , 2 0. So, on the second line of the command file you would have **/FCA,20**. Each line represents the exact keys pressed for a specific command. Use the N command in WordStar to build the following command file:

```
/ZY
/FCA,20
/LBALANCE,A
/GF/GM/FGD,$
/IR23
=A23
"Tax Rate
/P
>IF(B21<=1800,.3,.32)
/RB23,C23:N23
=A24
/U
"Taxes
</P
>B23*B21
/RB24,C24:N24
/GF!/ODALL,C
```

You will notice that on line 4 of our command file we have combined three commands. This is because carriage returns are not required after /GF and /GA for execution. Using unnecessary carriage returns would advance the worksheet cursor spuriously, thus easily losing your place on the worksheet.

If you wanted to build the command file within SuperCalc, each command line is entered as text, then the file is saved. You should save your SuperCalc command file in two segments: 1) by using the /Output command and creating a .PRN print file that can be read by the /X command, and 2) by using the /SAVE command to save the file for editing later. If you have not saved a .CAL-type file, you will not be able to reedit the file. .PRN-type files are not loadable by the SuperCalc program. Before creating the .PRN-type command file, be sure to remove its borders otherwise the EXECUTE command will not be able to load it.

To execute the command file, issue [/] [X] from SuperCalc and you will be prompted like so:

```
Enter filename (or < RETURN > for directory)
```

If you press RETURN, you will be given the options for the directory as in the /D(elete) command. If you enter a file name, SuperCalc reads each of the commands in the file and executes them in sequence. If the file is not in the proper format or an improper command is used, an error message is displayed on the status line and the EXECUTE command is aborted. You can also terminate the execution of the command file at any time by pressing ^[Z].

NOTE

The default extension for command files is .QXT. If your file has no extension, you must have a period at the end of the file name.

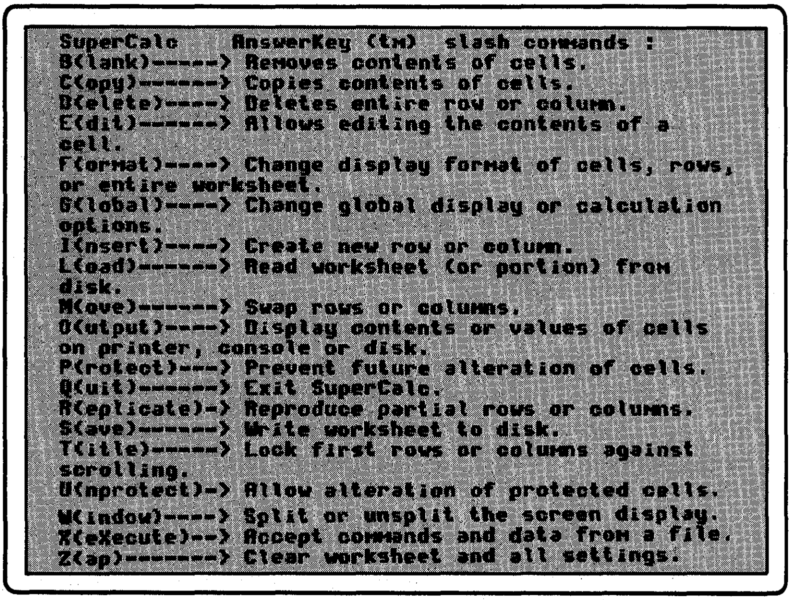
Some Last Words on SuperCalc

By now you know enough about SuperCalc to use it without step-by-step instructions. On the SuperCalc diskette are three "sample" files with which you may wish to experiment. Make changes and see their effects on the worksheet.

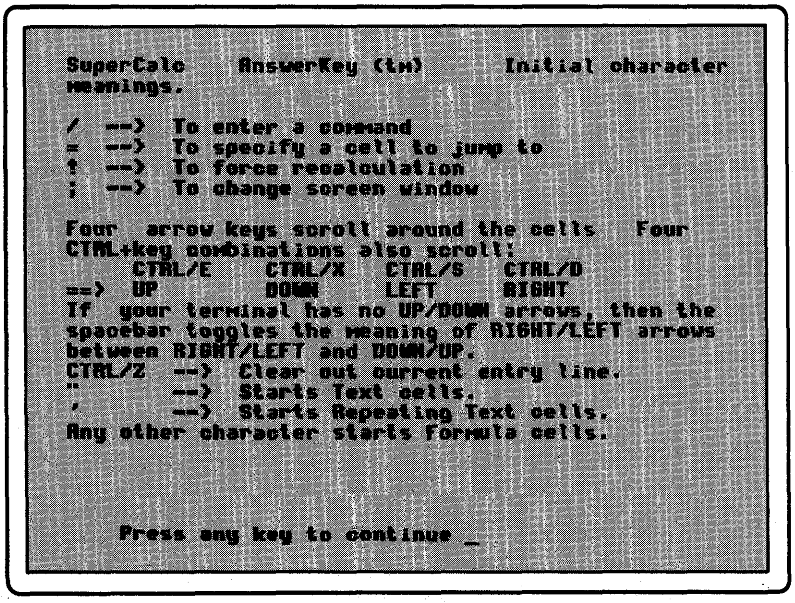
The samples deal with different subjects. One is a balance-sheet projection, one calculates the break-even point for a project, and the last does engineering calculations to specify requirements for an "air curtain." (An air curtain can separate two locations of different temperatures, such as separating a walk-in freezer from the rest of a room. Since you can walk right through the air curtain, you don't have to worry about anyone leaving the door open.)

Again, experiment with all the sample worksheets, even if the subject matter is out of your area of interest. The techniques used in the samples are general, and you can transfer them easily to your own work. In fact, you may be able to use one or more of the sample worksheets as a model for your own.

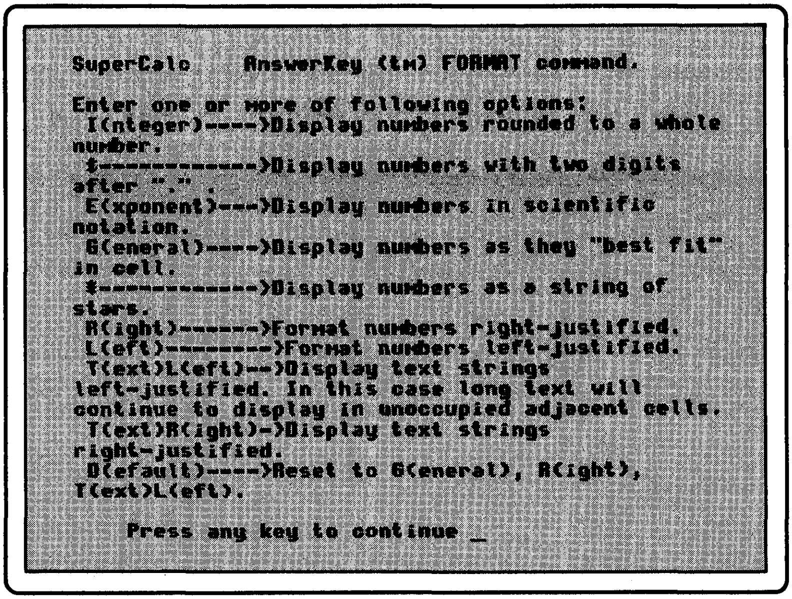
Before leaving SuperCalc, we want to remind you that SuperCalc, like WordStar, has its own built-in HELP function. Anytime you find yourself unsure of what to do next or what your options are, press the question mark (?). The HELP messages SuperCalc presents are useful reminders of the information we've presented in this chapter and should be your first source of information. So that you're familiar with what SuperCalc's HELP messages look like, we've reprinted several of them on the next two pages. And remember, if you're really stuck or need detailed information, there's always the Reference Guide. The Reference Guide contains all of the SuperCalc commands and error messages.



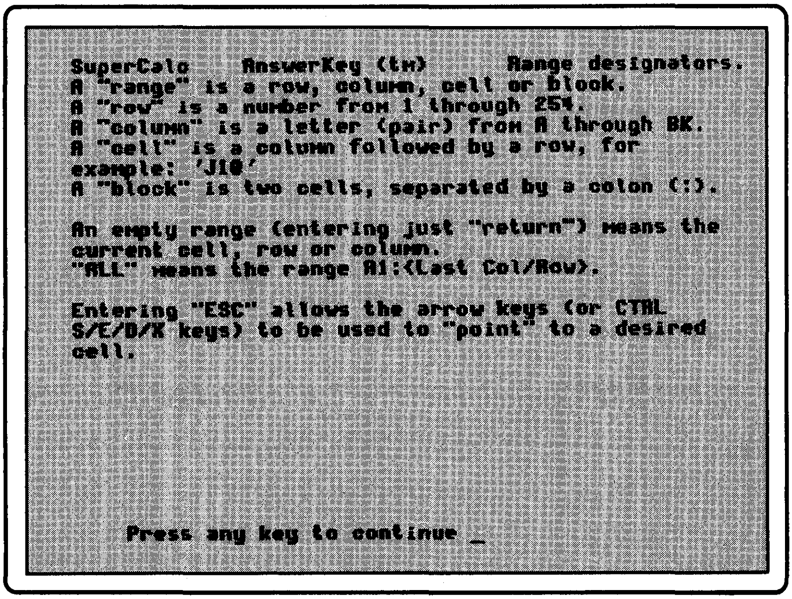
```
SuperCalc    AnswerKey (tm)  slash commands :
B(lank)-----> Removes contents of cells.
C(opy)------> Copies contents of cells.
D(elete)----> Deletes entire row or column.
E(dit)------> Allows editing the contents of a
cell.
F(ormat)----> Change display format of cells, rows,
or entire worksheet.
G(lobal)----> Change global display or calculation
options.
I(nsert)----> Create new row or column.
L(oad)------> Read worksheet (or portion) from
disk.
M(ove)------> Swap rows or columns.
O(utput)----> Display contents or values of cells
on printer, console or disk.
P(rotect)---> Prevent future alteration of cells.
Q(uit)------> Exit SuperCalc.
R(eplicate)-> Reproduce partial rows or columns.
S(ave)------> Write worksheet to disk.
T(itle)-----> Lock first rows or columns against
scrolling.
U(nprotect)-> Allow alteration of protected cells.
W(indow)----> Split or unsplit the screen display.
X(eXecute)--> Accept commands and data from a file.
Z(ap)-------> Clear worksheet and all settings.
```



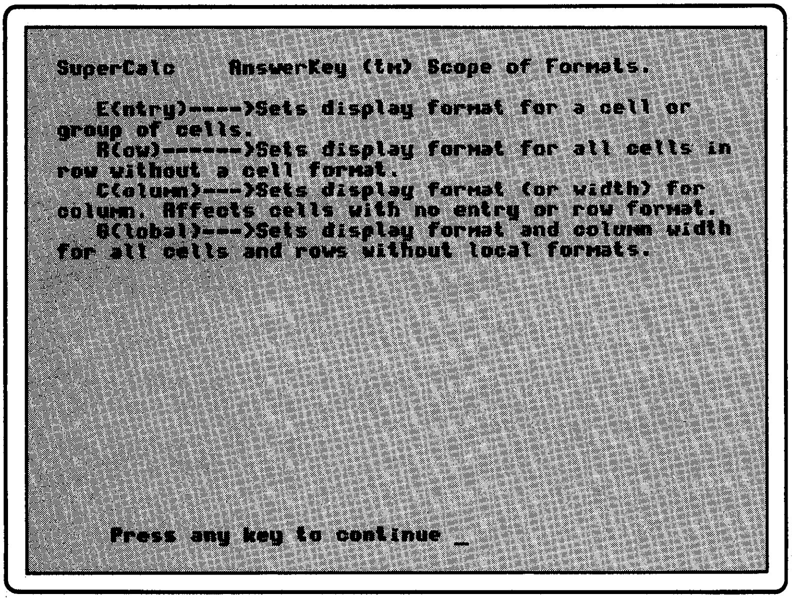
```
SuperCalc    AnswerKey (tm)      Initial character
meanings.

/  -->  To enter a command
=  -->  To specify a cell to jump to
!  -->  To force recalculation
;  -->  To change screen window

Four  arrow keys scroll around the cells   Four
CTRL+key combinations also scroll:
     CTRL/E    CTRL/X   CTRL/S   CTRL/D
==>  UP        DOWN     LEFT     RIGHT
If  your terminal has no UP/DOWN arrows, then the
spacebar toggles the meaning of RIGHT/LEFT arrows
between RIGHT/LEFT and DOWN/UP.
CTRL/Z  -->  Clear out current entry line.
"       -->  Starts Text cells.
'       -->  Starts Repeating Text cells.
Any other character starts Formula cells.



    Press any key to continue _
```



```
SuperCalc     AnswerKey (tm) FORMAT command.

Enter one or more of following options:
 I(nteger)---->Display numbers rounded to a whole
number.
 $------------>Display numbers with two digits
after "." .
 E(xponent)--->Display numbers in scientific
notation.
 G(eneral)---->Display numbers as they "best fit"
in cell.
 *------------>Display numbers as a string of
stars.
 R(ight)------>Format numbers right-justified.
 L(eft)-------->Format numbers left-justified.
 T(ext)L(eft)-->Display text strings
left-justified. In this case long text will
continue to display in unoccupied adjacent cells.
 T(ext)R(ight)->Display text strings
right-justified.
 D(efault)---->Reset to G(eneral), R(ight),
T(ext)L(eft).

    Press any key to continue _
```



```
SuperCalc     AnswerKey (tm)        Range designators.
A "range" is a row, column, cell or block.
A "row" is a number from 1 through 254.
A "column" is a letter (pair) from A through BK.
A "cell" is a column followed by a row, for
example: 'J10'
A "block" is two cells, separated by a colon (:).

An empty range (entering just "return") means the
current cell, row or column.
"ALL" means the range A1:<Last Col/Row>.

Entering "ESC" allows the arrow keys (or CTRL
S/E/D/X keys) to be used to "point" to a desired
cell.




    Press any key to continue _
```



```
SuperCalc    AnswerKey (tm) Scope of Formats.

   E(ntry)---->Sets display format for a cell or
group of cells.
   R(ow)------>Sets display format for all cells in
row without a cell format.
   C(olumn)--->Sets display format (or width) for
column. Affects cells with no entry or row format.
   G(lobal)--->Sets display format and column width
for all cells and rows without local formats.





    Press any key to continue _
```

CHAPTER 6

CBASIC

Introduction

This chapter provides you with information about the CBASIC software that accompanies your Osborne 1 computer. If you've bought programs that require the use of CBASIC, you'll want to read "What CBASIC Is," lessons 2 and 3; if you know how to program, you'll want to read all of this chapter. Otherwise, skip it for now.

What CBASIC Is

CBASIC is a version of the BASIC program language, which has been available for almost the entire history of microcomputers. The version of CBASIC you received is the current one, CBASIC2. Two earlier versions of the language were EBASIC and CBASIC.

CBASIC2 is a "pseudo-compiler." This means you create your program by using an editor (like WordStar) and then run your program through a portion of CBASIC, **CBAS2**, which translates your BASIC statements into a more compressed form. Once this "compiled code" is created, another portion of the CBASIC "interprets" these compressed instructions.

You need to understand two terms to use CBASIC:

"Source code"—your original BASIC instructions created using an editor, such as WordStar.
"Object code"—the compacted instructions CBASIC creates from your source code.

Lesson 1: Creating CBASIC Programs

In the Reference Guide, you'll find a full list of the statements and functions that CBASIC understands, complete with a full discussion of what each is for and the exact syntax in which to use each.

If you're familiar with another version of BASIC, you'll probably find that CBASIC is similar. The most notable differences between CBASIC and other BASICs are the additional statements and functions CBASIC provides for use in CP/M. For further elaboration on the commands, you might want to read *The CBASIC User Guide*, by Eubanks, McNiff and Osborne, published by Osborne/McGraw-Hill.

CBASIC source code is written as a sequence of CBASIC statements and functions. The best method to create your source program is to use the **N** (non-document file) command in WordStar. You may give your file any legal name, but it must be the **.BAS** type. For example, a source program could have the file name:

PROGRAM.BAS

You should note a few things in creating your CBASIC source code. Since we're assuming you already know something about programming, we'll just briefly summarize these unique aspects of CBASIC:

- The CBASIC compiler ignores anything on a line to the right of a [\] character. The [\] character itself is an instruction to CBASIC to consider that the information on the following line should be part of this previous line. With the backslash character, you can create multiline statements. **DATA**, **DIM** and **END** statements must be on separate lines by themselves, however.
- CBASIC programs do not require line numbers except where they specifically reference a line, as in a **GOTO NUMBER** statement. Line numbers may be any valid number, and you can express them in exponential notation or with decimal points, although CBASIC treats the following numbers as different line numbers:

```
100
100.0
1.0E02
```

- CBASIC variable names can have as many as 31 characters. If a variable name ends in a percent sign (%), CBASIC considers it an integer-only variable. Ending a variable name with a dollar sign ($) tells CBASIC that the name references a string variable.

- CBASIC has statements and functions not usually found in BASIC, or that you may not have encountered before. Most notably these are:

statements	**functions**
CALL	COMMAND$
CONSOLE	CONCHAR%
INITIALIZE	CONSTAT%
LPRINTER	FLOAT
PRINT USING	MATCH
SAVEMEM	SADD
WHILE—WEND	UCASE$

  Before attempting any significant CBASIC programming, you might want to look over the definitions for these statements and functions in the Reference Guide.

- Six compiler-directive statements in CBASIC expand your ability to create complex programs. These directives are also detailed in the Reference Guide and deserve your attention if you are doing complex CBASIC programming.

Building on your knowledge about how to program with the next chapter will help to supplement this brief introduction. Even with the complete Reference Guide, this manual will probably not suffice to teach you to use CBASIC to create programs.

We'll recapitulate what you have to know to program in CBASIC:

- You have to create CBASIC source code using an editor.

- CBASIC differs from other BASICs mostly in the additional statements and functions it provides.

- CBASIC has a few unique idiosyncrasies regarding line numbering, multiple-line statements, and variable names.
- CBASIC source code must reside in a **.BAS**-type file.

Lesson 2: Compiling: Getting CBASIC Programs Ready

To novice computer users who purchased a program that requires the use of CBASIC: the program you received is on a diskette containing instructions that constitute the program, but not the method by which to interpret them. CBASIC will be the interpreter for these programs.

Your CBASIC-based program may be in source-code format or in object-code format. The difference is critical. If you have source code, you must first "compile" the programs into the compacted form that CBASIC can interpret. If you have object code, the compacting is already done for you; the program is ready to use.

With source code, you can change the instructions should the need arise. Not so with object code. In fact, you can't even "list" these instructions to see what they are.

Let's assume for a moment that you have source code. You perform the actual "compilation," or compacting, by performing the following sequence:

1. Press the RESET button on the front of the Osborne 1 computer. Place your CBASIC/MBASIC diskette in the left-hand drive and press RETURN.
2. The Osborne logo will appear, and Microsoft BASIC will load. Once Microsoft BASIC identifies itself by displaying its name on the screen followed by an Ok, issue:

   S Y S T E M RETURN

You should see the CP/M prompt A >.

3. Place the diskette containing your CBASIC program in the right-hand drive. Type:

   [C] [B] [A] [S] [2] [SPACE BAR] [B] [:]

   [f] [i] [l] [e] [n] [a] [m] [e] [RETURN]

   Filename is the name of the program to compile. Do not include the file type when you define a file's name. CBASIC will load its compiler portion into memory and begin converting the program in drive B. If you don't know the name of the program to be compiled, first type [D] [I] [R] [SPACE BAR] [B] [:] [*] [.] [B] [A] [S] [RETURN] to see the names of the CBASIC programs on drive B.

NOTE

The instructions that accompany a CBASIC program coming from a software firm might tell you to type some special instructions —called "compiler directives"—after the file name. Be sure to do so or you may have an improperly functioning program.

4. You'll see a report on the screen at the end of compilation and, if it informs you that you made no compilation errors, your program is now ready to use. Make sure you compile all the **.BAS**-type files on the B drive —many complex programs actually consist of a series of "interlocking" short programs.

You should note two things in the above sequence. First, CBASIC displays the code it is compiling as it does so. CBASIC reports errors as it detects them, and summarizes them at the end of the process. If you want a "listing" of the program as it is compiled and have a printer attached to your Osborne 1, substitute:

CBAS2 B:filename $FRETURN

for the instruction listed in step 3, above. ($F is one of those special compiler directives we mentioned.)

A special program called **XREF** is also on your CBASIC diskette. With this program you can create a list of all the variables the program uses and where it uses them. This ability is often useful in finding and eradicating any errors in a program. You invoke **XREF** like this:

[X] [R] [E] [F] [SPACE BAR] [filename]

[SPACE BAR] [B] [:] [RETURN]

XREF will create a new file with the same name, but its type is **.XRF**; it contains this special listing.

Lesson 3: Using a CBASIC Program

To use your CBASIC programs, we advise you to create a special diskette, on which CBASIC and your programs reside, that you can load into the system just as you load your CP/M System, WordStar and SuperCalc diskettes. Let's learn how to do so.

NOTE

You might want to refer back to this section after you've read the Microsoft BASIC chapter, as the process is valid for MBASIC as well as CBASIC.

1. Press the RESET button on the front of your Osborne 1, put your system master diskette in drive A, and press RETURN.
2. The Osborne logo will appear, and the **HELP** program will load. Press the ESC key to start CP/M.
3. Place a fresh, unused diskette into drive B.
4. Format the diskette in drive B using the **COPY** program as you learned earlier in this manual. If you wish to create several CBASIC program diskettes, go ahead and format several diskettes.
5. Return to CP/M and type the following:

   S Y S G E N RETURN

   A special program that copies CP/M "systems" will load into memory and identify itself. In response to its question about "source diskette," respond with an A. After a few moments, **SYSGEN** will ask you for a destination diskette; specify B, as that's where your CBASIC diskette-to-be resides, then RETURN. **SYSGEN** will access the B drive for a moment then ask you again about a destination. If you have more diskettes with which to perform this procedure, place them, one at a time, in drive B and continue to press the B key in response to **SYSGEN**'s prompting. When you're done, press RETURN. You've just written the instructions that constitute CP/M onto your new diskettes—a necessary action because CBASIC requires CP/M in order to operate.
6. Type P I P RETURN. You should see an asterisk on your screen after a few moments. Remove your CP/M diskette from the left-hand drive and replace it with your CBASIC/MBASIC diskette. Type:

   ```
   B:=A:CRUN2.COM RETURN
   ```

(For Microsoft BASIC, substitute **MBASIC** for **CRUN2** in the above example.)

Both disk drives will be in use alternately for a few moments; then the asterisk will return. You've just copied the CBASIC interpreter onto your new diskette. You'll have to repeat this step for each diskette you create, but remember that to do so you have to tell CP/M that you've changed diskettes by typing ^[C]. This, unfortunately, causes **PIP** to stop running and return you to CP/M. A less-problematic alternative is to create one diskette and then use the **COPY** program to duplicate it.

7. You should still see the asterisk on your screen. Again place the CP/M system diskette in the left-hand drive. Enter:

```
B:=A:PIP.COM [RETURN]
```

then when the asterisk returns, enter:

```
B:=A:XDIR.COM [RETURN]
```

then when the asterisk returns, enter:

```
B:=A:COPY.COM [RETURN]
```

What you are doing is making copies of a few often used programs that will be convenient to have on your CBASIC program diskette.

8. You still have the asterisk prompt on your screen. Now transfer the CBASIC programs you compiled earlier. Place the diskette with the compiled programs on them into drive A and type:

```
B:=A:*.INT [RETURN]
```

(For Microsoft BASIC, substitute **.BAS** for the **.INT** in the above instruction.)

This time the copying process will inform you as each program is copied.

9. Put the system diskette back into the left-hand drive and type ^C. CP/M should restart.
10. Your last step is to use the SETUP program, described in chapter 2, to configure the necessary printer/special devices you intend to use with your CBASIC programs. Make sure you save these changes on the right diskette, the one in drive B.

One thing we didn't mention that might occur during one of the above steps is that you'll get a disk-error message—such as BDOS ERR ON A:—or a DISK FULL message. With the single-density disk capacity on the standard Osborne 1, you may have to omit one or more of the programs specified above when you copy files. If you run into the **DISK FULL** message often, maybe you should consider purchasing the double-density disk option.

One quick last word: the Osborne version of CP/M expects to find a program named **AUTOST.COM** on each diskette used to start the machine. Chapter 8 lists this program and makes suggestions on how to modify it.

CHAPTER 7

Microsoft BASIC

In this chapter, you'll learn about Microsoft BASIC. You'll learn how to run programs that use Microsoft BASIC, and you'll find an introduction to the commands that make up Microsoft BASIC. This chapter won't teach you everything you need to know about programming in BASIC, but it will get you started. If you are serious about learning how to program, we suggest you browse through the book rack of your local computer store and select a book on BASIC programming with examples specific to your area of interest—such as business programming or scientific programming.

Starting Microsoft BASIC

In CP/M, starting Microsoft BASIC can be simple or complex, depending on the use you intend for it. Since you're just beginning, we'll show you the general manner in which to run Microsoft BASIC.

To start MBASIC, follow the sequence below:

1. Press the RESET button on the front of your Osborne 1 computer.
2. Insert the diskette that contains Microsoft BASIC and press RETURN as the instructions on the monitor tell you.
3. The Osborne logo will appear along with the words

   Loading Microsoft BASIC.

   After a few more moments you'll see a multiple-line message, which identifies Microsoft BASIC, and the letters Ok.

Use the above sequence if you just want to load BASIC to play around with it or to start working on a new program. If you already have a partially or completely finished program, you'll want to load it into your computer's memory along with Microsoft BASIC. You could follow the above sequence and then type:

4. L O A D "*Program name*" RETURN
   or
   R U N "*Program name*" RETURN

depending on whether you wanted Microsoft BASIC to bring the program into memory for examination/modification or to begin running it.

You may encounter a few errors if you attempt to load and/or run a program. Some of the more common ones are:

SYNTAX ERROR IN XX

This message means that Microsoft BASIC tried to run your program but came across an instruction it did not understand in line XX. Something in your program is not entered in a way Microsoft BASIC recognizes; examine the line in question while consulting the appendix that gives syntax examples for each statement.

STATEMENT NOT FOUND IN XX

This message indicates that Microsoft BASIC tried to run your program but could not find a line number the program referenced. You have a flaw in program control and should check your program while rereading the section on program control.

DIRECT STATEMENT IN FILE

You receive this message if Microsoft BASIC finds a statement that is not preceded by a line number when it tries to load your program. This error sometimes occurs if you create a program using the WordStar editor and accidentally forget to include a line number or leave extraneous material in the program; try reediting the program.

FILE NOT FOUND

If you get this message, it means that Microsoft BASIC tried to load your program but couldn't find it. Check that you have the right diskette and that you specified the right program name.

Before we learn about the "building blocks" of the BASIC computer language, we need to ascertain that we're talking about the same thing when we say that you're learning how to "program" your computer.

Before you can build a computer program in BASIC, you must know what you are attempting to create.

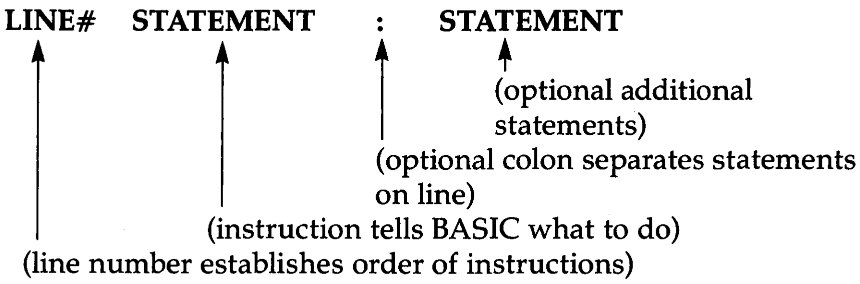
A **computer** is a group of components that work together to perform tasks. A **program** is the instructions that tell the computer how to perform the task. To be more precise, let's define a program as:

> *the complete set of instructions necessary for a computer to fulfill a predefined function (or functions)*

Your goal in this chapter is to learn about the various instructions that comprise the BASIC language—specifically, Microsoft BASIC—then to learn how to group these instructions to perform a complete function or task. More technical information is packed into this chapter than any of the previous ones, so we'll forewarn you that you'll probably have to put more effort into its lessons. Furthermore, since so much detailed information is presented here, you will have to decide which examples to try. Don't lose sight of your ultimate goal: telling the computer how to perform a task or function.

Lesson 1: Entering Programs

To understand how to enter a program using Microsoft BASIC, you must first understand a little about what makes up a program. In Microsoft BASIC, the basic component consists of what are known as "statements."



LINE# STATEMENT : STATEMENT

(optional additional statements)

(optional colon separates statements on line)

(instruction tells BASIC what to do)

(line number establishes order of instructions)

A statement is one complete instruction to BASIC to perform some action. The action can be assigning a value to a specific variable name, or something more complex such as formatting output according to a "mask" you design. When you press the carriage return, Microsoft BASIC stores each line for later use. A program consists of statements strung together in a particular order of execution.

Generally, each individual statement is on a line of its own. Each line must have a number associated with it. When a BASIC program "runs," lines execute in numeric order, unless you specify otherwise. So BASIC will perform instructions on a line numbered **10** before it performs the instructions on a line numbered **20**. For ease in changing a program after parts of it are written, you need not use consecutive numbers when you assign line numbers. Thus, the following example is a valid sequence of BASIC statements:

10 STATEMENT (executes first)
30 STATEMENT (executes second)
32 STATEMENT (executes third)
33 STATEMENT (executes fourth)
199 STATEMENT (executes fifth)

Any number between 0 and 65529, inclusive, can refer to a specific line in Microsoft BASIC. You must use actual digits to indicate the line number. Microsoft BASIC doesn't recognize typing **one** as meaning line number **one**. Note that a space must follow the line number to separate it from the instructions that follow.

It is also possible to have more than one instruction on a line. Each separate statement in such a case is separated by a colon, as in the example below.

LINE# STATEMENT1 : STATEMENT2 : STATEMENT3

In this example, when the line executes: **STATEMENT1** would execute first, followed by **STATEMENT2** and **STATEMENT3**, in that order. Some exceptions to this rule occur, and we'll state them in the discussion of commands that differ from the norm.

You can have any number of statements on a single line—each set off by a colon—up to a maximum of 255 characters. On the Osborne 1, 255 characters are almost two complete display lines (remember that you only see 52 characters of a 128-character line at a time). The line number counts as part of your 255 characters.

In addition to the "program" mode described above, there is an "immediate" mode of Microsoft BASIC. You can issue most Microsoft BASIC commands directly after you receive the Ok prompt. You can even issue multiple commands by setting off each with a colon, as described above. Your instructions will execute as soon as you press the **RETURN** key to indicate that you have entered a complete line. The 255-character rule also applies to the immediate mode, by the way.

You have available two ways to make Microsoft BASIC perform functions for you: 1) **immediate mode**—instructions execute after you enter a line of valid BASIC commands terminated with a **RETURN**, or 2) **program mode**—instructions will execute in the order of their associated line numbers when you later issue the **RUN** command.

For ease in entering programs, Microsoft BASIC has a built-in command that automatically generates line numbers for you as you're programming; this command is called **AUTO**. You issue it in the immediate mode, thus:

AUTO x,y RETURN

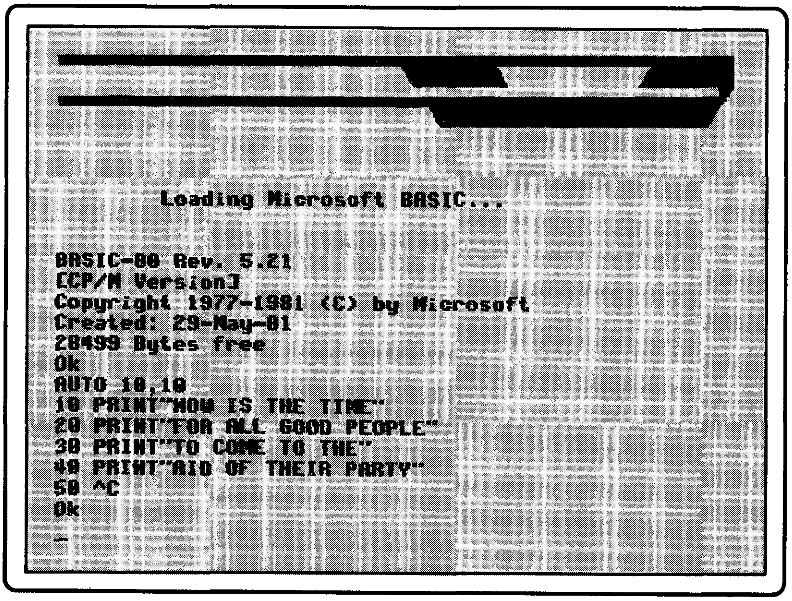
where **x** is the line number with which you wish Microsoft to begin your entry sequence, and **y** is the increment to use for successive line numbers. Typing:

A U T O SPACE BAR 1 0 , 1 0 RETURN

tells Microsoft BASIC to begin with line number 10 and to increment subsequent line numbers by **10** (i.e., the second line number will be **20**, the third, **30**, and so on).

When you use the **AUTO** command, Microsoft BASIC displays the current line number and waits for you to enter the instructions for that line. When you complete the line by pressing **RETURN**, BASIC will provide the next line number automatically and wait for you to enter the instructions for that line. This process continues until you issue a control-C as the first character on a line, at which time Microsoft BASIC drops out of the auto-line-numbering mode and returns to the immediate mode. Since you did not enter any valid instructions on the last line number displayed, your program won't include it.

To help you understand the process described above, here's a complete example of **AUTO** being used to program:



```
        Loading Microsoft BASIC...

BASIC-80 Rev. 5.21
[CP/M Version]
Copyright 1977-1981 (C) by Microsoft
Created: 29-May-81
28499 Bytes free
Ok
AUTO 10,10
10 PRINT"NOW IS THE TIME"
20 PRINT"FOR ALL GOOD PEOPLE"
30 PRINT"TO COME TO THE"
40 PRINT"AID OF THEIR PARTY"
50 ^C
Ok
_
```

The program you entered would look this:

```
10 PRINT"NOW IS THE TIME"
20 PRINT"FOR ALL GOOD PEOPLE"
30 PRINT"TO COME TO THE"
40 PRINT"AID OF THEIR PARTY."
```

Line 50 does not exist; you didn't place an instruction on that line—the control-C told BASIC you wanted to quit.

Before we leave the **AUTO** command, you need to learn that if you omit either of the numbers the command expects, Microsoft BASIC will assume you want the default value, 10, meaning that:

```
AUTO 10 is the same as AUTO 10,10
AUTO    is the same as AUTO 10,10
AUTO ,10 is the same as AUTO 10,10
```

Here the topic of how to number lines when you're programming arises. The default values that Microsoft BASIC assumes —line numbers starting at **10** and progressing in intervals of **10** —are a good starting point because you can create any line numbers you wish, and the program will execute in the order of those numbers. What happens if you enter the following program?

```
1 PRINT"NOW IS THE TIME"
2 PRINT"TO COME TO THE"
3 PRINT"AID OF THEIR PARTY."
```

You've accidentally left **FOR ALL GOOD PEOPLE** out of the quote. Since you've used line-number intervals of one, in order to tell BASIC to print the missing information you'll have to create a line number **1.5**. Since Microsoft BASIC only recognizes whole numbers to identify lines, you'll now have to reenter lines 2 and 3 in the above example and assign them higher line numbers to replace the missing part of the quote.

> You've just learned the first of what we might call **Murphy's Laws of BASIC Programming**: If you need to insert two lines between existing ones, you'll find you have room for only one.

Wise programmers choose their line numbers carefully. You should follow these rules for choosing line numbers:

1. Each major section of a program starts on a line number that is divisible by 500.
2. Each major "subroutine" within a program starts on a line number that is divisible by 500 and greater than 10000.
3. Otherwise, use line numbers with intervals of 10.

Follow the above rules and you can add extra lines, sections, or subroutines at a later date without having to renumber any existing line.

While we're on the subject of line numbers, you might wonder how you get rid of an existing line. You have two choices:

1. Type the line number you wish to eliminate, followed by a carriage return (i.e., an empty line).
2. Use the **DELETE** command described in a later section of this manual.

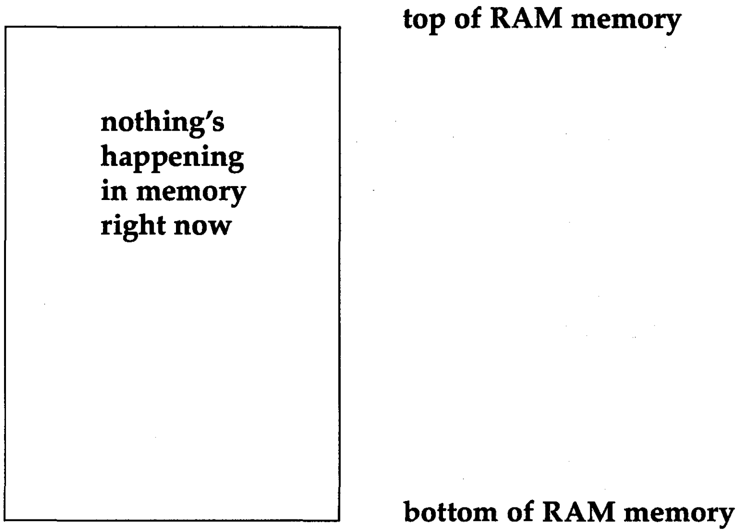
Okay, you should be all "lined up," ready to learn about variables and begin programming.

Lesson 2: Introduction to Variables

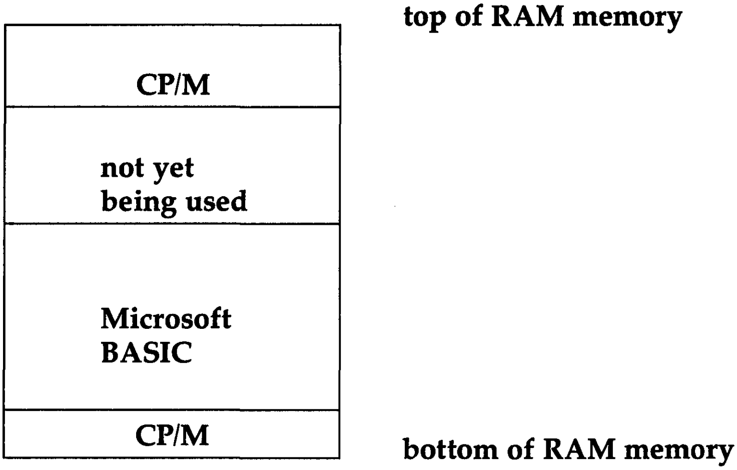
Your computer has a lot of "memory," but how do you make use of that storage capability.

Actually, when you load CP/M and Microsoft BASIC into your computer, you already begin using some of the RAM (random-access memory) your system contains. Before we show you what variables are and how BASIC maintains them, we want to make sure you know what's happening inside your computer.

If you could have looked inside the Osborne 1's memory when you first turned it ON, you would have found that nothing had yet been stored there:



If you could have looked inside your memory after you loaded and executed CP/M and Microsoft BASIC, you would have found this:



As you can see, CP/M takes up a little memory at the bottom (256 bytes) and some more room at the top—usually between 4K and 8K, depending on the version of CP/M. Microsoft BASIC, which has loaded toward the bottom of memory, uses approximately 24K of memory area. The portion of memory labeled **not yet being used** is where your program and variables (information) will be stored.

Generally, you don't need to know exactly where each piece of your program is stored. We will point out, however, that Microsoft BASIC uses the space immediately following the program and works its way up as it needs to use more memory.

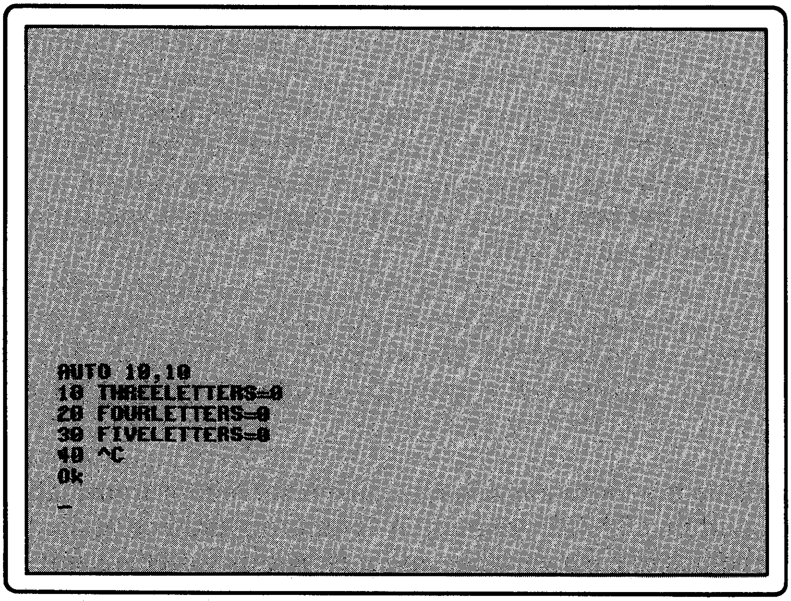
A variable is a temporary storage area reserved for information used within a BASIC program. As you program, you assign arbitrary names to each variable, and it is by this variable name that you can tell BASIC what to do with the information.

We'll show the concept of variables with an example. Suppose that you were trying to count the number of three-, four-, and five-letter words in this paragraph. If you were counting in your head, without the benefit of a pen and paper to make "tally marks," you'd probably say the following to yourself:

> "That's no three letter, one four letter, no five letter; that's no three letter, two four letter, no five letter; that's one three letter, two four letter, no five letter.

Consider this example carefully. You're using placement and identification to keep track of each changing piece of information. When you're programming, BASIC keeps track of the placement, while you use the identification to tell your program when to update a piece of information.

Let's look at how you do this in Microsoft BASIC:



```
AUTO 10,10
10 THREELETTERS=0
20 FOURLETTERS=0
30 FIVELETTERS=0
40 ^C
Ok
_
```

To start, each variable has a name and is set equal to zero. The first time it encounters a variable name, Microsoft BASIC assumes that the value is zero, unless you inform it otherwise, but it is good programming practice to explicitly declare a value for every variable at the beginning of your program. That way you'll be sure that the values used are the ones you intended.

Note that our choice of names is purely arbitrary. Within certain constraints, Microsoft BASIC allows you to assign names that are any combination of letters up to 40 characters long. Actually, you could have longer names, but BASIC would only use the first 40 characters.

At this point, you only need to know a few rules about choosing names:

1. All names must begin with a letter.
2. Make your names between 1 and 40 characters in length.
3. Do not start a name with the letters **FN**.
4. Do not use special characters (@, #, $, %, ^, &, etc.); use only uppercase letters and numbers.
4. Variable names cannot be any word already in use as an MBASIC command.

Now, back to our example.

We introduced you to the most used BASIC command without your realizing it. Remember the statement **10 THREELETTERS = 0**? Did you wonder what the **= 0** part of it meant?

All three statements in our example are called **LET** statements. The general form of a **LET** statement is:

LINE# LET X = Y

As in our example, you need not type **LET**; the equal sign in the statement implies it. The above means: 1) calculate or retrieve variable **Y**; 2) move its value into the storage location assigned to **X**.

Acquire the habit of saying to yourself, "New equals old," because we're going to confuse the picture a bit with a new statement for our example:

```
50 THREELETTERS = THREELETTERS + 1
```

In the above statement, using the first rule we just stated, BASIC will retrieve the value of **THREELETTERS**, add one, and then store the result in **THREELETTERS** (following the second rule).

LET statements can be simple, like the one above, or they can be quite complex, like the one below:

```
1010 RESULT = ((XPOS*YPOS)-(42/ZPOS))/2
```

The above calculation uses algebraic notation. For the most part, Microsoft BASIC follows algebraic notation conventions exactly. Below are some of the rules of evaluating algebraic expressions:

1. Do all multiplication and division before addition and subtraction operations.

   $$A + B + \underset{\uparrow}{\underline{C * D}}$$

   calculated first

   We'll elaborate on "precedence" in a few paragraphs, when we introduce some other arithmetic operations.

2. Calculate the expressions in the innermost set of parentheses first.

   $$(A + (B * C + \underline{(D} \underset{\uparrow}{/} \underline{E)}))$$

   calculated first

3. Work from left to right if none of the above rules apply.

$$\underset{\substack{\uparrow \\ \text{calculated first}}}{\underline{\mathbf{A} + \mathbf{B}}} + \mathbf{C} + \mathbf{D} + \mathbf{E}$$

Using the above rules, if you were calculating the expression

(24 * 5) / 2 – 5 + 10

you'd perform the following steps:

1. multiply 24 times 5	(equals 120)
2. divide by 2	(equals 60)
3. subtract 5	(equals 55)
4. add 10	(equals 65)

In Microsoft BASIC, the statement that represents the above calculation looks like this:

```
200 RESULT = (24*5)/2−5+10
                  or
200 RESULT = 24*5/2−5+10
```

While we're concentrating on things mathematical, let's make sure you know the basic arithmetic functions a **LET** statement can use, in order of their precedence during calculation:

^ represents exponentiation: **2 ^ 2** means "two raised to the power of two"; exponents are calculated before multiplication and division are performed, but otherwise follow the rules stated above.

* represents multiplication: **2 * 2** means "two times two."

/ represents division: **2 / 2** signifies "two divided by two."

\ represents integer division: **2 \ 2** means "the integer result of two divided by two," or that any remainder is forgotten after the division is completed. Integer division takes precedence over addition and subtraction, but not over multiplication and division.

MOD represents modulus division: **2 MOD 2** denotes "the integer remainder left after two is divided by two"; modulus division is performed after integer division, but before addition and subtraction.

+ represents addition: **2 + 2** means "two plus two."

− represents subtraction: **2 − 2** signifies "two minus two."

One other "operator" you should know about is called "negation." Negation looks like subtraction in that it uses a hyphen (-) to indicate its presence. You can tell whether the ⊟ implies subtraction or negation by the context in which it is used. Look at the example below.

X–Y	subtraction
X * (-Y)	negation (parentheses must separate two consecutive operators . . . one hint that negation is implied)

Before you leave **LET** statements, note just a few more things about them. First, you can freely intermix variables with explicitly stated values, as in this example:

```
100 RESULT = 45 * VARIABLE2
```

You must, however, place a variable on the left side of the equal sign. In fact, only one element can go on the left side of the equations; this is another requirement of the LET statement: you can only assign values to one variable at a time. Thus, none of the following are valid statements:

10 45 = RESULT * SECONDRESULT ← doesn't have a variable to the left of the equal sign

10 RESULT + SECONDRESULT = 45 ← you've assigned two variables on the left of the equal sign

Do You Know the Variable Type?

So far we've used only examples in which numbers or values are stored in variables. Actually, you can specify four types of variables in Microsoft BASIC.

1. **INTEGER** variables can store only values between −32768 and 32767. Integer numbers cannot have embedded decimal points.
2. **SINGLE-PRECISION** variables can store any positive or negative real numbers, including those with fractional values indicated by a decimal point (i.e., 3.1456). Single-precision variables, so-called because they store only the seven most significant numbers and print with only the six most significant. If you overstep these bounds, the result will be incorrect rounding or truncation of information, depending on the operation you are performing.

   45 / 7 = 6.4285714 with 9-digit precision,

   but the result is only **6.42857** with 6-digit precision

The rounding in the above example is correct—at least if you're concerned only with the first six digits of accuracy—but what would happen if the result stored were 6.4285771? Precision errors have a tendency to mushroom. Although no error may exist after one operation, repeated operations can exaggerate any imprecisions.

3. **DOUBLE-PRECISION** variables not only store any positive or negative real number, but they can also do so with 16 digits of accuracy.
4. **STRING** variables are a special type. Even though they can store certain types of values, they almost always store "characters" of information. You assign information to be stored in a string variable by using quotes around the characters you wish to store:

   ```
   100 STRINGVARIABLE$ = "ABCDEF"
   ```

   The above example places **ABCDEF** in storage with the name STRINGVARIABLE$ assigned to it.

   You can do more than make assignments with string variables. In a later lesson, we'll show you how to add, subtract, compare, and parse, among other functions, with string variables.

You may wonder how BASIC can know which type of variable you're referencing when you use a name. Actually, if you do not explicitly state how you want BASIC to store information, it will assume that your variables are single-precision.

To tell Microsoft BASIC the type of variable you desire, append a single character to the end of the variable name. The character informs BASIC of your intentions.

%	integer variable
!	single-precision variable
#	double-precision variable
$	string variable

METOO% **METWO%** **METO%**	all integer variables
YOUANDME	Note: no symbol means single-precision, also
YOUANDME! **UANDME!**	all single-precision variables
HE# **SHE#** **IT#**	all double-precision variables
MONEY$** **DOLLAR$ **MULLAH$**	all string variables

Once you define a variable using an extra character you must continue to employ that character every time you use the variable in your program. **HE#** and **HE**, for example, are interpreted as different variables.

You might sometimes notice that as you are typing a program, Microsoft BASIC supplies an exclamation point or number sign that you didn't specify. For instance, typing the following statement:

```
5440 ANDHIKE = 654578
```

will appear in any program listing as:

```
5440 ANDHIKE = 654578!
```

If you had declared **ANDHIKE** a single-precision variable, the exclamation point would not have appeared after the specified number.

We don't want to bog you down in arithmetic details this early in the chapter, so we'll save supplemental details for later. Be forewarned, however, that you haven't learned everything there is to know on the subject of Microsoft BASIC's variable notation.

Print It

So far you only know how to number lines and assign values to variables. Now we'll show you a way to get information out of the computer.

Enter each line of the following program lines, pressing [RETURN] after each:

```
10 VARIABLE2 = 10
20 RESULT = 45 * VARIABLE2
```

It would be nice to learn the value of **RESULT**, and it's simple: you use a **PRINT** statement. Enter:

```
30 PRINT RESULT
```

followed by [RETURN]. If you now run the program—issue ^[C] and [R] [U] [N], then press [RETURN]—you'll see **450** appear on your screen. Try this new statement. Type:

[A] [U] [T] [O] [SPACE BAR] [1] [0]

and press [RETURN], then type:

```
VARIABLE2 = 10
RESULT = 45 * VARIABLE2
PRINT RESULT
```

Follow each line with [RETURN] and on line 40 issue ^[C] followed by [R] [U] [N]. Microsoft BASIC will display the value of **RESULT** 450 and follow that with another Ok.

PRINT statements can be as complex as **LET** statements. Instead of adding line 30 in the example, we could have made our program:

```
10 VARIABLE2 = 10
20 PRINT 45 * VARIABLE2
```

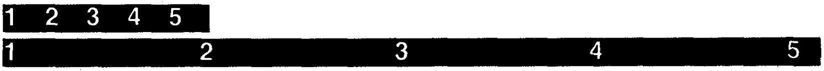
Don't rush to any conclusions about this last point, though. Sometimes it pays to shorten your program by combining **LET** and **PRINT** statements—as we have just done—sometimes it doesn't. Generally, if a value is going to be calculated only once, it is okay to calculate it on a **PRINT** statement. In the interest of clarity (does someone else have to be able to understand your programs?), however, you should avoid the combining practice.

Two other aspects of the **PRINT** statement are important. First, if you wish to send something to a printer instead of the Osborne 1 screen, substitute the word **LPRINT** (Line PRINTer) for **PRINT**. For this procedure to work, you must have a printer attached to your Osborne 1 and have used the **SETUP** program to specify what kind of printer you're using. In addition, the printer must be turned ON and ready to print; don't scoff, many so-called "problems" with printers arise because no one bothered to turn the printer ON!

Another significant aspect of **PRINT** statements is that you can use one **PRINT** command to print multiple items. Each item to print is set off by a semicolon or a comma, depending on whether you want the items to print with 14 or no spaces between them. Enter the following:

```
10 PRINT 1;2;3;4;5
20 PRINT 1,2,3,4,5
```

On line 30, type ^[C] to get in the command mode, then type [R] [U] [N], and press [RETURN]. You should see:



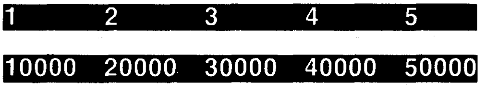
```
1 2 3 4 5
1             2             3             4             5
```

Hold on. Didn't we just say no spaces would appear between the values if you used the semicolon? True, but Microsoft BASIC is leaving a space in the printout for the "sign" (positive/negative) of the number. Since the numbers are positive, and BASIC doesn't show a + sign, you should assume from this that BASIC exhibits only negative signs unless you explicitly tell it to display the positive sign for a value. In addition, when you print numbers, each number is always followed by at least one space, even if you use the semicolon.

Let's look at the use of the comma in a **PRINT** statement a little more carefully, too. Enter the following:

```
10 PRINT 00001,00002,00003,00004,00005
20 PRINT 10000,20000,30000,40000,50000
```

On line 30, issue ^C, and type RUN and press RETURN. You'll see:



```
1             2             3             4             5

10000         20000         30000         40000         50000
```

If you count carefully, you'll find that each new item starts 14 spaces after the preceding one. Regardless of the length of the item, the numbers print at the beginning of 14-character columns.

Some final tidbits about the **PRINT** statement: for the word **PRINT**, you can substitute a question mark. This saves typing four additional characters. Also, Microsoft BASIC can understand **PRINT**, even if a line number doesn't precede it. Enter ?45*10 followed by RETURN. You should see the result just as before. This last feature is handy when you're trying to calculate a value but don't want to write a program to figure it out. Think of this feature as an expensive calculator.

Remarks

Sometimes you merely want to insert into a program a comment—something that suggests what is happening but that does not appear to BASIC as an instruction to do something. For such comment insertions, use the **REM** (remark) statement:

```
10 REM This is the start of the program
```

It is wise to use plenty of remarks in your program when you are still building it—you can take them out later if the program starts to become large or cumbersome.

You can add remarks to lines that contain instructions:

```
10 PRINT HRS;:PRINT SECS:' Tell user how much time
has expired
```

What?! We pulled three tricks on you in the above example: 1) we substituted a ' for **REM**, 2) we put three instructions on one line (two **PRINT**s and one **REM**); and 3) we made the hours (**HRS**) and seconds (**SECS**) print together because we ended the first PRINT statement with a semicolon.

The first trick is the same as replacing **PRINT** with a question mark, and Microsoft BASIC recognizes it as a valid substitution.

To perform the second trick, set off each statement with a colon. Thus, our example is the equivalent of:

```
10 PRINT HRS;
11 PRINT SECS
12 REM Tell user how much time has expired
```

The third trick should make sense to you. Since the semicolon —when used between items to be printed—inserts no spaces between the items, the semicolon will perform the same function, even if the items to print are in different statements; the same is true of the comma.

If you're concerned that BASIC is complex and you have too many things to remember, don't worry. We've already introduced about ten percent of Microsoft BASIC to you, and this fraction of the language will probably account for over half of your BASIC programming.

To round out this lesson, we'll teach you some other ways to manipulate statements. Type [N] [E] [W] [RETURN], then [A] [U] [T] [O] [SPACE BAR] [1] [0] [RETURN], and enter the following program:

```
PRINT"The Osborne 1 is a great computer."
PRINT"It only costs:"
SOFTWARE=1400
PRICE=1795
PRINT PRICE-SOFTWARE
END
```

When entered, issue ^[C] and [R] [U] [N], then press [RETURN].

Already you've encountered two new statements:

NEW erases from memory any previously existing program. Use this with caution!

END tells Microsoft BASIC that it has come to the end of a program. For reasons that will become apparent later, this should be the last statement every program you write executes.

Take a close look at what the program printed on the screen. Are you happy with the results? We aren't: there's no dollar sign in front of the **395**, and the price really should be on the line where **It only costs:** is. Is there any way to make the program function the way you want without having to type it all again?

To this question there is a reasonable answer. Type [L] [I] [S] [T] followed by [RETURN]. Microsoft BASIC should redisplay your program. Type [L] [I] [S] [T] [SPACE BAR] [1] [0] followed by

RETURN. Microsoft BASIC should display line 10 of your program. Type L I S T SPACE BAR 1 0 – 3 0 followed by RETURN. Microsoft BASIC should display lines 10–30 of your program.

NOTE

*To make the listing of your program, go to your printer instead of to the screen, issue **LLIST** instead of **LIST**. This construct is similar to the **LPRINT** statement described earlier.*

Okay, so now you can relist your program to see what Microsoft BASIC thinks is in it. How do you change it?

You can enter in new lines as necessary, so let's change the way the end of the program functions. Type the following, remembering to complete each line with a RETURN:

```
45 REALCOST = PRICE - SOFTWARE
50 PRINT REALCOST
```

List your program again to make sure that the changes are made correctly. Run your program again to ascertain that it works the same way. Does it? Good, now proceed to make some "cosmetic" changes. Type:

E D I T SPACE BAR 2 0 RETURN

You should see the line number (20) and nothing else but your cursor. Microsoft BASIC has retrieved line 20 from memory and is now waiting for you to issue one of the "editing" commands it recognizes:

SPACE: moves cursor forward one character and shows previous character.

LEFT ARROW: moves cursor backward one character but does not erase any characters.

I: tells Microsoft BASIC to begin "inserting" characters at the current cursor position. All characters to the right of the cursor will "push ahead" to make room for the new characters you enter.

ESC: ends the insertion of characters and resumes normal editing.

X: places cursor at the end of the current line and allows you to add to that line.

D: deletes the character at the current cursor position. So that you can tell deleted characters from retained ones before you execute the command, BASIC displays deleted characters in backslashes.

H: deletes all characters from the current cursor position to the end of the line.

Schar: typing **S** followed by another character tells Microsoft BASIC to go to the next occurrence of the character; the cursor positions just before the next occurrence.

Kchar: deletes all characters from the current cursor position to the next occurrence of the character specified.

C: replaces the next character in the statement with the next character you type.

RETURN or E: ends editing of the current line with all changes being used.

Q: quits editing of the current line without implementing the changes you may have made to the line.

L: lists the remainder of the line and repositions the cursor at the beginning of the line.

A: restarts the editing of a line; useful when you've made some changes you want to "take back."

Don't worry too much about memorizing the above commands. Eventually they'll be normal to you, as each single-letter command implies the editing function (**C**, Change; **D**, Delete; **L**, List). If you've learned WordStar, the actual functions performed should seem quite natural to you.

With the above commands in mind, return to that program with the pricing problem. We want to add a semicolon to the end of line 20 so that the price prints to the right of the **It only costs:** element. Type the following sequence to do so:

E D I T SPACE BAR 2 0
X
;
RETURN

Now rerun the program and make sure it works as you want it to. It should look better, but still not correct. Where's the dollar sign? Change that line again; the sequence of commands you type this time is:

E D I T SPACE BAR 2 0
S " S "
I $ ESC RETURN

You just inserted a space and a dollar sign at the end of the information to print. Run the program one more time to make sure that it works.

Now get rid of a line by using the **DELETE** statement. Type D E L E T E SPACE BAR 6 0. Now list the program; line 60 should be missing. **DELETE** works as does **LIST**, although it always requires a single line number or a series of line numbers.

The subject of line numbers evokes Murphy's Laws of Programming, one of which states that if you need to insert two lines between two others, you'll only have room for one. The people at Microsoft must have heard of Murphy, because they've provided a **RENUM** command, which renumbers lines. Type R E N U M SPACE BAR 1 0 , 1 0 , 2 0, and press RETURN. Now list your program. You should see:

```
10 PRINT"The Osborne 1 is a great computer."

30 PRINT"It only costs: $";

50 SOFTWARE  = 1400

70 PRICE  = 1795

90 REALCOST  = PRICE - SOFTWARE

110 PRINT REALCOST
```

Notice that line 45 has been renumbered with the same multiple (20) as the rest of the lines. The components of the **RENUM** command are:

RENUM #1 , #2 , #3

where #1 is the first number to use in the new sequence;
#2 is the first current line number to change;
#3 is the increment to use for numbering lines.

You can omit any of the three, but be sure to use a comma to show its place if you are omitting only one or two of the components.

Despite all the abilities of the Osborne 1, there are a few things you cannot do:

- use renumbering to rearrange the order in which statements occur, and
- create numbers greater than 65529.

You'll learn later about other statements, such as **GOTO** and **GOSUB**, that reference line numbers. If the line numbers these other statements reference exist, **RENUM** automatically changes the references.

Some Cleanup Instructions

You've now completed two of the 13 topics we want to present concerning Microsoft BASIC. So you have something to "play" with for the remaining lessons, make sure you know how to save your program for later use.

We're assuming that you have your Microsoft BASIC diskette in drive A. Type:

```
SAVE"PROGRAM1"
```

and press RETURN. Note that your program's name **(PROGRAM1)** must be enclosed in quotes. To save the program on drive B, type:

```
SAVE"B:PROGRAM"
```

and press RETURN.

Lesson 3: Control Structures

You know about variables, line manipulations, printing of information, arithmetic operations, and the **LET** statement. These attributes comprise about half of most simple programming chores in BASIC.

What you don't yet know is how to make BASIC execute only sections of a program. We'll call this "forced execution" "control structures" because you are creating a program structure that controls which statements to execute and which not to execute.

You want to retrieve that program you've been working on. Type:

L O A D " P R O G R A M 1 "

and press RETURN. Next type L I S T followed by RETURN to verify that the program loaded correctly. This exercise is more for your benefit than anything else, as your Osborne 1 usually reports any problems it encounters in loading a program file. To give yourself some space to work, renumber the program again:

R E N U M SPACE BAR 1 0 0 , 1 0 0 , 1 0

RETURN

The result of this command is that your program now starts with line number 100, and each succeeding line number is in increments of 10.

Add the following to your program:

```
10 REM This is the new start of the new program
20 GOTO 100
30 END
```

Notice the **END** statement in line 30. If Microsoft BASIC sees this statement, the program stops executing. Notice also the

GOTO 100 statement in line 20. Since Microsoft BASIC executes a program by looking at line numbers in order, it will see **GOTO 100** before it sees **END**. The **GOTO** statement forces execution to take place at the line number indicated, so Microsoft BASIC never reaches line 30. Verify this by running the program; it should still work as before.

Just for fun, type 160 GOTO 10, and press RETURN. Now run the program. The program goes around and around, printing the same message on the screen. Your program is now in an "infinite loop," so named because there is no possibility that the program will come to an ending place—it merely keeps on going back to the beginning and repeating the same thing over and over again. Depending on what you're trying to do, infinite looping can be both useful and frustrating.

Want to stop the looping? Issue ^C to tell Microsoft BASIC to interrupt what it is doing and return control back to you. You will see a message such as Break in 100, to indicate what line number Microsoft was executing when you stopped it. Now type: C O N T (for continue), and press RETURN. The message resumes where it left off. Issue ^C to stop again.

Although that exercise was interesting, all you've learned so far is to command Microsoft BASIC to do something, period: "forced execution."

Another type of execution is called "conditional": execution resumes at a location dependent upon what "condition" Microsoft BASIC encounters. Let's examine this idea more carefully.

"If I were rich, I'd buy a computer": that statement exhibits conditional execution. **If** you are rich, you'll buy a computer; if you're not rich, the implication is that you won't. Let's add this construct to our program:

```
160 END:'CANCELTHE CONTINUOUS RUNNING
20 RICH=1
```

```
30 IF RICH=1 THEN GOTO 100
40 PRINT"You can't afford a computer!"
50 PRINT"But it doesn't take much to buy an Osborne 1."
60 GOTO 110
```

Run your program now. It still does the same thing, because at line 30 Microsoft BASIC "tests" to see if **RICH=1** or not. If that statement is true, then execution proceeds at line 100; if not, execution proceeds with the next line, 40. Now change line 20 to read **RICH=0** instead. What happens when you run the program now? It executes lines 40–60, skips line 100, and executes lines 110–160.

If you don't believe what you just read about which line numbers were executed and which were not, we'll teach you a trick you can use to verify which lines BASIC executes. Issue T R O N (for trace on), and press RETURN; then type R U N, and press RETURN. You should see:

```
[10] [20] [30] [40] You can't afford a
computer! [50]
But it doesn't take much to buy an Osborne 1.
[60] [110] It only costs: $ [120] [130]
[140] [150] 395 [160]
Ok
```

Each of the bracketed numbers is a line number that executed. Although the example above clutters up the display, it allows you to verify exactly what Microsoft BASIC is doing at any point in the program. You might not need this facility now, but when your program becomes difficult to follow, **TRON** is a handy tool. To turn this feature off, type T R O F F (for trace off) followed by a RETURN.

The complexity of the **IF** test is up to you. The tests you create work about the same way as does the **LET** statement, introduced earlier. For example, the following is a valid **IF** test:

```
IF ((RICH=1) AND (POOR<>1)) THEN 100
```

Notice the use of parentheses to force BASIC to evaluate certain portions of the statement before others. Also, any of the "logical operators" that follow can be used:

<>	not equal to
=	equal to
<=	less than or equal to
=>	greater than or equal to
>	greater than
<	less than
AND	both subequations must be true
OR	one of the subequations must be true

The last two possibilities may be past your understanding at this point, but don't worry—we'll return to them at an appropriate time.

We want to show you one more trick concerning the **IF** test. Enter the following replacements for lines 20 and 30:

```
20 RICH=1
30 IF RICH THEN 100
```

We're using a lot of shorthand here. First, the test is missing! The reason this trick works is because Microsoft BASIC evaluates the entire expression between the **IF** and the **THEN**; if the expression is equal to zero, then the expression is false; if the expression is not equal to zero, the expression is true. To BASIC, a value of zero is always "false," while any other value is "true." Another trick is that we've left off the **GOTO** in line 30. In this example, the context implies **GOTO**.

One other attribute of **IF** before we move on: you can insert any valid expression between **IF** and **THEN**, and any valid BASIC statement can follow **THEN**. Thus,

```
30 IF RICH<>1 THEN END
```

is a valid statement.

A variant of the **GOTO** statement you learned earlier is the **GOSUB** statement; **GOSUB** stands for "go to subroutine and return." **GOSUB** means almost the same thing as **GOTO**, but a **GOSUB** always comes back to the statement that follows it. You might compare the **GOSUB** statement to a **GOTO** boomerang —it always comes back. Let's kill two more birds while learning one new command:

```
5 GOSUB 1000
1000 PRINT CHR$(26)
1010 RETURN
```

Now run your program. The screen should clear, then your message should appear. Line 1000 clears the screen; for the time being, never mind how. Line 5 tells BASIC to execute statements that begin with line 1000, but to remember to return when it has finished the "subroutine" beginning at line 1000. Line 1010 contains the command that tells BASIC that your subroutine is complete and it's time to return to the main program's next statement.

In other words, **GOSUB** directs BASIC to execute a new section of the program—so far, just as **GOTO** does—but when BASIC encounters the magic word **RETURN**, **GOSUB** instructs BASIC to come back and begin executing the instructions that immediately follow the original **GOSUB**. Sound confusing? It shouldn't, especially since you already know the statements to use to check what's happening: turn the trace function on, run your program, and list it.

NOTE

You won't see the first two statements that execute because the screen clears after they've been passed; sorry about that, but let's move on to another control structure: ***FOR/NEXT*** *loops.*

So far, the control structures we've introduced are simply "go from here to there (and maybe return)" types. The **FOR/NEXT** construct is an example of a structure called "looping."

A loop is a section of code that repeats. Remember the "endless loop" you created earlier in this lesson using the **GOTO** statement. That program performed instructions, then looped back to the first one and started over again. We'll do the same thing with **FOR/NEXT**, except we'll tell the computer how many times to loop.

```
10 FOR LOOP=1 TO 10
20        PRINT"The Osborne 1 is a great computer."
30        PRINT"It only costs: $";
50        SOFTWARE=1400
70        PRICE=1795
90        REALCOST = PRICE - SOFTWARE
100       PRINT REALCOST
110 NEXT LOOP
```

This new version of our program has some additional qualities. First, there is a variable named **LOOP** involved in the **FOR** statement; there could be others to the right of the equal sign, as well. Second, the program instructions ("code") that will execute each time the loop executes are contained between the **FOR** and the **NEXT** statements. Also, we have indented (using the **TAB** key) the instructions within the loop, so that the actual loop becomes more visible to the casual observer. This last point is not frivolous. In complex programs, loops often become buried in instructions, where they are difficult to find and check for proper functioning.

Several components make up the **FOR** and **NEXT** statements; we want to be sure you understand each part:

> **FOR variablename = startnumber TO endnumber**
>
> **NEXT variablename**
>
> **Variablename** is a name you give the loop;
> **startnumber** is a starting value for the loop count; and
> **endnumber** is the ending value for the loop count.

Every **FOR/NEXT** loop must have a variable name associated with it. During each iteration of the loop, the current loop number will be the value of that variable, and you can use that variable in calculations if you want. Type:

```
FOR LOOP=1 TO 10
        VALUE = 2 * LOOP
        PRINT VALUE
NEXT LOOP
```

You have to specify the starting and ending count, although another variable can set these values.

```
FOR LOOP=START TO LAST
       VALUE = MULTIPLIER * LOOP
       PRINT VALUE
NEXT LOOP
```

Make sure, however, that you initialize (set the starting values of) the variables used for the starting and ending count before you get to the loop. A loop will not execute if its ending count is lower than its starting count. A special circumstance arises when the two counts are the same: the loop executes once. Because the starting count is not greater than the ending count, execution of the loop proceeds. When execution reaches the **NEXT** statement, the loop-counter variable has equalled the end value; therefore execution stops.

You may "nest" loops within loops. Each loop must be wholly contained within another—they cannot "overlap." Also, each loop must have its own name. Try entering and executing the following program; but before you do, see if you can guess what will print at each step of execution:

```
10  FOR LOOP1=1 TO 10
20          FOR LOOP2=1 TO 10
30                FOR LOOP3=1 TO 10
40                      PRINT LOOP1,LOOP2,LOOP3
50                      FOR DELAY=1 TO 100
60                      NEXT DELAY
70                NEXT LOOP3
80          NEXT LOOP2
90  NEXT LOOP1
100 STOP
```

We sneaked a new statement into that last program example, by the way: **STOP**. **STOP** performs the same function as does typing a control-C while a program is running: it stops the program temporarily. To continue the program (if there's more, which there isn't in this example), simply issue **CONT** and press **RETURN**, and execution picks up with the statement following the **STOP** statement.

STOP is handy to use when you're trying to figure out why a program doesn't work correctly. Insert a few **STOP**s into your program, and, when execution stops, check the values of the variables in your program by using the **PRINT** command in the immediate mode (without line numbers). Then type **CONT** to resume where you left off.

Lesson 4: Getting Around on the Diskette

You already know how to save and load BASIC programs. Before continuing with our programming examples, we want to teach you a few new tricks to do with the SAVE and LOAD statements, and briefly introduce some other commands for manipulating files on a diskette.

In CP/M, to get a directory of the files on a diskette, type **DIR**, and press **RETURN**. Sometimes you use the asterisk and question mark to search for certain types of names (as in **DIR *.BAS**). In Microsoft BASIC, the **FILES** command does the same thing. The only difference you must learn between **FILES** and **DIR** is that you must specify any additional factors enclosed within quotation marks. The chart below shows equivalent commands in BASIC and CP/M.

BASIC	**CP/M**
FILES <cr>	DIR <cr>
FILES "A:" <cr>	DIR A: <cr>
FILES "B:*.BAS" <cr>	DIR B:*.BAS <cr>

<cr> stands for **RETURN**.

You can use the **FILES** command within a BASIC program.

To erase a file in CP/M, you use **ERA**; in Microsoft BASIC, use **KILL**. Despite its violent name, the command works in the same benign way as **FILES** does:

```
KILL "A:*.*" <cr>
KILL "YOREFILE.BAS" <cr>
KILL "B:*.BAS" <cr>
```

The Microsoft BASIC equivalent of the CP/M **REN** command, **NAME AS**, works a bit differently:

BASIC	CP/M
NAME "OLDFILE" AS "NEWFILE"	REN NEWFILE= OLDFILE

We told you earlier that almost universally in computing, "new = old" is true, remember? We suppose Microsoft BASIC must be the exception that proves the rule, since it does just the opposite. Fortunately, the syntax of the command suggests the order in which the file names should appear.

The next command we'll introduce, **MERGE**, is often handy when you're creating large programs.

MERGE "programname" <cr>

MERGE takes the program file you specify and "merges" it with the program in memory. Where line numbering corresponds between the two programs, the program being merged from the diskette will replace the old program. Where line numbering diverges between the two programs, the new program will simply be added to the existing program. Keep two caveats in mind: the file to be merged must have been saved as an ASCII text file (see below), and if the **MERGE** command is used within a program, execution stops and you will see the **Ok** prompt again (this is not true if you use the **MERGE** option of the **CHAIN** command; see the Reference Guide for details).

Suppose you want to return to CP/M. Simply type **SYSTEM** and press **RETURN**. Be careful with this command, however, because once you issue it, you have no way to recover any data your program may have stored in memory.

Earlier in this lesson we promised you some more information about the MBASIC **SAVE** command and ASCII text files. When we first introduced the **SAVE** command, we did not tell you that three different formats enable you to save your programs:

ASCII text
binary
protected binary

"ASCII text" refers to a standard method by which your program saves every one of its characters on the diskette. You specify this option by appending a comma and an **A** to the **SAVE** command:

```
SAVE "filename",A [RETURN]
```

You can edit ASCII text files by using the **N** option in WordStar, but this type of file takes up more room on the diskette than the other types. ASCII, which stands for American Standard Code for Information Interchange, is a numbering system in which each different character has a different pattern of bits used to represent it; it is the format in which most computers store text.

Microsoft BASIC does not store your program in memory as ASCII text. Instead, it converts each command it recognizes into a single "special" character. For instance, it stores the command **PRINT** in memory as a 26, or 00011010 in the numbering method computers understand. This computer format is called "binary" because it uses the binary representation (so named because it involves counting by twos) to store each command. Text and variable names within a program are stored in ASCII format; but if you try to coax CP/M to print a copy of this file, all kinds of "weird" things will happen because of those binary commands. The binary format will be used to store your program files if you do not specify an option at the end of the **SAVE** command:

```
SAVE "program" [RETURN]
```

A third method to save programs is called "protected binary." This method uses a special "encrypted" form of the binary instructions, so that after the program has reloaded into the computer's memory, the program cannot be examined, only executed. DO NOT SAVE A PROGRAM IN THIS FORMAT unless you have copies in an unprotected format; otherwise, you'll never be able to make changes.

SAVE "program",P RETURN

We've come to the end of another section. By now you should be able to enter some simple BASIC programs, change them, store them on diskette, and retrieve them. We're now going to start introducing commands that build upon the base you already have. We strongly suggest that you try writing a simple program of your own without any additional instructions from us. When you pass this intermediate test, you'll want to plunge ahead. If you can't write a simple program yet, we suggest you review the material already presented before you continue.

Lesson 5: Getting Information to a Program

If BASIC were restricted to the commands we've presented, it wouldn't be very flexible.

BASIC is an "interactive" computer language. Interactive means that BASIC presents some information, you interact with—or react to—it, and BASIC presents some more information. So far we haven't shown you any ways—with the exception of typing ^C—to make BASIC pay any attention to what you're doing while a program is running; but we're now going to teach you how to do it.

The **PRINT** command sends information to your Osborne 1's screen. To get information from the Osborne keyboard, BASIC uses the **INPUT** command:

INPUT variablename

If the type of the variable indicates that it stores a number—as in integer, floating point, and double-precision variables—BASIC will wait for you to type a number when it encounters the above statement. BASIC indicates it is waiting for you to enter something by displaying a question mark. If you enter something other than a valid number, an error message (**?Redo from start**) will show up, and the question mark will reappear. Press **RETURN** to tell BASIC that you've finished entering a value.

You can also employ string variables—variables that contain letters and symbols as well as numbers—in an **INPUT** statement. To indicate that you are finished entering information into a string variable, press **RETURN** .

A special version of the **INPUT** command combines the **PRINT** and **INPUT** functions:

```
LINE INPUT"What is your name?";N$
```

The above example displays the message **What is your name?** first, and then waits for you to enter a name. Notice that the question mark is contained within the message to be displayed, as use of the **LINE INPUT** statement does not automatically supply it.

You can specify multiple variables with a single **INPUT** statement:

```
INPUT NAM$,DATE$,AGE
```

In this case, you must type the information in the order the program expects it—name first, date second, age third as in our example—and you must separate each distinct item from the others by using a comma, not **RETURN**. Use **RETURN** to tell BASIC you're done entering all items. If you are going to employ this last form of the **INPUT** statement, make sure you use a message that explains exactly what users are to do; otherwise, they may not enter the information the way BASIC expects it.

```
10 PRINT"Sample Multiple Data Entry Program"
20 PRINT
30 PRINT"Enter the two pieces of information"
```

```
40 PRINT"requested below in the order requested,"
50 PRINT"each set off by a comma. Press RETURN"
60 PRINT"when done."
70 PRINT
80 INPUT "Enter NAME and AGE: ";NAM$,AGE
90 END
```

NOTE

*Lines 20 and 70 use a simple trick in BASIC—a **PRINT** statement with nothing else on the line creates a blank line on the display because BASIC issues a **RETURN** when it comes to the end of a PRINT statement and doesn't find a comma or a semicolon telling it to stay at a location.*

What if you want to insert a comma into a string variable? You use a second variation of the **INPUT** command:

LINE INPUT"Enter CITY and STATE: ";CITYSTATE$

LINE INPUT accepts any character (up to 254 characters in all) in a string up to the terminating **RETURN**. If the words **LINE INPUT** are immediately followed by a semicolon—before the prompt message and variable name—pressing the **RETURN** key to terminate the input will not cause the cursor to move down a line, as normally happens when you press **RETURN**.

You now know how to get information and assign it to a variable. All of the methods so far described have the drawback that they require you to press the **RETURN** key when entry of information is complete. Even if you type only one character, you have to press two keys.

One way to cause BASIC to respond immediately to key presses is to use the **INPUT$** command:

CHARS$=INPUT$(NUMBER)

where **CHARS$** is the variable to store the input, and where **NUMBER** is the number of characters to expect.

As soon as BASIC receives the expected number of characters, program execution resumes with the next statement.

If you only want to check to see if a user has added some other character, use the following routine:

```
1000 'CHECK FOR CHARACTER ENTRY
1010 'CHAR=1 if character is ready, =0 if none
1020 '
1030      CHAR=0:CHAR$=""
1040      CHAR$=INKEY$
1050      IF CHAR$<>"" THEN CHAR=1
1060      RETURN
```

The key new statement here is the **INKEY$** command, which searches to see if anyone has entered a character. If so, **CHAR$** will contain that character upon execution of line 1040; if no character is ready, **CHAR$** will remain unchanged; we set it to nothing in line 1030.

The routine listed above is designed for use as a subroutine. Every time you wish to check to see if a character is entered, you add a **GOSUB 1000** command to your program. When the subroutine has executed and control has returned to the main portion of your program, the variable **CHAR** will have a value of **1** if a character has been entered, or **0** if no character is ready. You could then use the **IF** statement you learned earlier to conditionally execute portions of your program, depending on whether or not a character was typed.

Let's leave the subject of user-typed input and consider a different method of getting information to your program.

Wouldn't it be nice if you could "store" some data in a program and then look at it as you needed it? Well, you can. First, you need to store the data in **DATA** statements:

DATA 10,10,"Now, what is the",time,100

The above statement has five pieces of information:

information	type
10	numeric
10	numeric
"Now, what is the"	string
time	string
100	numeric

A comma separates each piece of data from the next. Notice that the third entry has a comma embedded in it and is surrounded by quotes. Use quotes to surround information destined for string variables only if you have a comma, an extra starting or ending space, or a colon in the data.

To use the information in a **DATA** statement, you must "read" it; the **READ** statement performs this task.

READ VALUE1,VALUE2,CHAR1$,CHAR2$,VALUE3

is a valid **READ** statement for the data in our example. You don't have to use all of the data in one **READ** statement. BASIC keeps track of how much of each **DATA** statement you've used, and will use the next unused data item each time it encounters a **READ**. The following is a valid method of using **DATA** and **READ**:

```
10 DATA Now, is, the, time
20 DATA for, all, good, people
30 DATA to, buy, Osborne, 1's
40 READ INFO1$,INFO2$,INFO3$,INFO4$
50 READ INFO5$
60 READ INFO6$,INFO7$
```

```
70 READ INFO8$,INFO9$,INFO10$,INFO11$,INFO12$
80 PRINT INFO1$+" "+INFO2$+" "INFO3$+" "+INFO4$;
90 PRINT " ";INFO9$+" "+INFO10$+" "+INFO11$+"
"+INFO12$
100 END
```

NOTE

As has been our habit in this chapter, we have introduced a new concept in the context of a programming example. Note that lines 80 and 90 have "concatenated" (brought together) items to print. Try the example and note that a space is inserted between each word and the next when the word is displayed.

You can tell BASIC to forget where the next data item is located and to go back to the start of, or other location in, the list by using the **RESTORE** statement in your program. **RESTORE** by itself tells BASIC to start at the first **DATA** statement it encounters. You can also specify a line number following **RESTORE**, in which case the **DATA** statement beginning on the line number specified becomes the location of the next data items the program will use.

This lesson has taught you new things about getting information into the computer. We'll close the section by briefly introducing a few more items about getting information out of the computer.

The Osborne 1 stores 128 characters on each line and then shows you 52 of them, but Microsoft BASIC was designed for use on terminals with 80-character line widths. Therefore, every time BASIC detects that it is doing something past column 79, it "wraps around" to the next line. You can change this "default value" by using the **WIDTH** statement:

WIDTH NUMBER

where **NUMBER** is the value you want as the new screen width. The two values that make the most sense are 52 and 128. As long as your printer can print more than 80 characters on a line, you can also change the width of lines that appear on your printer by typing:

WIDTH LPRINTER NUMBER

Using the commands we've given you to work with so far, in order to display spaces on the screen you must enclose them within quotes in a **PRINT** statement. Two ways are available to circumvent this requirement:

```
PRINT TAB(NUMBER)
PRINT SPC(NUMBER)
```

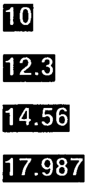
Either command will print the number of spaces you specify, and you can use either one in multiple-element **PRINT** commands:

```
PRINT "Now";TAB(30);"is";TAB(5)
```

We've saved the hardest command to master for the end of this lesson. If you've been experimenting while reading this chapter, you've probably noticed that it's not easy to line up the decimal places in amounts that contain different numbers of characters.

```
PRINT 10:PRINT 12.3:PRINT 14.56:PRINT 17.987
```

would appear as:



```
10
12.3
14.56
17.987
```

To make printed information appear the way you want it to, you use a formatting technique called a "mask." The mask is a string expression (it may be a variable) that contains a picture of the way you wish to format information. Here are the elements that go into the mask:

#	represents a digit position
+	represents the sign of a digit
–	represents trailing minus sign for negative numbers
**	means leading spaces will be filled with asterisks
$$	represents two digit positions and requests that a dollar sign appear at the left side of a number
&	represents a variable-length string position
\ \	enclose spaces to be printed
!	represents the first character of a string
^^^^	represents exponential notation
%	is a program flag to indicate that an oversize number has been printed
.	represents decimal point

Those descriptions probably don't make much sense to you yet. Let's show you an example of how to print numbers up to $100,000.00 in the dollars-and-cents format:

```
PRINT USING "$$#####.##";VARIABLE
```

The **USING** in the above statement is what tells BASIC that the information in the following string will be used for the mask. Note that the information printed in the above example will have leading blanks (spaces) if the number doesn't totally fill the format string.

The number of combinations of the masking elements listed above is extraordinary, so you need to experiment with this feature to achieve the results you desire. Use the Reference Guide to help you further understand how each masking element works.

Lesson 6: Advanced Variable Use and Functions

We're going to speed up the learning process a bit by using only brief introductions for most of the commands left for you to learn. We assume at this point that you are either a proficient programmer who is trying to learn the Microsoft BASIC syntax, or that you've consulted an outside work to learn more about BASIC and what it does.

If you're a newcomer to computing and have made it this far without outside help, GREAT! We are not the final authorities on everything, however, and we think some additional help will facilitate your learning. A book we think you'll find useful is *Programming for Poets: A Gentle Introduction Using BASIC* by Richard Conway, James Archer, and Ralph Conway. Even you experts out there will probably benefit from some outside material, and we suggest *Software Debugging for Microcomputers* by Robert Bruce.

You're continuing, eh? Okay, buckle up; the pace will be quicker.

Microsoft BASIC has built-in functions, much like the function keys on some hand calculators. These functions are often a shorthand way of performing an action or calculation that otherwise might take considerable programming effort. Let's briefly examine the functions available:

ABS(NUMBER)	the absolute value of the number
ASC(CHAR$)	the ASCII value of the first character in a string
ATN(NUMBER)	the arc tangent of the number
CDBL(NUMBER)	converts number to a double-precision number
CHR$(NUMBER)	the character string that has the ASCII value of the number
CINT(NUMBER)	converts number to an integer by rounding

COS(NUMBER)	the cosine of the number
CSNG(NUMBER)	converts number to single-precision number
EXP(NUMBER)	raises e to the power of the number
FIX(NUMBER)	the truncated portion of the number (not rounded)
FRE(NUMBER)	the amount of free memory available
HEX$(NUMBER)	the string in hexadecimal representation of the number
INT(NUMBER)	the rounded-down integer of the number
LOG(NUMBER)	the natural logarithm of the number
LPOS(NUMBER)	the current position of the printer's printhead
OCT$(NUMBER)	the string in octal representation of the number
PEEK(NUMBER)	the value stored at the memory location with the address of the number
POS(NUMBER)	the current position of the cursor
RND(NUMBER)	a random number between 0 and 1
SIN(NUMBER)	the sine of the number
SPACE$(NUMBER)	a string of spaces equal in length to the number
SQR(NUMBER)	the square root of the number
STR$(NUMBER)	the string equal to the value of the number
TAN(NUMBER)	the tangent of the number
VAL(CHAR$)	the numerical value of the string

The method by which you use these functions is to assign them to a variable:

VARIABLE=FUNCTION(PARAMETER)

Make sure that the type of variable you use matches the type of function you're using—string variables should be used with string functions, for example.

A second set of functions you'll want to know about are some special string functions:

> **LEFT$(CHAR$,NUMBER)** — used to get the string of characters of the number's length from the left-hand side of a string you specify.
>
> **RIGHT$(CHAR$,NUMBER)** — used to get the string of characters of the number's length from the right-hand side of a string you specify.
>
> **MID$(CHAR$,STARTPOS,NUMBER)** — used to get the string of characters of the number's length from the starting position in the middle of the string specified.
>
> **INSTR(STARTPOS,CHAR1$,CHAR2$)** — used to get the position (number) at which the occurrence of the second string is duplicated by the first string.

If you need more information about any of these functions, turn to the Reference Guide for a complete description of their use.

For the most part, we've presented the straightfoward aspects of Microsoft BASIC. It's now time to examine some of the "sneakier" commands.

DIMENSIONING VARIABLES: The term "dimensioning" refers to the process of telling BASIC ahead of time how much space to reserve for a variable. You must do this when you use a variable as an array: a table of values all referenced by the same name. Say you wanted to maintain a table of values consisting of four columns and four rows of information. You could do so by assigning the following variable names:

VALUE1	**VALUE2**	**VALUE3**	**VALUE4**
VALUE5	**VALUE6**	**VALUE7**	**VALUE8**
VALUE9	**VALUEA**	**VALUEB**	**VALUEC**
VALUED	**VALUEE**	**VALUEF**	**VALUE0**

This could become unwieldy, however, especially if you had a large table. In BASIC, you could create this table like this:

```
OPTION BASE 1
DIM VALUE(4,4)
```

Your table would now look like this:

VALUE(1,1)	**VALUE(1,2)**	**VALUE(1,3)**	**VALUE(1,4)**
VALUE(2,1)	**VALUE(2,2)**	**VALUE(2,3)**	**VALUE(2,4)**
VALUE(3,1)	**VALUE(3,2)**	**VALUE(3,3)**	**VALUE(3,4)**
VALUE(4,1)	**VALUE(4,2)**	**VALUE(4,3)**	**VALUE(4,4)**

Mathematicians will immediately recognize this concept as a "matrix." The numbers in the parentheses following the variable name are simply the location of that item in the matrix. In BASIC, we call this concept an "array," and this particular array is a two-dimensional one.

But what about that **OPTION BASE 1** statement—what is it doing? Most computers use internal counting procedures that begin with zero. The **OPTION BASE** statement we use before the **DIM** statement sets the internal counting BASIC uses to begin at one.

You must declare all multidimensioned variables by using a **DIM** statement before you attempt to use them in your program. The maximum number of dimensions in an array is 255, and the maximum number of entries in any one dimension is 32767. Dimensioning does take up memory space, so be careful not to get carried away with this concept, or else you might find you have little or no room to expand your program.

You can make BASIC "forget" your dimensioning commands by using the **ERASE** command followed by a list of variable names. If you do not use this statement and try to redimension a variable, you will get an error message.

Another method of quickly starting over with variables is to use the **CLEAR** command:

CLEAR ,topofmemory,stackspace

Topofmemory is the topmost memory address BASIC should use, and **stackspace** is the amount of stack space (temporary information storage) you wish Microsoft BASIC to use. All numeric variables are set equal to zero, while string variables are set to "null" ("" indicates "nothing" between the quotes) when you use the **CLEAR** command.

DEFINING: You can assign a variable type to all variable names beginning with a single letter or range of letters by using the **DEF** statements in Microsoft BASIC:

DEFINT A defines all names beginning with A as being integer variables.

DEFSTR B-C defines all names beginning with B and C as being string variables.

DEFSNG A,Z defines all names beginning with A and Z as being single-precision variables.

DEFDBL A-Z defines all variable names as being double-precision variables.

In general, Microsoft BASIC deals with integer variables the most quickly, and with double-precision and string variables the most slowly. Your programs will execute more quickly if you use integer variables for **FOR/NEXT** loops and other control structures where possible.

OTHER VARIABLE TRICKS: Let's say that you have two variables and want to exchange their values. In most BASICs, you effect the exchange by entering something like this:

```
5   'Initial VALUEs
10  X=10
20  Y=20
30  'Now swap 'em
40  Z=X:'use dummy variable temporarily
50  X=Y
60  Y=Z
```

Keeping track of that third, "dummy," variable is sometimes difficult, and it certainly masks your real intent: you don't want to assign a value to Z; you just want to swap X and Y. Microsoft BASIC lets you use one statement to perform the swap:

SWAP variable1,variable2

The only limitation of this command is that the variable types must match.

Lesson 7: Advanced Control Structures

You already know about the **IF/THEN** structure introduced earlier. Let's add some new variations to the control structures you know.

One method of extending **IF/THEN** is to put multiple statements to the right of the test:

IF test THEN statement1:statement2:statement3

Each statement following the **THEN** will execute, in order, only if the test condition is true (unequal to 0).

You can also extend the **IF/THEN** construct by using the **ELSE** option:

IF test THEN statementl ELSE statement2

This structure is useful if you wish to program an "either/or" situation. In the above example, if the test proves "true," then "statementl" executes; otherwise, "statement2" executes.

What if you want to test for more than two conditions (true and false)? Simple—use the **ON/GOTO** or **ON/GOSUB** construct:

ON value GOTO placel,place2,place3, etc.
ON value GOSUB placel,place2,place3, etc.

The "value" encountered directs execution to the "place" that corresponds to that value. So, if the value is 1, the first line number named ("placel") will be used; if the value is 2, the second line number named is used, and so on. When you use **ON/GOTO** or **ON/GOSUB**, make sure you have a line number to go to for every possible value you can encounter. If the value is greater than the number of line numbers listed, execution will drop down to the statement following **ON/GOTO** or **ON/GOSUB**. Most programmers put a "trap" either immediately preceding or immediately following the **ON/GOTO/GOSUB** construct so that an unexpected value is captured and reported rather than allowed to "wander off in the program untended."

The last control structure we'll examine is the **WHILE/WEND** construct. Here is the basic layout of this structure:

WHILE test
STATEMENTS
WEND

The **STATEMENTS** in the body of the **WHILE/WEND** loop will execute over and over, in order, until the test is equal to zero. This loop structure is similar to the **FOR/NEXT** loop. Unlike the

FOR/NEXT loop, however, the **WHILE/WEND** structure executes a variable number of times. It is important to make sure you've provided a way for the test to become false, because you may otherwise find that you have coded another "endless loop."

Lesson 8: Disk Files and BASIC

Even with all the fancy and complicated BASIC commands you've learned, Microsoft BASIC wouldn't really be too useful without a permanent method for storing and retrieving information your programs use. One reason we've saved this section for last is that the way Microsoft BASIC "talks to" disk files is not intuitively obvious to casual observers.

Two methods exist to store information on a diskette and retrieve it: sequential access and random access. A file you maintain using "sequential access" is one in which you must always start with the first piece of information in the file and work your way to the item you want. Watching TV is a sequential-access activity—even if you want to watch only the last five minutes of a program, the first 25 minutes must still play.

Take an example. Say you are storing the scores of your five favorite football teams on a diskette. One week's scores might be:

Team 1	**45**
Team 2	**14**
Team 3	**20**
Team 4	**21**
Team 5	**32**

You might store this in a sequential-access file like this:

```
1,45,2,14,3,20,4,21,5,32
```

The format in the above example is team number, score, team number, score, and so on. To find the score for team 5, you must first "read" all the rest of the information in the file that precedes it.

Each piece of data in a sequential-access file is separated from the others by a comma. Let's examine the process necessary to create a sequential-access file:

1. "Open" the file to be used (ready it for use).
2. "Write" information to it.
3. "Close" the file when you finish adding information.

Each process above has a BASIC command associated with it. Using our football-score example, the following program statements would create a sequential-access file:

```
100'  CREATE A FOOTBALL SCORE SEQUENTIAL-
      ACCESS FILE
110'
120   OPEN "O",#1,"SCORES.DAT"
130   PRINT#1,1,45,2,14,3,20,4,21,5,32
140   CLOSE #1
```

The format of the **OPEN** statement consists of the following information:

- the word OPEN;
- O to write information to a file, or I to get information from a file;
- an arbitrary number assigned to that file (#1 here); and
- the name of the file to be used.

The file has an assigned number so you can have multiple files all open for use at the same time. Since each file has a different number associated with it, Microsoft BASIC knows which file you wish to write information to.

In fact, line 130 in our example shows how you specify how to write to a sequential-access file. Note that—except for the #1—the write-to-file statement takes the same form as the **PRINT** statement. When Microsoft BASIC sees a number sign followed by a number in a print statement, it assumes that you are referencing a file and checks to make sure it is open for use.

Okay, you now know how to create and write information to a sequential-access file—how about getting information out of it? Simple—you can substitute the following lines in the above program:

```
120   OPEN "I",#1,"SCORES.DAT"
125   FOR LOOP=1 TO 5
130         INPUT #1,TEAM,SCORE
135         PRINT TEAM, SCORE
137   NEXT LOOP
```

The lines above read a team number and score from the file and display them on your screen.

Next you need to know how to add information to a sequential file . . . it's not as easy as you may think. To add information, follow these steps:

1. Open the file you want to add to in the I (for input) mode.
2. Open a second file with a "dummy" name in the O (for output) mode.
3. Read the data from the first file and write it to the second.
4. Add your new information to the second file.
5. Close the second file.
6. Close the first file and use the KILL command to delete it.

7. Rename the second file using the NAME AS command. Make sure that it ends up with the same name as the first file (the one you were adding to).

This procedure can be complicated, and the limitation of disk-storage space available will compound your problems if you are maintaining a lot of data in a sequential-access file because your original file cannot be larger than 46K for the above scenario to work correctly.

Two functions sequential-access files often use are **EOF** and **LOC**.

EOF(filenumber) is true if the next item in the file is the file's end and false if you're in the middle of a file.

LOC(filenumber) gives the number of sectors read or written to in a sequential file.

Use these functions just as any other function we described earlier.

Random-access files may require a little extra effort to learn; but once you've written a few programs using random files, you'll find that the commands come naturally to you.

A random file is one in which everything is ordered in such a way that you can go directly to a piece of information you wish to read or write. If watching TV is somewhat akin to reading a sequential file, using a videotape recorder to find a particular frame of a TV show is akin to reading a random-access file.

Let's continue to use our example of five football teams and their scores from last week's games. What if every team number consisted of one digit, and every score consisted of two digits? Every combination of team and score would occupy three characters in our file. To get to team three's score, we'd then need to tell the computer to go to the eighth and ninth characters in the file and read them. So we'll find something like this in the file:

```
145214320421532
```

That mishmash isn't very easy to scan, but that's okay because you're going to make the computer scan it. Before you continue, however, make sure that when you first set up a random-access file you allow for all possibilities. You must consider what will happen if you have a score less than 10 or greater than 99. So that your program can accommodate such an eventuality, you should make team numbers two digits in length and use zeros in front of single-digit numbers. You might also want to make the scores three digits long if you expect prodigious point production.

Let's create our random file. In this example, we're going to use comments on every line to explain what is happening.

```
100 '  CREATE A RANDOM DISK FILE OF TEAMS AND
       SCORES
110 '
120    OPEN "R",#1,"SCORES.DAT",4:'note "R" and number
       at end
130    FIELD #1,2 AS TEAM$,2 AS SCORE$:'we define
       fields in file
140    FOR LOOP=1 TO 5
150        READ TEAMNUM$,TEAMSCORE$:'scores are in
           a DATA statement
160        LSET TEAM$=TEAMNUM$:'get information into
           field
170        LSET SCORE$=TEAMSCORE$:'get information
           into field
180        PUT #1,LOOP:'store it on disk
190    NEXT LOOP
200    CLOSE #1
210    DATA 01,45,02,14,03,20,04,21,05,32
```

The **OPEN** statement for a random file is different in two respects from the sequential-access **OPEN** statement: first, you use **R** (for "random") instead of **I** or **O**; second, after the file name you must add a comma and tell Microsoft BASIC how long each "record" in the file will be.

A "record" is one complete set of data. It's kind of like a file folder in a file drawer. Within the record can be a number of "fields." A field is one individual set of data, like a sheet of paper in a file folder. Here's an example of one file hierarchy:

file:	Osborne owners' names and addresses	"OZ.DAT"
record:	Information on one Osborne owner	RECNUM%
fields:	Osborne owner's name	NAM$
	Osborne owner's address	ADDRESS$
	Osborne owner's city	CITY$
	Osborne owner's state	STATE$
	Osborne owner's ZIP code	ZIP$

We have put the variable names that reference each item in the right-hand column.

Next in the sample random-disk-access program comes the **FIELD** statement. Given the information we just presented, you should guess that you have to identify each field and give it a length. The total length of the fields must add up to the total length specified in the **OPEN** statement.

To write information into a random-access file, you first must move the information into the variables named in the **FIELD** statement, using either **LSET** (for leftset, or left justify) or **RSET** (for rightset, or right justify). Any information that "spills over" the amount of room set up for a given field will vanish.

NOTE

DO NOT USE THE VARIABLES NAMED IN THE FIELD STATEMENT FOR ANYTHING EXCEPT MOVING INFORMATION TO AND FROM THE DISKETTE. If you use one of these variables to display something on the screen or to change a value, you will confuse Microsoft BASIC's internal buffer "pointers" and get some interesting—but incorrect—results.

Finally, to actually write the information onto the diskette, use the **PUT** statement. Its two elements are the file number you'll use and the record number where you want to store the information. A nice feature of random-access files is that you can begin by storing information in record number 50, if you want, and then come back and fill in the rest at a later time.

To get information from a random-access disk file, simply get rid of the **LSET** or **RSET** statements and use **GET** in place of **PUT**. After your **GET** statement, you'll want to move the information from the field variables to variables you can use within your program:

```
OPEN "R",#1,"SCORES.DAT",4
FIELD #1,2 AS TEAM$,2 AS SCORE$
FOR LOOP=1 TO 5
     GET #1,LOOP
     TEAMNUM$(LOOP)=TEAM$
     TEAMSCORE$(LOOP)=SCORE$
NEXT LOOP
END
```

We keep using string variables in our examples to store numeric information because Microsoft BASIC stores all information on disk in a special binary form. We're using strings so you don't have to deal with the conversion from the binary form to the one used within your program.

To convert numbers into their requisite format for storage in random-access files, you must use one of the following commands to do the conversion in conjunction with the the **LSET** or **RSET** command:

LSET(FIELDVARIABLE)=MKI$(VARIABLE) for integer variables
LSET(FIELDVARIABLE)=MKS$(VARIABLE) for single precision
LSET(FIELDVARIABLE)=MKD$(VARIABLE) for double precision

To restore the information to the form you can use within your program, use the following conversion commands instead of the **LET** statement used in the above read-from-random-file example:

VARIABLE=CVI(FIELDVARIABLE)	for integer variables
VARIABLE=CVS(FIELDVARIABLE)	for single precision
VARIABLE=CVD(FIELDVARIABLE)	for double precision

We want to tell you only one more thing about files, and that is how to change diskettes and let Microsoft BASIC know that you've done so. Use the **RESET** command. Simply placing **RESET** in your program at the point immediately before you remove the first diskette from the drive closes all files in use and rewrites the correct information about them in the CP/M directory onto that diskette.

Closing Up the BASIC Shop

A few commands we have not covered in this text are fully explained in the Reference Guide:

CALL	**PEEK**	**POKE**	**INP**	**OUT**
DEF USR	**WAIT**	**DEF FN**	**USR**	**&H**
&O	**RANDOMIZE**	**VARPTR**	**ERR**	**ERL**
ERROR	**ON ERROR**	**NULL**	**RESUME**	**/F:**
/M:	**/S:**			

We hope we gave you a good "taste" for Microsoft BASIC. You can find more information on the language in *Microsoft BASIC* by Ken Knecht. Also, *TRS-80 Disk BASIC* is almost exactly the same as the version of BASIC provided with the Osborne 1. For more information on CBASIC, try the *CBASIC User Guide* by Adam Osborne, Gordon Eubanks, and Martin McNiff.

CHAPTER 8

CP/M Revisited

This chapter is intended to teach advanced Osborne 1 users some of the technical details of CP/M, including how to use the assembly-language utility programs supplied with CP/M. Reference to CP/M addresses within this chapter pertain specifically to the attributes of the 1.4 ROM and BIOS.

Most users of the Osborne 1 will not need to read this chapter. The material in this chapter and the next are presented as a discussion and differ from preceding chapters which were interactive tutorials. Topics covered here include the advanced and technical aspects of CP/M and the utility programs that accompany it. Certain exercises are highlighted as in the tutorials just in case you want to follow along.

Lesson 1: Creating New CP/M Systems

Some day you may encounter a few circumstances in which you'll need to modify CP/M or create a CP/M system of a different size. Two utilities help you do so.

SYSGEN is a program that reads a copy of the CP/M system from the first tracks of a diskette where it resides and transfers it into the main memory of your Osborne 1. Once CP/M operating system is in memory, SYSGEN asks you on which diskette you wish to save it. Press RETURN instead of specifying a drive, and you will see the A > return.

Each time you type SYSGEN from the A > prompt, SYSGEN will prompt you for the drive you wish to load CP/M from. If you press RETURN without specifying a drive letter, SYSGEN will assume that CP/M is already in memory. There are three steps required to modify CP/M at the machine-language level:

1. use SYSGEN to place a copy of CP/M in memory,
2. use DDT to examine and modify CP/M, and
3. use SYSGEN to place CP/M back onto a diskette.

SYSGEN's other purpose is used to copy CP/M from diskette to diskette. You need to do this for diskettes you wish to place in drive A to start up the system after you've pressed the RESET button or just turned ON the power.

MOVCPM is the second utility for changing CP/M. Normally, the CP/M you use on your Osborne 1 allows the use of 59K of memory, with CP/M filling the uppermost area of memory —from about 52K to 59K.

Some programs require that you create a different-size CP/M system, usually so that you can place special instructions in a protected area above that occupied by CP/M. To do so, you would issue:

```
MOVCPM ## * <cr>
```

where ## is the size CP/M system you wish to create. The video display uses the top 4K of memory, with 1K used as a disk buffer, so the largest CP/M system you can create is 59K. We forewarn experienced CP/M users that merely issuing **MOVCPM** and pressing **RETURN**— usually used to create the largest possible CP/M system—won't work on the Osborne 1 because CP/M will find the video memory and think that it is available for use. Instead, to create the largest possible CP/M system on the Osborne 1, type:

```
MOVCPM * * <cr>
```

Here is an example of how to use MOVCPM: To leave 4K of memory space between the last memory CP/M uses and the first memory the video monitor uses, type:

```
M O V C P M SPACE BAR 5 5 SPACE BAR
* RETURN
```

MOVCPM replies with the message:

```
READY FOR "SYSGEN" OR
"SAVE 39 CPM55.COM"
```

This message means that the new version of CP/M is in memory, and you should either use the CP/M SAVE command to place the copy in a file on diskette for later use, or use the SYSGEN program to save this new CP/M on diskette. You must perform one of these two options immediately. Any intervening command may change the CP/M instructions stored in memory.

If you used SYSGEN to save the new CP/M, you should wait for the disk activity light to go off and then press the RESET button. Place the diskette with the new CP/M system in drive A and press RETURN. You should see A > appear momentarily. If the diskette you created didn't have a program named **AUTOST.COM** on it, you might also see the **AUTOST?** message.

Several things can go wrong if you start playing with MOVCPM and SYSGEN. Make sure that the first few times you use these commands you are using diskettes from which you can afford to lose information. Watch out for the following:

- Making a CP/M system too large. You can't use MOVCPM for any size larger than 59K.
- Saving "junk," instead of CP/M, on a diskette. If you don't have a copy of CP/M in memory when you use SYSGEN to write to a diskette, that diskette will not have CP/M on it and will act erratically.
- Modifying CP/M incorrectly. If you use DDT to examine and modify CP/M, make sure you know what you're doing and why.
- Making a CP/M system too small. The smallest size CP/M system you can create on an Osborne 1 is 20K.

Lesson 2: Odds and Ends

Some Editing Characters

You've used WordStar and are aware of the way electronic editing works. Here's a complete list of the control characters CP/M recognizes and what each does (we neglected to tell you about a few "editing" commands that you can use when you see the A > in CP/M in chapter 3):

^E Performs a line feed and carriage return without sending the command to be processed; useful for typing commands longer than 52 characters without using the scrolling feature.

^H backspaces one character; same as left arrow.

^J ends command; same as RETURN.

^M ends command; same as RETURN.

^R repeats the current command line for verification.

^U or ^X cancels the entire command line.

^- deletes one character by echoing it on the screen instead of erasing it (as does the left-arrow key).

You can use these special editing characters any time that CP/M interprets keyboard input—as it does when you're using **DDT**, **PIP**, and most of the rest of the CP/M utilities. Most special programs, such as WordStar or SuperCalc, bypass this feature and substitute their own editing capabilities.

STAT filename $SYS

Something you might find interesting is the ability to change any file into a CP/M system file. To make a file a system file, you type:

S T A T SPACE BAR filename.typ SPACE BAR $ S Y S

RETURN

where **filename.typ** is any valid CP/M file name or ambiguous file name. Remember the [*] and [?] characters and their use in defining file names? When you perform this action, CP/M replies **filename.typ set to SYS** for each file turned into a system file.

The problem with this is that system files don't appear in directories. The only way you can find out if a system file exists on a diskette is with XDIR (extended directory) or:

[S] [T] [A] [T] [SPACE BAR] [*] [.] [*] [RETURN]

which will identify all files, with system files shown in parentheses. To turn a system file back into a normal file, type:

[S] [T] [A] [T] [SPACE BAR] filename.typ [SPACE BAR] [$] [D] [I] [R]
[RETURN]

Issuing the above command sequence returns the appropriate files to their original (called "directory") status.

While we're concentrated on the intricacies of the **STAT** command, we want to briefly mention two other similar uses:

STAT filename.typ $R/O RETURN	makes a file that can only be read, not changed
STAT filename.typ $R/W RETURN	makes a normal file that can be changed as well as read

R/W and **R/O** stand for "read/write" and "read/only," respectively.

"STAT"ting Other Things

We still haven't come to the end of the list of uses for STAT. So far, you know about finding out how much room files take up (chapter 3) and changing files to different "types" (the last section). Here, briefly, are some other uses of STAT:

STAT DEV:<cr>	reports on current devices
STAT VAL:<cr>	lists possible device assignments
STAT USR:<cr>	reports user numbers in use
STAT DSK:<cr>	reports statistics on disk drives
STAT X:=R/O<cr>	changes diskette in drive X: into a "read/only" diskette
STAT log:=phy:<cr>	assigns a "physical" device to a "logical" one

Note the recurrence of of the word device. A device is a "unit" or "module" of your Osborne 1. Unfortunately, the standard CP/M names for devices don't quite match the names you probably already know:

CP/M DEVICE	**CP/M TYPE**	**OSBORNE 1 DEVICE**
CON:	logical	keyboard + screen
RDR:	logical	serial/parallel/IEEE*
PUN:	logical	serial/parallel/IEEE*
LST:	logical	serial/parallel/IEEE*
TTY:	physical	keyboard + screen
CRT:	physical	serial
BAT:	physical	parallel
UC1:	physical	IEEE
PTR:	physical	serial
UR1:	physical	parallel
UR2:	physical	IEEE
PTP:	physical	serial
UP1:	physical	parallel
UP2:	physical	IEEE

LPT:	physical	parallel
UL1:	physical	IEEE

*depends on physical assignment

In the table, a "physical" device is one that exists physically on the Osborne 1: the serial port or the parallel port. A "logical" device is a fictional device CP/M uses for various functions. Printing goes to the "logical LST: device," for instance.

The table means that you can, more or less, assign any physical device to any logical device. Say you wanted to send printing to the screen instead of to the serial port; you'd issue:

[S] [T] [A] [T] [SPACE BAR] [L] [S] [T] [:] [=] [T] [T] [Y] [:] [RETURN]

Now all printing (the LST: device in CP/M) will go to the screen instead of to the printer.

PIP Again

Having introduced you to "devices" under CP/M, we can now tell you about some of the other uses of the **PIP** program you encountered briefly in chapter 3.

The device names listed for STAT also apply to PIP; so instead of copying from file to file as we showed you earlier, you can also copy from file to device, device to file, or device to device. What does this mean in plain English? Say you wanted to copy a file from a diskette to your printer and didn't want to run WordStar to create a printout. You already know that you can use the CP/M commands TYPE and ^P together to send a file to the printer. Sometimes using TYPE is not a satisfactory solution, however, as the screen will display the information being printed—perhaps you want to go on to see some important information now on the screen but also need a printout of a file. The following PIP command is valid and solves your problem:

[P] [I] [P] [SPACE BAR] [L] [S] [T] [:] [=] filename.typ [RETURN]

Actually, for files that contain only text, PIP is a wonderful resource. Not only can you send files to your printer, but you can "capture" files from another computer by connecting them with an RS-232 cable and typing:

P I P SPACE BAR filename.typ =

R D R : RETURN

Five additional "special" devices are available using **PIP**:

NUL: sends 40 "null" characters (00 hex)
EOF: sends a CP/M end-of-file marker (1A hex)
OUT: uses an output routine you must create
INP: uses an input routine you must create
PRN: a special LST: device that prints eight spaces for each tab character, line numbers, and pagination

Be careful that your PIP command makes sense. Sending a printer device **(LST:)** to a diskette file makes no sense since your printer is unable to send any characters. PIP will usually catch such mistakes and issue a message similar to **CANNOT READ: LST:**—but you'll save yourself a lot of frustration by always double-checking the instruction you issue to CP/M before pressing RETURN.

Another aspect of PIP is the options you can instruct it to use during the copying process. We'll briefly describe each one here, but if you need more information, please consult the Reference Guide or one of the books on CP/M mentioned earlier.

B "Block mode" transfer
D# Deletes characters after #th column
E Echoes the copying to the Osborne 1 screen
F Form feeds removed during copying
G# Gets file from user area #
H Hex-format file transfer method used
I Ignores "null" characters (00 hex)
L Lowercase conversion (all characters)

N	Numbers each line during transfer
O	Object file transfer, not text file
P#	Paginates transfer every #th line
Qstring^Z	Quits copying when "string" found
R	Allows "system files" to be copied
Sstring^Z	Starts copying when "string" found
T#	Tab stops set at # columns
U	Uppercase conversion (all characters)
V	Verifies copy before completion
W	Writes to "R/O" files
Z	"Zeroes" the parity bit (eighth bit) on all ASCII characters

The format used to specify an option in a PIP command is:

```
PIP destination = source[options] RETURN
```

About the above structure, you should note:

- Options are enclosed in a set of square brackets.
- The first bracket must immediately follow the final character in the source specification; no spaces are allowed.
- More than one option is allowed (you may use blanks to separate them at your discretion).
- The options specified apply to the entire copying process to the left of the options.

The last point probably doesn't make any sense to you yet because we haven't told you that you can include multiple copying commands in a PIP command line:

```
PIP dest = source[options],dest = source[options] RETURN
```

Commas should separate commands. As long as the total number of characters in your overall command does not exceed 128, you can have any number of commands on the line. We've briefly mentioned all of PIP's capabilities. If you feel you need more information about anything in this chapter, consult a reference work on the relevant subject. See notes, page 760.

The Missing Program

Even though we've supplied Digital Research's **ED** program on your CP/M Utility diskette, we're not going to describe how to use it in this tutorial manual. ED was designed for use with a Teletype terminal; your Osborne 1, a CP/M system, is much more sophisticated than the system for which ED was designed. We see no need to burden you with additional information, especially when 99 percent of Osborne 1 owners will probably never use ED.

Lesson 3: Assembly-Language Programming

On your CP/M Utility diskette you'll find utility programs suited for 8080 assembly-language programming and debugging:

ASM.COM	an 8080 two-pass assembler
DDT.COM	an 8080 dynamic debugging tool
DUMP.COM	lists file contents in hex
LOAD.COM	converts hex files to command files

To show you how to use these four assembly-language utility programs to create a finished program, we'll assume that you're using the source code (assembly-language instructions) for the Osborne **AUTOST.COM** program listed below. Take a close look at the code so that the descriptions of each element that follow will make more sense to you.

```
;-----------------------------------

;AUTOCPM.ASM       version 2.0      21 Sept 82
;
; copyright 1982 by Thom Hogan
;                   THInc.
;
;This program executes CP/M command at COMMAND:
;
;-----------------------------------
```

```
;
; EQUATES
;
clear    equ   26          ;clear screen
escape   equ   1bh         ;escape character
pbuff    equ   9           ;BDOS print buffer
bdos     equ   5           ;location of BDOS
cr       equ   0dh         ;carriage return
lf       equ   0ah         ;line feed
;
; START OF PROGRAM
;
         org   0100h
;
; BEGIN: Changes logo letters into graphic
;        characters for display.
;
         lhld  01          ;find BIOS
         mvi   1,00        ;zero low-order byte
         mov   a,h
         sui   16h         ;subtract BIAS from BIOS
         mov   h,a         ;you now have hi-order byte
         shld  ccp         ; of address of CCP; store it
         lxi   d,logo      ;point to coded graphics
         lxi   b,2047      ;set up counter
over:    ldax  d           ;get the byte pointed to
         sui   65          ;subtract the magic coded number
         stax  d           ;save the new byte
         inx   d           ;increment the pointer
         dcx   b           ;decrement the counter
         mov   a,b         ;check to see if we're done
         ora   a
         jnz   over        ;if zero, we're done, otherwise . . .
;
; START: displays all information on screen
;
start:
         lxi   d,init      ;point to initialization message
         call  print
         lxi   d,0f000h    ;point to video RAM
```

```
            lxi     h,logo          ;point to logo
            lxi     b,14*128        ;number of bytes to move
            db      0edh,0b0h       ;fake LDIR
            lxi     d,endmes        ;point to loadmessage
            call    print
            lxi     d,command       ;point to file name
            lxi     b,10            ;set counter
;
; MOVE: moves information to the CCP locations for
;           autostart process
;
move:
            lhld    ccp             ;get CCP back
            mvi     1,07            ;calculate autostart address
            call    again           ;move it
            lhld    ccp             ;get the CCP back
            mvi     1,88h           ;point to autostart pointer
            mvi     a,08h           ;lsb of ccp pointer
            mov     m,a             ;put it in place
            lhld    ccp             ;get CCP one more time. . .
            mvi     1,89h
            mov     a,h
            mov     m,a
            lhld    ccp             ;and get it back again
            pchl                    ;execute cold start
again:
            ldax    d               ;get byte to move
            mov     m,a             ;move it
            inx     h               ;increment location
            inx     d               ;increment location
            dcx     b               ;decrement counter
            mov     a,b             ;get counter in a
            ora     c               ;check if done
            jnz     again           ;...not done
            ret                     ;...done
print:
            mvi     c,pbuff         ;get proper call in c
            jmp     bdos            ;do it
;
; STORAGE AREA
```

```
;
ccp:      ds     2              ;place to store CCP address
command:
          db     04,'HELP',0,0,0,0,0,0,0
;                ↑   ↑___     ↑______________
;                |       |                  |
;             length    command  must be followed by a
;          of command   string   zero to work properly
;
; note: if command string is longer than 8 characters,
;       you must change "lxi b,10" to lxi b,length+2"
;
init:     db     clear,'$'
;
; note: may put anything in the following lines of code that
;       you wish to, although be careful to make sure that it is
;       8 lines or less.
;
endmes:   db     escape,'=',16+32,1+32    ;position cursor
db        '                                                              ',cr,lf
db        '                        LOADING CP/M AND HELP...              ',cr,lf
db        '                                                              ',cr,lf
db        '                                                              ',cr,lf
db        '                        Osborne Computer Corporation          ',cr,lf
db        '                           26538 Danti Court                  ',cr,lf
db        '                           Hayward, CA 94545                  ',cr,lf
db        '                                                             $'
;
; DO NOT DISTURB THE FOLLOWING information as it contains
; a coded graphic display. Changing anything from here to the
; end of the program may result in catastrophic results.
;
logo:     db     'YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaaaKWWWWWWWWWWWWWWWWWWWWI'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY'
          db     'laaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaKWWWWWWZaaaaaaaaaaVWWWWW'
          db     'Wlaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY'
          db     'YYaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaWWWWWWWaaaaaaaaaaaaWWWWW'
          db     'WWaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY'
          db     'YYaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaWWWWWWWaaaaaaaaaaaaWWWWW'
          db     'WWaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY'
          db     'YYaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaWWWWWWWaaaaaaaaaaaaWWWWW'
          db     'WWaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db     'aaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db      'YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY'
          db      'YYaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db      'aaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db      'aaaaaaaaaaaaaaaaaaaaaaaaaaVWWWWWWIaaaaaaaaaaKWWWWW'

          db      'WZaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db      'aaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db      'YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY'
          db      'Zaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db      'aaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db      'aaaaaaaaaaaaaaaaaaaaaaaaaaaaVWWWWWWWWWWWWWWWWWWWWZ'
          db      'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
          db      'aaaaaaaaaaaaaaaaaaaaaaaaaaaa$'

end
```

The assembly language the CP/M assembler uses is called Intel format 8080 source code. The actual instructions within your program must conform to the syntax described in the Intel 8080 Reference Manual. Most of the examples in this User's Reference Guide are written in Intel 8080 code so that the ASM assembler may be used; however, the IEEE appendix uses Z80 code. A table of valid Intel assembly language instructions appears below:

ADD
ADI
ADC
ACI
ANA
ANI
CALL
CZ
CNZ
CP
CM
CC
CNC
CPE
CPO
CMA
CMC
CMP
CPI
DAA
DAD
DCR
DCX
DI
EI
HLT
IN
INR
INX
JMP
JZ
JNZ
JP
JM
JC
JNC
JPE
JPO
LDA
LDAX
LHLD
LXI
MOV
MVI
NOP
ORA
ORI
OUT
PCHL
POP
PUSH
RAL
RAR
RLC
RRC
RET
RZ
RNZ
RP
RM
RC
RNC
RPE
RPO
RST
SPHL
SHLD
STA
STAX
STC
SUB
SUI
SBB
SBI
XCHG
XTHL
XRA
XRI

Each instruction above is called a "mnemonic." Many mnemonics require additional "arguments." Thus, a complete 8080 instruction might look like this:

```
MOV B,A
```

where **MOV** is the instruction mnemonic and **B,A** is the argument. The above instructions say to move the contents of internal register A to internal register B.

To use assembly language correctly, you'll need to purchase a book describing the action of each 8080 instruction. A good choice is *8080a/8085 Assembly Language Programming* from Osborne/McGraw-Hill.

In addition to the 8080 instruction set, the CP/M assembler also recognizes special "directives," instructions that instruct the assembler to perform a special operation not defined by the 8080 instruction set. A list of the CP/M assembler directives follows:

DB	initializes memory byte by byte
DW	initializes memory two bytes at a time
DS	reserves an area of memory for storage
END	tells the assembler that it has come to the file's end
EQU	assigns a permanent value to a label
IF	allows conditional assembly of sections of code
ENDIF	defines end of conditional-assembly section
ORG	defines beginning location of the next instructions
SET	assigns a temporary value to a label

In assembly-language programming, each line of text contains one instruction to the assembler. The format of each line is:

line # label MNEMONIC argument* comment

*sometimes referred to as an "operand"

Each of the above elements, if present, must be set off by at least one space; assembly-language programmers usually separate elements by using tab characters.

The **LINE #, LABEL**, and **COMMENT** "fields" in assembly language are optional. If you use WordStar to create your assembly-language instructions, you won't use line numbers.

LABELs identify a memory address or a value. Valid labels may be 1 to 16 characters in length; the first character must be a letter. The CP/M assembler automatically translates lowercase letters to uppercase. **LABEL**s are optional in all statements except **EQU** and **SET** directives. In practice, if you properly assign a value to a **LABEL**, you can use **LABEL**s as the arguments in many assembly-language instructions. For example:

LABEL	MNEMONIC	OPERAND	COMMENT
PLACE	EQU	0E500h	;assign value of ;0E500 hex to PLACE
	JMP	PLACE	;transfers program flow ;to location just defined

Be careful not to use any of the assembly-language instructions, directives, or register names as a **LABEL**, as this will confuse the assembler.

COMMENTs always start with a semicolon. If there is nothing else on the line, a **COMMENT** can start at the left margin; just remember that everything the assembler encounters on a line to the right of the first semicolon will be considered a **COMMENT**. Consequently, you cannot start a line with a **COMMENT** and then place an instruction following it.

You can also use defined values (called "constants") or expressions in the argument or operand field. The following conventions are recognizeable to the CP/M assembler:

11	interpreted as 11 decimal
11D	interpreted as 11 decimal
11B	interpreted as 11 binary (3 decimal)
11H	interpreted as 11 hex (17 decimal)
11O	interpreted as 11 octal (9 decimal)
11Q	interpreted as 11 octal (9 decimal)
+	represents addition
−	represents subtraction
*	represents multiplication
/	represents integer portion of division
MOD	represents remainder portion of division
NOT	bit-by-bit logical complement
AND	bit-by-bit logical AND
OR	bit-by-bit logical OR
XOR	bit-by-bit logical exclusive OR
SHL #	shift left # of bit positions
SHR #	shift right # of bit positions

The assembler performs all arithmetic operations using 16-bit unsigned numbers. Valid expressions in the operand field using some of the above conventions are:

16 MOD 8	result = 0
5+LABEL	adds 5 to value of LABEL
111b AND 11b	result = 011b

If you like, follow along and try some assembly-language source codes. Put your WordStar diskette into drive A, and a blank, formatted diskette into drive B. Start WordStar as normal, but instead of using the **D** option to create a document file, use the N option to create a non-document file. Specify the name of the file to edit as B:AUTOST.ASM: Next, we will show you how to create a new version of the autostart program listed earlier.

Enter the text for AUTOST.COM using the normal WordStar editing features. (Do not use **^B**.) The primary difference between the document and non-document methods of creating files is that the non-document way does not insert extra characters for automatic margin control or justification. Use the TAB key on the Osborne to separate the various fields on each line of assembly-language code—it's faster than spacing and should result in the vertical alignment of every field, as in our example.

Make sure that your cursor is at the end of the file before using the ^$\boxed{\text{K}}\boxed{\text{X}}$ command to finish editing the file.

You're now ready to "assemble" the source code you just entered. Put your CP/M Utility diskette into the A drive. Before we proceed, we'll explain what happens next and what your options are.

On the CP/M Utility diskette is a file named **ASM.COM**. This program looks at your source code in two successive "passes," first compiling a table (called the "symbol table") in memory of all the labels and cross-referencing, and then using this table in the second pass to create the machine instructions for your program.

The assembler normally generates two files—one containing the combination of the machine-language code, in hex representation, called the "print file," and a file containing a hexadecimal representation of the final program, usually referred to as the "Intel hex" file. Three options are available to cancel the creation of either of these files, or to tell the assembler to send the print file to your monitor screen. These options are specified when you start the assembler:

$$\textbf{A}>\textbf{ASM AUTOST.}\underbrace{\textbf{123}}_{\textbf{three option specifiers}}\textbf{ RETURN}$$

Option 1, which tells CP/M where to find the source-code file, must be either **A** or **B** on the Osborne 1. For the example above, the source code should be on the diskette in the B drive.

Option 2 tells CP/M where to put the hex file: into **A**, **B**, or **Z**; the latter indicates that you do not want a hex file.

Option 3 tells CP/M where to put the print file. Again, you can specify drive **A** or **B**, or **Z** to skip generation of the print file. You can also use the letter **X** to tell CP/M to display the print file on the monitor.

For now, create both output files on drive B. Type:

A S M SPACE BAR A U T O S T . B B B RETURN

After a few minutes, you'll see the following message:

```
xxxx

yyyH USE FACTOR

END OF ASSEMBLY
```

If you see any lines above the ones just listed, you've made an error in entering the source code. The types of errors you might encounter are:

D Data Error. The value of the expression you indicated may be too long.

E Expression Error. The expression you indicated is incorrect or cannot be computed (too complex).

L Label Error. You used a label incorrectly. Occurs when you use a label inconsistently in a program.

N Not Implemented. You used an expression that is appropriate for MAC, a different assembler.

O Overflow Error. Your expression is too complicated.

P Phase Error. The value of a label changes between passes of the assembler.

R Register Error. You specified a register inappropriate to the mnemonic you used.

S Syntax Error. There is probably a typographical error in your source file.

U Undefined Symbol. You used a label in an expression without assigning a value to it.

V Value Error. The operand is incorrect.

The format in which error messages appear is:

```
U 0100 06 00 MVI B,IMPROPER
↑
└─error message
```

If you have any errors in your assembly, you need to go back to reedit your source file and correct them before you proceed. Assuming that the assembly reported no errors, you might be wondering what the two numbers in the **USE FACTOR** message mean. The first indicates in hexadecimal the first address unused by your program—that is, one byte greater than the last location your program used; while the second number, which can be as large as 0FF hex, indicates how large the symbol table is.

The directory of the diskette in drive B would reveal the following files:

AUTOST.ASM	your source file
AUTOST.HEX	the Intel hex file
AUTOST.PRN	the print file
and maybe	
AUTOST.BAK	a backup copy of the source file

However, you still don't have a working program. All that's stored in **AUTOST.HEX** is a series of hexadecimal numbers that can represent the final machine-language instructions. To use the program, you'll need to convert these hex numbers into instructions the Osborne 1 can execute by using the CP/M program **LOAD.COM**.

The LOAD program reads files in the Intel hex format and converts them into machine language that the Osborne 1 can execute. Using LOAD is simple:

L O A D SPACE BAR B : A U T O S T RETURN

If your assembly is correct, the right-hand disk drive will whirr and clack; then a message like the following will appear:

```
FIRST ADDRESS          0100
LAST ADDRESS           0456
BYTES READ             0357
RECORDS WRITTEN          04
```

This message indicates the pertinent details about your program: its length, the portion of memory it occupies, and how many 128-byte records are written on diskette. If you now look at the directory of the second drive, you'll find another file: **AUTOST.COM**. This is your executable program. Just type its name and the computer follows its instructions.

You don't always get ready-to-use assembly-language programs to copy from; sometimes you'll create your own programs from scratch. When you do so, your program may not work, and then you'll be interested in another utility program, **DDT**.

DDT stands for Dynamic Debugging Tool. It is a versatile program with which you can perform tasks in the "bowels" of your Osborne 1:

- load an assembled program into memory
- make changes in machine-language programs
- locate errors in assembled programs
- make simple corrections in programs
- examine or modify the contents of memory
- enter assembly-language code one line at a time
- disassemble an assembled program
- examine or modify the contents of CPU registers

- set breakpoints to stop program execution
- trace the execution of a program

The above list reveals that DDT is a program with many uses, and here are the two methods to invoke it:

DDT RETURN ← loads DDT only

DDT filename.type RETURN ← loads DDT and program

DDT identifies itself by presenting its version-number message and, if a program is loading, two other pieces of information:

```
DDT VERS 2.2

NEXT PC

#### ####

-
```

NEXT stands for the next available memory location, while **PC** indicates the setting of the Z80's "program counter." The numbers underneath these two headings are the current settings and are expressed as hexadecimal numbers.

DDT's prompt is a hyphen. Your cursor should be located just to the right of the prompt, indicating that DDT is waiting for instructions. DDT understands 14 different instructions:

> **A####** tells DDT that you want to give assembly-language instructions beginning at the hexadecimal address you indicate (**####**). As you type each instruction, DDT displays the next address at the left-hand side of the screen. You may use any valid assembly-language instruction exclusive of **LABEL**s or **COMMENT**s. Typing a period will return you to the DDT command level.
>
> **D####,####** displays the contents, in both hexadecimal and ASCII, of memory locations beginning with the first one you specify and ending with the last one.

If this display fills more than one full screen, you must use **^S** to temporarily cause the display to pause. Alternatively, you can omit the last address, in which case DDT will display the next 96 memory locations. Simply typing **D** will display the 96 memory locations that follow the last group you looked at.

F####,####,@@ fills the memory between, and including, the first two memory locations specified with the value specified (@@). The value must be a valid hexadecimal constant.

G####,#### begins execution of machine-language instructions at the first location with a "breakpoint" (halting point) at the second location specified. If the second location is omitted, no breakpoint is set. If both locations are omitted, execution starts with the current value of the Z80 program counter. By typing **G####,####,####** you can set two breakpoints. Typing **G,####** will start program execution at the current program counter with a single breakpoint.

H#,# computes the sum and difference of the two hexadecimal numbers you specify. The first number shown will be the sum; the second, the difference.

Ifilename.typ tells DDT and CP/M that the default file block should be filled with the information pertaining to the file name specified. Unfortunately, DDT does not allow you to specify the drive to be used, so you must make sure that the default disk drive is correct before you enter DDT.

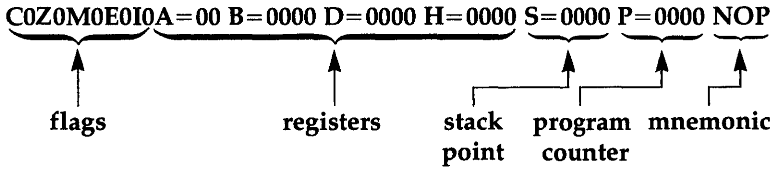
L####,#### lists (disassembles) the assembly-language program between the two addresses specified. If the second address is omitted, 11 instructions beginning at the first address will disassemble. If both addresses are omitted, 11 instructions beginning at the last location listed display.

M####,####,#### moves the block of memory between, and including, the first two addresses to the area of memory beginning at the third location.

R#### reads the file to which the current default file block points, using the number specified as an "offset" (displacement) factor. If no number is specified, the file reads into memory at its usual memory location—normally 0100 hex for most files.

S#### "sets" the memory at the location specified. DDT will display the memory location, a space, and the value currently stored there. You enter the new hexadecimal value or a period to conclude this command.

T#### "traces" the execution for the number of instructions specified. If no number is specified, one instruction is traced.



```
C0Z0M0E0I0A=00 B=0000 D=0000 H=0000 S=0000 P=0000 NOP
```

Trace Display

U#### is similar to trace, but it displays the beginning contents of the registers before executing the number of instructions specified. Unlike trace, the **U** command does not show what happens each time an instruction executes.

X displays the current contents of the registers.

X# displays the current contents of the register specified and allows you to change the contents.

replace # with	**C**	**to change**	"carry flag"
	Z		"zero flag"
	M		"minus (negative) flag"
	E		"even parity flag"
	I		"interdigit carry flag"
	A		"accumulator"
	B		"register pair BC"
	D		"register pair DE"
	H		"register pair HL"
	S		"stack pointer"
	P		"program counter"

If you need to know more about how DDT works, you've progressed well beyond beginner status. Beginners should use DDT only with diskettes they can easily replace. Be careful that you don't accidentally modify any portion of CP/M while using DDT, or your Osborne 1 may exhibit unpredictable results.

One last note: be careful about setting breakpoints on Z80 instructions or in routines in the middle of a bank-switching operation. Either alternative will result in a system crash.

When you exit from DDT, you may want to use the SAVE command to create a file with the modified file with which you were working. SAVE is simple to use; just type:

S A V E SPACE BAR # # filename.typ RETURN

where # # is the decimal number of 256-byte blocks you want CP/M to save.

Lesson 4: Inside CP/M

We're going further beyond beginners' material, but for the dedicated assembly-language programmer, here is a concise tour of the important aspects of CP/M.

CP/M resides in the top portion of the available memory, nominally from 0CB00 hex to 0F000 hex on the Osborne 1 with 1.4 ROM. The top and bottom addresses of CP/M are, of course, dependent on the location of CP/M itself which you may wish to move. In addition, the first 256 bytes of memory are called the "base page" and are reserved for certain CP/M functions and pointers. The area from 0100 hex to the beginning of CP/M is known as the transient program area (TPA), and is free for you to use as you wish.

Overall, CP/M consists of three parts:

CCP	console command processor
BDOS	basic disk-operating system
BIOS	basic input/output system

By default, CCP and BDOS reside between E100 and CB00, and BIOS between E100 and EA80. If you alter the size of CP/M, these addresses will be different. To find the beginning address after a MOVCPM or if you happen to have a different ROM than 1.4, use these assembly routines:

To find the starting address of BIOS—

```
LHLD     1       ;get warm start address

MVI      L,00    ;set low-order byte zero to obtain start
                   of BIOS

                 ;BIOS start is now in HL
```

To find the start of CCP—

```
MOV      A,H     ;get BIOS page boundary in A

SUI      16H     ;subtract offset to find start of CCP

MOV      H,A     ;CCP start is now in HL
```

To find start of BDOS—

```
LHLD      6         ;get BDOS entry point

MVI       L,00      ;set low-order byte to zero to obtain
                      BDOS start and top of memory
                      that can be used by programs
```

The CCP occupies the first 7FF hex locations of CP/M and interprets the commands you type when you see the A >. The CCP handles all the built-in commands and, if the CCP cannot find a command that matches what you type, it will look on the disk drive for a file with a matching name and a file type of **.COM** (command). If the CCP finds a command file, said file will load into the TPA, and the CCP will pass execution to location 0100 hex, the assumed starting place of your file.

BDOS takes care of all device handling for your programs and for CP/M. Thirty-eight functions are built into BDOS, all of which are available to assembly-language programmers. Both CCP and BDOS are exactly the same on every standard CP/M-based computer. A recent trend indicates that some manufacturers are modifying portions of CCP and BDOS, but Digital Research does not condone such activity, as it results in "nonstandard" CP/M systems.

BIOS is the section of CP/M that contains instructions specific to the Osborne 1. At the beginning of the BIOS section must be a series of 17 predefined "jumps" (called the "jump table") that tell CP/M where to go when it needs to communicate with certain parts of the Osborne—the keyboard, monitor, printer, and disk drives, for example. Unlike the rest of CP/M, the instructions in BIOS were programmed by Osborne Computer Corporation, and the company is responsible for their being correct. BDOS uses the instructions in BIOS, as needed, and relies on that jump table in order to find the correct instruction.

The "base page" (the first 256 bytes of memory) contains reserved memory locations that CP/M uses for various functions:

HEX LOCATION	**FUNCTION**
0000–0002	location to jump to for warm start
0003	IOBYTE, which tells CP/M which devices to use
0004	DISKBYTE, which tells CP/M which disk is active
0005–0007	BDOS entry vector, the entry to BDOS
0008–004F	reserved for hardware interrupt handling
005C–007C	File Control Block, used to define file being accessed
007D–007F	Random Record Position, which tells CP/M exactly where to go on diskette
0080–00FF	Disk Buffer, used to store 128 bytes of information temporarily before writing it to a diskette, or after reading it from a diskette

At the lowest level, when a program wants CP/M to perform a function, the program places a set of parameters and/or information into the CPU registers and then CALLs BDOS (0005 hex). When CP/M has completed the assigned task, it returns control to the calling routine (i.e., the program) with a new set of parameters and/or information in the CPU's registers.

For example, say you want to write a program that can detect if a user is typing on the keyboard. The BDOS function requires that you place a 0B hex in the C register; CP/M will return a value in the A register to tell you if the console is ready with a new character:

```
INPUT:  MVI    C,0Bh   ;BDOS function number
        CALL   0005h   ;CALL BDOS
        CPI    '00'    ;check if not ready
        JZ     INPUT   ;keep going back til
                       ;character available
```

A complete list of BDOS functions, with the parameters used, is listed in the Reference Guide.

The second area of CP/M available to programmers is the jump table in BIOS. It should be noted that the addresses in this BIOS jump table are dependent on the current size of CP/M. Each of the 17 jumps in the table references a different function on the Osborne 1. Here are the default jump addresses:

0E100	perform cold start
0E103	perform warm start
0E106	get console status
0E109	get console input
0E10C	display output on console
0E10F	display output on printer
0E112	display output on modem
0E115	get modem input
0E118	move head on disk to home position
0E11B	select disk drive
0E11E	select track on diskette
0E121	select sector on diskette
0E124	select DMA address
0E127	read from diskette
0E12A	write to diskette
0E12D	get printer status
0E130	translate to new sector

In addition, on the Osborne 1 we've extended this jump table with routines for the IEEE-488 interface; see Appendix 2 for complete details.

To get input from the console, simply insert the following instruction in your program:

```
CALL        0E109h  ;get console input
```

We strongly discourage such a practice, by the way. First, you should always use the BDOS functions to maintain full compatibility of your program with other CP/M-based machines.

Second, it is not good programming practice to make an absolute reference to a memory location that may move. If you later perform a **MOVCPM** command and create a different-size CP/M, the above instruction will no longer work. Many programmers skirt this possibility by noting that the location in 0001 and 0002 hex is the same as the second entry in the jump table for all versions of CP/M, and therefore calculates the position of a function in the table; Microsoft BASIC does this, as do other common CP/M programs.

Lesson 5: SUBMIT and XSUB

For the most part, every time you use your Osborne 1, you will "interact" with it: you type something, the Osborne 1 responds, you type something, the Osborne 1 responds . . .

Computing has not always been so interactive, nor is every task you may want to use your computer for an interactive one. The alternative is something called "batch processing," which groups a batch of commands—in a diskette file in CP/M. The computer then performs them in succession, without human intervention.

Two programs provided on your CP/M Utility diskette allow you to perform batch processing on your Osborne 1: **SUBMIT** and **XSUB**.

SUBMIT reads a file you name that contains commands CP/M recognizes. Be careful here: SUBMIT works only with the type of commands allowed when you see the CP/M A >, not any other command or from any drive except A. To create a file SUBMIT can use, employ the WordStar **N** option and give your file the **.SUB** type. You would type one command on each line, as in the following:

```
DIR *.COM
XDIR *.COM
STAT *.COM
```

Save the file and use SUBMIT as follows:

S U B M I T SPACE BAR filename RETURN ← don't type .SUB in the filename

What happens next is that CP/M builds a file of your commands —in reverse order—with the name **$$$.SUB**, and then it begins reading that file and executing the last command in it. A nifty programming trick continually makes SUBMIT think that one less command is in the file, and execution proceeds until SUBMIT thinks the file is empty. In short, with the above file, you'll see the three commands you typed execute in order. When SUBMIT is done, it will erase the temporary file from your diskette and return control to you.

NOTE

*By prefacing a line with a semicolon, you can include "comments" in your **SUBMIT** file.*

SUBMIT has even more uses; two features add to its abilities. First, you can leave special "variables" in your **.SUB** file that will fill when you actually invoke SUBMIT. Anywhere in your file you wish to have a variable, use a dollar sign preceding a number—**$1** would be the first variable, for example. When you're ready to use SUBMIT, you type the values for each variable—separated by a space—following the SUBMIT command. For instance, the following **.SUB** file and SUBMIT command will produce the indicated results:

.SUB file:

```
;There is no SUBstitute for quality!
XDIR $1.$2
;See what we mean?
```

command: S U B M I T SPACE BAR
filename * SPACE BAR C O M RETURN

results:

```
A > ;There is no SUBstitute for quality!

A > XDIR * .COM

Extended Directory version 3.5

SUBMIT .COM 2K

XDIR .COM 4K

XSUB .COM 4K

Disk A: 2K blocks

Size =   92K, 13 files, Used =   92K,

Space =   0K

A >  ;See what we mean?

A >
```

The second feature you may want to use is the program named **XSUB**. Remember, SUBMIT works only at the CP/M command level—you can't type in instructions once a program, such as DDT, has been loaded by SUBMIT. XSUB allows you to do so.

If you needed to type **S0003 RETURN 00 RETURN . RETURN** after invoking DDT, you would need to have the following lines in your **.SUB** file:

```
XSUB
DDT
S0003
00
```

As part of your **.SUB** file, make sure that a line containing the command XSUB is present at the point before you wish to have individual characters "submitted" to a program.

Eight Down, One to Go

Well, you've made it successfully, we hope, through eight chapters. So far, everything we've described is related purely to the software (programs) that accompany your Osborne 1. In the next chapter, we'll turn our attention to the hardware (equipment) that comprises the computer.

CHAPTER 9

System Specifications

This chapter examines the "insides" of the Osborne 1 and how to use some of the built-in hardware functions of the machine. If you're not familiar with assembly-language programming or aren't interested in programming at all, it's probably best if you skip this chapter. This revision of chapter 9 is specific to the 1.4 monitor ROM currently being shipped with the Osborne 1 computer.

The Osborne 1 is a complete computer—you cannot buy a version that does not have disk drives and a monitor.

We ship the Osborne 1 with the following software:

CP/M 2.2
WordStar 2.26
SuperCalc 1.12
Microsoft BASIC 5.21
CBASIC 2.37

All of the above software is the standard version the primary vendor supplies us. WordStar and SuperCalc have terminal and printer drivers modified to work with the Osborne 1, but we supply the **INSTALL** programs for both, should you wish to change our I/O (input/output) programming.

The hardware of the Osborne 1 is of the "plain-vanilla" variety:

Z80A-type central processor
64K dynamic RAM memory
built-in 5-inch video display monitor
dual double-density disk drives
serial RS-232 port/modem port
IEEE-488 port
external video port
battery-pack option
full uppercase/lowercase keyboard
auto-dial, auto-answer modem option

Following are the most important nitty-gritty details most programmers want to know about the Osborne 1 specifications:

SCREEN SIZE:

- 32 lines of 128 characters maintained in RAM
- 24 lines of 52 characters shown on screen
- dim, normal, underlined video supported

- 32 block-graphic characters predefined
- uppercase/lowercase text display
- video emulates TeleVideo terminal
- external video available via edge connector

DISK CAPACITY: Double-Density:

- 200K bytes per diskette
- 185K bytes of data space using CP/M
- 40 tracks of information
- 5 physical sectors each track (soft-sectored)
- 1024 bytes per sector
- 40 logical sectors to CP/M (128 bytes each)
- 1K-byte extents maintained by CP/M
- 3 reserved system tracks

Single-Density (old style):

- 100K bytes per diskette
- 92K bytes of data space using CP/M
- 40 tracks of information
- 10 physical sectors each track (soft-sectored)
- 256 bytes per sector
- 20 logical sectors to CP/M (128 bytes each)
- 2K-byte extents maintained by CP/M
- 3 reserved system tracks (See notes, page 760.)

SERIAL PORT:

- 1200- or 300-baud, software-selectable

- 2400- or 600-baud, jumper-selectable
- uses 6850 chip, all parameters memory-mapped
- standard female DB-25 connector provided

IEEE-488 PORT:

- standard IEEE-488 implementation
- may be configured as Centronics parallel port
- 26-pin edge connector provided

In this chapter, we'll abandon our tutorial instruction and describe each "module" of the Osborne 1 design individually, with a particular emphasis on how to use software to access or change parameters.

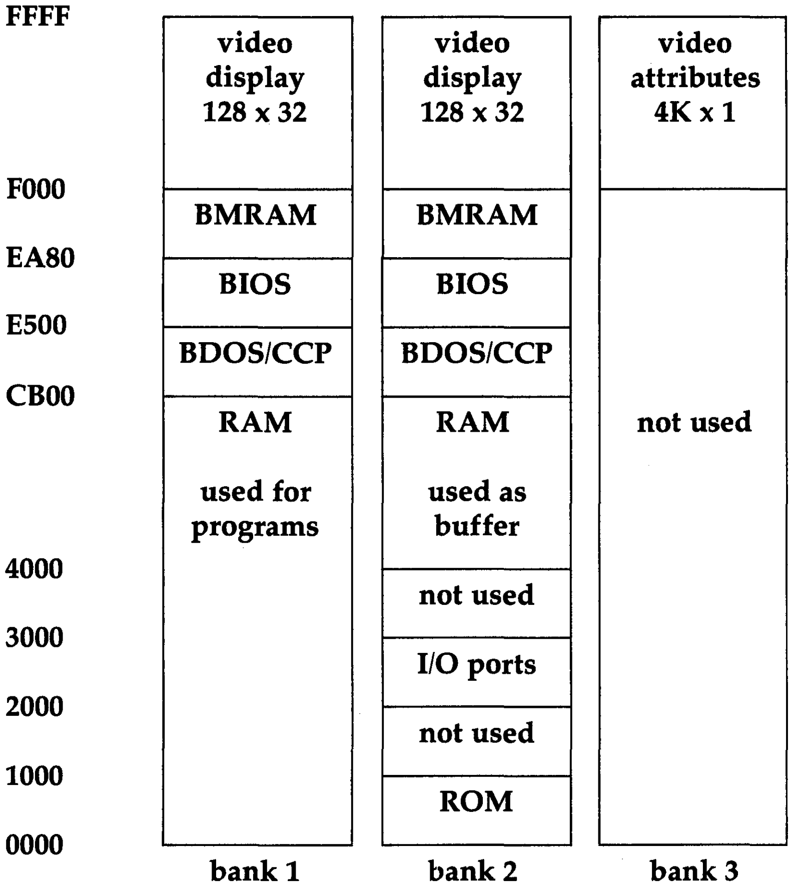
Memory Layout

The memory layout of the Osborne 1 is unique. Three basic components of system memory exist:

bank 1	64K dynamic RAM memory
bank 2	4K system ROM, memory-mapped I/O devices
bank 3	4K video-display-attribute dynamic RAM

While we refer to the three components as "banks" of memory, they do not perform exactly as do the memory banks used in most S-100-based computers. Bank two, for instance, is a combination of system ROM and memory-mapped I/O ports, with the remainder of the 64K address space filled by "mimicking" the first bank (you may thus execute the same program from memory above 4000 hex in both banks).

The memory map on the following page is a closer look at the way the Osborne 1 uses memory.



Bank 1 of memory is the one you normally use for programming. CP/M, for instance, loads itself into the uppermost free area—just below the video display memory—with the Basic Input Output System (BIOS) beginning at E100 hex. The Basic Disk Operating System (BDOS) and the Console Command Processor (CCP) take up the memory from CB00 hex to the beginning of BIOS. As with all CP/M systems, the memory area from 0000 hex to 0100—commonly referred to as "page 1" of memory —is reserved for use by CP/M. Overall, about 51K bytes of usable memory are available to the programmer when CP/M is resident (a 59K CP/M system like the Osborne 1's means that, including CP/M, 59K of memory is used).

Bank 2 of memory contains the Osborne 1's basic intelligence. When the power is turned ON, the ROM located at the bottom of bank 2 initializes the system, and some parameter passing information stored in the area labeled **BMRAM** in the memory layout we just presented, then presents the "start-up" display on the video monitor. The ROM is contained in a single 2732 memory chip. There is a slight difference between 1.3 ROMs and the 1.4 ROM presently being shipped with Osborne 1 computers. (A technical manual describing the Osborne 1 hardware in detail is now available from your dealer.)

Briefly, the following is what you'd find in the Osborne 1 ROM if you looked at its source code:

- disk-boot routines
- I/O drivers
- video-display drivers
- disk drivers
- keyboard drivers and N-key rollover routine
- parallel port drivers

Earlier versions of the ROM (REVA and REV 1.2) also contained some simple diagnostic routines and a real-time clock routine. These were removed in the 1.3 ROM to make room for additional routines and because the diagnostic and clock code can be duplicated easily in RAM.

Think of the system ROM as an extension of the BIOS routines used in CP/M. If you disassembled the BIOS supplied with the Osborne 1, you would find more references to the ROM than you'd normally encounter in many computer systems. Indeed, a section of memory from 0EA80 hex to 0F000 hex—called BMRAM—is reserved so that the ROM and BIOS may freely pass information back and forth.

The advantage of the Osborne memory scheme is that we are able to offer a 59K CP/M system while still providing the benefits and speed of memory-mapped video. Most other CP/M systems that use memory-mapped video have only a 56K CP/M system; some have even less memory available.

The disadvantage of the way we've laid out the system memory is that you'll have to learn a new way to get from one place to another. Systems that contain only a single memory bank merely require you to supply a memory address when you refer to a location. With the Osborne 1, you must make sure that you're addressing the proper bank of memory before you supply an address.

We'll tell you, next, how to accomplish the switch between memory banks. The first bank-switching technique for microcomputers was pioneered by Cromemco. It requires you to send a special message to I/O port 0. The Osborne 1 bank-switching technique is also simple. To go from bank 1 to bank 2, you must

a. disable the interrupts,

b. output to I/O port 0, and

c. store a 00 hex in memory location EF08 hex.

This bank-switching routine looks like this in assembly language:

```
                    FLAGPOS     EQU     0EF08h
F3                  SWITCH:     DI
D3    00                        OUT     00h
3E    00                        MVI     A,00h
32    08    EF                  STA     FLAGPOS
```

Simple, right? Now let's see what you have to do to get back to memory bank 1:

a. output to I/O port 1,

b. store a 01 hex in memory location EF08 hex, and

c. enable interrupts.

In assembly language, that procedure looks like this:

```
                        FLAGPOS     EQU     0EF08h
D3    01                GOBACK:     OUT     01h
3E    01                            MVI     A,01h
32    08    EF                      STA     FLAGPOS
FB                                  EI
```

There's only one catch to all of this switching between banks of memory: you are limited to what you can do in memory located below 4000 hex.

Look at the memory map again. You'll note that in bank 2 below 4000 hex there is no RAM memory, only ROM and I/O ports. Therefore, the following restrictions apply to switching between banks:

1. Your switch routine must be in bank 1 at a memory location above 4000 hex.
2. You cannot have any executable code (other than the system ROM) in bank 2 located below 4000 hex (i.e., when you are using bank 2, all executable code must be at locations 4000 hex and above).
3. If you are operating in CP/M, you must be extremely careful to set up a buffer in bank 1 that corresponds to any memory location in bank 2 you may wish to change.

Of these restrictions the third is the most subtle and likely to cause you problems. Memory from 4000 hex to FFFF hex in bank 2 is actually the memory in bank 1! We call this mimicking "the shadow mode" because the memory is actually shadowing the primary bank. When you're operating in bank-2 memory, you can change anything you want above location 4000 hex, but remember that when you get back to bank 1, its memory will reflect any changes you make.

There is an easy way around this "shadowing" problem. Simply use MOVCPM to create a smaller CP/M system. If you need 4K of memory space in bank 2, create a 55K CP/M system, and you'll be able to use memory locations 0E000 to 0EF70 hex without worrying about "crashing" any application program running under CP/M in bank 1.

A close study of the way we implemented CP/M on the Osborne 1 shows you that we use areas within BIOS as data buffers and temporary storage areas. In essence, we've already moved CP/M down in memory just a smidgen to make room for our disk buffers, so don't be afraid to create your own buffer room above CP/M.

We have more on the memory map of the Osborne 1, but it properly belongs in a discussion of the video display, which follows, and the I/O ports. If you need to know more about memory addressing, skip ahead and look at those sections.

Direct Screen Manipulations

You've already learned that the video display is memory-mapped using RAM memory beginning at 0F000 hex. The video memory is a 128 x 32 matrix—128 characters on each of 32 lines.

Now wait a minute, you say—the Osborne's monitor displays 52 characters on each of 24 lines. What gives?

The display you see is actually a "window" on the larger video matrix, so you can select any window of the larger matrix to display. If you want to put the character in position 52 of line 12 in the upper left-hand corner of the screen, do so by telling the computer to issue the following sequence of characters:

<ESCAPE> S yposition xposition

Here, **yposition** is the line number you want in the upper-left corner and **xposition** is the column number. You have to add an "offset " of 32 to both numbers. In BASIC, you might code that sequence like this:

```
10 PRINT CHR$(27)+"S"+CHR$(YPOS+31)+CHR$(XPOS+31);
```

Alternatively, you can place your desired location in the upper-left corner in the HL register and use an assembly-language routine like the one that follows to reposition the screen:

```
         BDOS    EQU 0005h
         ESC     EQU 27
         PLACE   EQU yyxx    ;you fill in
SETSCR:  LXI     H,PLACE
         MVI     E,'S'       ;set screen function
         PUSH    H           ;save YX
         PUSH    D           ;save setscr function
         MVI     E,ESC
         CALL    OUTCH       ;send ESCAPE
         POP     D           ;restore setscr
         CALL    OUTCH       ;send SETSCR
         LXI     D,2020      ;bias for YX
         POP     H           ;restore position
         DAD     D           ;add bias to position
         PUSH    H           ;save xposition
         MOV     E,H         ;prepare yposition
         CALL    OUTCH       ;send yposition
         POP     D           ;prepare xposition
         CALL    OUTCH       ;send xposition
OUTCH:   MVI     C,2         ;CP/M write to
                               console function
         JMP     BDOS        ;call CP/M
```

To move the cursor—as opposed to the screen—use the same routine as listed above, with the one exception that the sequence of characters now becomes:

<ESCAPE> = yposition xposition

Again, offsets of 32 decimal to the X and Y positions are required for proper positioning to occur. Only one byte need be changed in the above assembly-language routine to set cursor position instead of screen position: **MVI E,'='** instead of **MVI E,'S'** —thus, with very little reworking, you can create an assembly-language routine that sets either the screen or the cursor position.

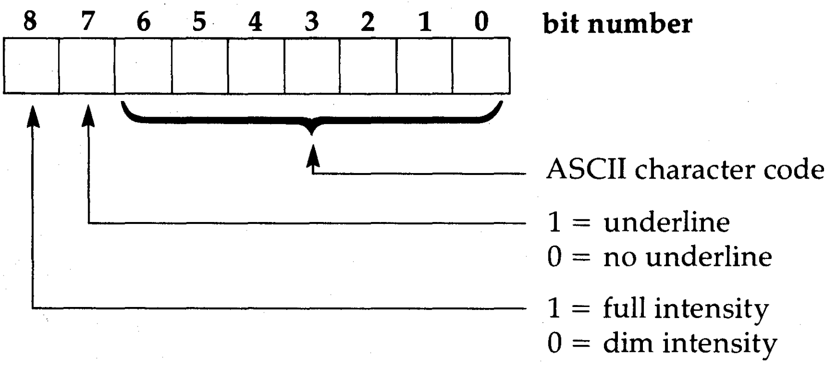
You can control many other video attributes by having the computer issue a sequence of characters. Here is a complete list of the screen controls:

HEX SEQUENCE	ACTION
07	rings the bell
08	moves the cursor left one position (no erasure)
0A	moves the cursor down one line
0B	moves the cursor up one line
0C	moves the cursor right one position
0D	performs a carriage return (no line feed)
1A	clears the screen and homes the cursor
1E	homes the cursor
1B 29	begins half-intensity video display (dim)
1B 28	ends half-intensity video display (bright)
1B 45	inserts a line at cursor position
1B 67	starts graphic character interpretation
1B 47	ends graphic character interpretation
1B 6C	starts underlining all characters
1B 6D	ends underlining of characters
1B 51	inserts a character at cursor position
1B 52	deletes a line at cursor position
1B 54	deletes from cursor to end of line
1B 57	deletes a character at current cursor position

To clear the screen in BASIC, therefore, a **PRINT CHR$(&H1A);** is all you need. In assembly language, simply place a 1A hex in the E register and use **CALL OUTCH** as listed in the last assembly-language-program example. When accessing graphic characters from within BASIC, use an offset of 96 decimal in order to get all of the characters available.

A close look at the above sequences will show that, for the most part, the Osborne 1 "emulates" the TeleVideo 920C terminal's video functions. So, if you have a program that works correctly with the TeleVideo terminal, it should also work correctly with the Osborne 1.

After so much "standard" fare in the video area, it may come as a shock to discover that the Osborne 1 really thinks of the video display as a 128 x 32 matrix of 9-bit memory locations. The layout of these 9-bit locations looks like this:



This lets you manipulate individual memory locations directly, without the output sequences: 1BH,29H or 1BH,28H.

Of course, the memory isn't actually 9 bits wide. It's just made to look that way by using a BIOS call to the ROM's "dim" routine (STODIM).

To work properly, the CPU's D&E registers must hold the STODIM address (015DH) when calling. Also, the STODIM routine requires that the address of the pertinent screen location be in the H&L registers, and that the contents of that location be in the B register. Then you can "AND" the location with 7FH for dim, or "OR" the location with 80H for bright. Finally, for the actual addressing: if your Osborne 1 is double density, its "ROM resident" call address is 0E136H; if it's single density, the address may be at 0E536H.

One last area to examine during discussion of the video display is the character set. Each character on the Osborne 1 is defined as a dot matrix of eight columns wide by ten rows high. 128 characters are possible—underlining creates another 128, but the matrix used is the same except for the bottom row.

The character-generator ROM on the Osborne 1 uses the first 128 characters of its memory to define what the top row of each character looks like, the next 128 characters in ROM define the second row of each character, etc.

You should note that the graphics character set occupies the first 32 ASCII codes. An offset of 60 hex (96 Decimal) should be used when accessing the graphics, otherwise there might be a problem. Assuming that you have told the Osborne 1 to switch to graphics presentation, the control functions will no longer work correctly (RETURN will show up as a white circle). Be careful here. If you must use cursor positioning with graphics, you'll either have to do so using direct memory manipulations, or you'll have to drop out of the graphics mode every time you want to reposition the cursor. This problem poses little trouble in assembly language, but if you're programming in a high-level language, expect this switching back and forth for graphics to slow down the speed at which you can display things—sorry, folks, games are going to be difficult to program in BASIC.

The Ports

Experienced CP/M programmers are apt to be a bit confused when they start studying the keyboard, serial, modem, and IEEE ports. Since CP/M runs only on the 8080 family of central processors, most CP/M-based machines use the 8080 family of support chips to provide I/O ports. Many popular CP/M machines use an 8251 chip to provide a serial interface, for instance.

In the 8080 (and the Z80 and 8086) family of support chips, the usual method of "programming" a port is to use an OUT instruction to send something to the port, and an IN instruction to receive something from it. The Osborne 1 is different. In designing it, we chose to use support chips from the 6800 family of central processors. In particular, a 6850 chip is used for the serial port and a 6821 is used to address the IEEE-488 port.

These support chips use memory-mapped I/O, as opposed to the port-addressed I/O typical in CP/M systems. Thus, to send something to a port, you must send bytes of information to memory addresses; however, you read bytes from the memory addresses to receive something from the port. Before we go too far with our explanation, let's look at the memory map for the I/O ports; all of these locations are in bank 2:

2FFF		
	xxxxxxxxxxxxxx	**xxxxxx = not used**
	xxxxxxxxxxxxxx	
2C03	**part b control**	
2C02	**part b data**	**video**
2C01	**part a control**	
2C00	**part a data**	
	xxxxxxxxxxxxxx	
2A01	**buffer**	
2A00	**status/control**	**serial**
	xxxxxxxxxxxxxx	
2903	**part b control**	
2902	**part b data**	**IEEE**
2901	**part a control**	
2900	**part a data**	
	xxxxxxxxxxxxxx	

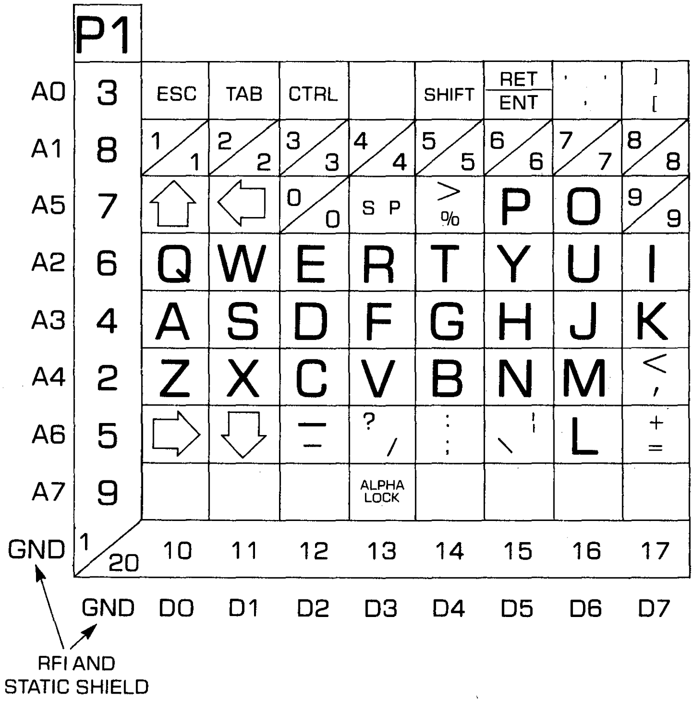
2280	row 7	
2240	row 6	keyboard
2220	row 5	
2210	row 4	
2208	row 3	
2204	row 2	
2202	row 1	
2201	row 0	
	xxxxxxxxxxxxxx	
2103	data	
2102	sector	disk
2101	track	
2100	status	
	xxxxxxxxxxxxxx	
	xxxxxxxxxxxxxx	
2000		

To fully understand how these memory locations work, you need specification sheets for the 6850 ACIA and 6821 PIA chips, and one for the Western Digital 1793 floppy-disk controller. The specification sheets will define what each bit does in each memory location. Remember, many of the locations perform dual functions: read and write, for example.

The Keyboard

If you look at the construction of the Osborne 1 keyboard, you quickly discover that it has no decoding logic.

When an Osborne 1 user presses a key, that key has a unique location in an 8 x 8 matrix:



	P1								
A0	3	ESC	TAB	CTRL		SHIFT	RET ENT	' ' ,	] [
A1	8	1/1	2/2	3/3	4/4	5/5	6/6	7/7	8/8
A5	7	⇧	⇦	0/0	S P	> %	P	O	9/9
A2	6	Q	W	E	R	T	Y	U	I
A3	4	A	S	D	F	G	H	J	K
A4	2	Z	X	C	V	B	N	M	< ,
A6	5	⇨	⇩	— –	? /	: ;	¦ \	L	+ =
A7	9				ALPHA LOCK				
GND	1/20	10	11	12	13	14	15	16	17
	GND	D0	D1	D2	D3	D4	D5	D6	D7

If you remember the memory map presented earlier, you'll immediately associate the rows listed in the above matrix with the memory locations set aside for keyboard I/O. The system ROM in the Osborne 1 continually checks to see if a key has been pressed. The logic in that routine allowed us to implement three-key rollover and a set of ten quasi-function keys.

The function keys are control-0 through control-9, and are user-programmable. In BIOS you'll find two tables associated with the function keys. The first is a table of pointers to the function definitions, which by default is located at 0E16B hex. The other is a 76-character table, located at 0E192 hex, which maintains the character sequence associated with a function. For the most part, you'll program these keys using the SETUP program supplied with the Osborne 1. Since the table is in RAM, you can change the table from within your program.

You may wonder if you can reinterpret the keyboard characters as you depress them. Only programs executing in bank 2, like the system ROM, can access the keyboard. You can replace our BIOS call for input with your own, using a translation table to substitute keystrokes. Because of the three-key rollover in ROM, we do not suggest that you replace the ROM routine with your own.

The Modem and RS-232 Interfaces

A close look at the circuitry in the Osborne 1 will show you that the modem and RS-232 interfaces are basically one and the same; the primary differences being that many of the signal levels on the RS-232 connector are held constant, while you can manipulate them using the modem connector. Also, the (RTS) on the RS-232 serves as a Transmit enable/disable while the modem uses this "request to send" signal in the conventional manner.

The IOBYTE is a reserved memory location (0003 hex) which defines the current assigment of physical to logical devices. The IOBYTE is divided into four distinct fields for the four logical devices recognized by CP/M. The logical device names are now obsolete (do not match Osborne 1 physical device names) and are used only for the sake of maintaining the standard CP/M nomenclature.

The Osborne BIOS fully implements the IOBYTE function of CP/M. If all you need to do is read to and from the modem and/or serial port, simply change the IOBYTE indirectly by using the CP/M STAT command, or directly by changing the byte at location 0003 hex. The IOBYTE is split into four sections, one each for the console, punch, reader, and list devices. Each device is represented by two bits, as follows:

```
7       6 5       4 3       2 1       0  bits
 \     /   \     /   \     /   \     /
 printer    punch    reader    console
```

The table of possible values for each of these 2-bit indicators is as follows:

CONSOLE field (bits 1 and 0)	**00** = keyboard + screen (TTY:)
	01 = serial port (CRT:)
	10 = parallel port (BAT:)
	11 = IEEE-488 port (UC1:)
READER field (bits 3 and 2)	**00** = keyboard + screen (TTY:)
	01 = serial port (PTR:)
	10 = parallel port (UR1:)
	11 = IEEE-488 port (UR2:)
PUNCH field (bits 5 and 4)	**00** = keyboard + screen (TTY:)
	01 = serial port (PTP:)
	10 = parallel port (UP1:)
	11 = IEEE-488 port (UP2:)
LIST field (bits 7 and 6)	**00** = keyboard + screen (TTY:)
	01 = serial port (CRT:)
	10 = parallel port (LPT:)
	11 = IEEE-488 port (UL1:)

(Note: the values in parentheses are used with the STAT command to change the IOBYTE to the value indicated.)

Thus, to make the serial port the new CP/M console, you'd change the IOBYTE (memory location 3) from 80 hex to 81 hex. However, the easiest way to accomplish this type of logical-to-physical assignment is using STAT.

The IOBYTE

The IOBYTE is a reserved memory location (0003 hex) which defines the current assignment of physical to logical devices. The IOBYTE is divided into four distinct fields for the four logical devices recognized by CP/M. The logical device names are now obsolete (i.e., do not match Osborne physical device names) and are used only for the sake of maintaining the standard CP/M nomenclature.

7	6	5	4	3	2	1	0	bit number
list		punch		reader		console		

A value in the range 00 to 3 hex (0 to 3 decimal) determines the assignment of each logical device as follows:

Logical Device	Value	CP/M Physical	Osborne Physical
List:	00	TTY:	keyboard + screen
(LST:)	01	CRT:	serial port
	10	LPT:	parallel
	11	UL1:	IEEE port
Reader:	00	TTY:	keyboard + screen
(RDR:)	01	PTR:	serial port
	10	UR1:	parallel port
	11	UR2:	IEEE port
Punch:	00	TTY:	keyboard + screen
(PUN:)	01	PTP:	serial port
	10	UP1:	parallel port
	11	UP2:	IEEE port

Console:	00	TTY:	keyboard + screen
(CON:)	01	CRT:	serial port
	10	BAT:	parallel (Centronics)
	11	UC1:	IEEE port

You can determine the output status of both the modem and serial ports by using the BIOS call LISTST located at 0E12D hex. A value of 0FF hex indicates that the list device is ready; 00 indicates busy. To find the input status of the modem or serial ports, you must first switch to bank 2 of memory and then look at memory location 2A00 hex. To change the status of the modem or serial device directly, you use the same memory location and write a special "control" byte as dictated in the 6850 specification sheet (see the technical manual for more details). Memory location 2A01 hex in bank 2 is the data buffer: you read information from external devices by moving the byte to one of the Z80 internal registers, you send information to the external device by moving data from the Z80 register to the memory location.

The pin connections on the modem port are as follows:

1	**GND**	signal ground
2	**TXD**	transmitted data—TTL logic, 1=high
3	**not used**	
4	**MSB**	modem status bit—open collector, 50μa sink=inactive
5	**CTS**	clear to send
6	**RXD**	receive data
7	**+12v**	connected to power supply through 22 ohms
8	**MCB**	modem control bit—TTL, low suppresses output
9	**RI**	ring indicator—TTL, high-to-low sets flag

Osborne has an optional modem. Connecting other modems without using some sort of external adapter may damage your

Osborne 1, as pins 4 and 5 are open collectors and are sensitive to signal-edge transitions. If pin 4 is not connected to the modem, make sure that nothing is connected to pin 4 at the Osborne end; otherwise, adjacent signals may be received inadvertently.

Below are the pin assignments for the RS-232 connector from the printer point of view:

1	**AA**	frame ground (optional)
2	**BA**	transmitted data (low=1)
3	**BB**	received data (low=1)
4	**CA**	request to send (high or no connection enables)
5	**CB**	clear to send (always high on OCC 1)
6	**CC**	data set ready (always high on OCC 1)
7	**AB**	signal ground
8	**CF**	received line signal detected (always high)
20	**CD**	data terminal ready (high or connection enables)

Baud rates for the serial port are 300 and 1200, software-selectable (use the SETUP program to permanently change the baud rate from 1200 that BIOS assumes). If it's necessary, you can change an internal jumper to double the baud rate.

The Disk Interface

You can address the disk interface directly, although we strongly discourage this practice. If you must control the disk drives directly, do so through the standard CP/M BIOS calls:

0E118	move head to track 0 on selected drive
0E11B	select disk-drive number
0E11E	set track number
0E121	set sector number
0E124	set the DMA address

0E127	read the selected sector
0E12A	write the selected sector
0E130	translate the sector

Each of these routines is documented in the Osborne 1 technical manual should you need to use them. If you're familiar with the way CP/M handles disk I/O, you also know that "BDOS functions" are accessible through memory location 0005 hex. Also note that the default BIOS addresses shown in this manual are affected by the currently implemented size of CP/M. The Osborne 1 is a standard CP/M system, and thus works as documented in such books as *The Osborne CP/M User Guide*.

If you're curious about how information is stored on the diskette, here are the bare facts:

40	tracks
5	physical sectors
1024	bytes per sector
3	tracks reserved for operating system
200	total diskette capacity
64	number of directory entries allowed
1K	smallest addressable disk space

Track 0 contains the CCP portion of CP/M, along with BDOS. Track 1 contains all of BIOS, and track 2 is reserved for future use.

The directory is the first data track, track 3. Each entry consists of the standard CP/M format: one byte to indicate deletion, 11 bytes for the file name, and 20 bytes representing the "groups" assigned to the file.

There are two unusual aspects of the Osborne 1's use of the disk system. First, even though information is stored on the diskette in 5 sectors, to CP/M there are 40 sectors of 128 bytes each on the diskette. In other words, if you are using the Osborne 1 ROM routines, as documented in the technical manual, you'll be working with 5 physical sectors of 1024 bytes, but if you're working with CP/M BIOS or BDOS routines, you'll be dealing with 40 logical sectors of 128 bytes each.

Also, the Osborne 1 does not allow transfers of data directly to memory in the first 16K of memory space because the ROM and I/O in the second bank reside there. Instead, transfer information involving the first 16K of memory by first buffering the information in high memory (above BIOS) and then moving it into position. The opposite procedure occurs when you write information to the diskette from the initial 16K of memory. Use of the Z80 block-memory-move instruction makes this buffering transparent to users, and almost completely cancels any speed penalty involved.

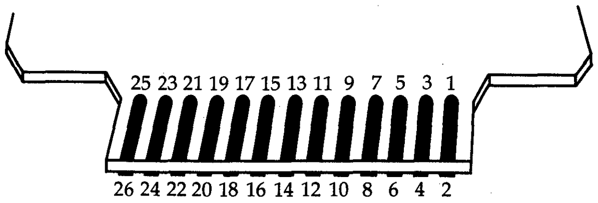
The IEEE-488 Interface

Any IEEE-488-compatible device can connect to the Osborne 1 by the IEEE connector. Because this connector is used for more than just IEEE-488 signals, we've declined to use a standard IEEE connector. The following table shows the pin assignments for both the IEEE standard connector and the Osborne 1 edge connector:

IEEE	OSBORNE 1	SIGNAL NAME	
1	1	Data bit 1	(DIO1)
2	3	Data bit 2	(DIO2)
3	5	Data bit 3	(DIO3)
4	7	Data bit 4	(DIO4)
5	9	End or Identify	(EOI)
6	11	Data Valid	(DAV)
7	13	Not Ready for Data	(NRFD)
8	15	No Data Accepted	(NDAC)
9	17	Interface Clear	(IFC)
10	19	Service Request	(SRQ)
11	21	Attention	(ATN)
12	23	Cable Shield + GND	(SHIELD)
13	2	Data bit 5	(DIO5)

14	**4**	Data bit 6	(DIO6)
15	**6**	Data bit 7	(DIO7)
16	**8**	Data bit 8	(DIO8)
17	**10**	Remote Enable	(REN)
18	**12**	Signal Ground	(DAV)
19	**14**	Signal Ground	(NRFD)
20	**16**	Signal Ground	(NDAC)
21	**18**	Signal Ground	(IFC)
22	**20**	Signal Ground	(SRQ)
23	**22**	Signal Ground	(ATN)
24	**24**	Signal Ground	(Logic)

Note: The pin connections on the Osborne 1 are counted from the top left to the lower right:



Hewlett-Packard computers and Commodore's PET and CBM use the IEEE-488 interface to hook peripherals to the computer, so peripherals designed for these computers will work with the Osborne 1. Create a cable with the pin connectors indicated above; then run the **SETUP** program to configure the IOBYTE properly. Printers which are other than device number 0 on the IEEE bus will need some extra manipulation; one byte in the BIOS memory will need to be changed. You can specify any valid IEEE device number from within the SETUP program.

We've taken the headache out of much of the other programming you'll need to do for the IEEE-488 port, as well. The eight primary commands are included as jumps appended to the end of the CP/M BIOS jump table. You must employ them in an intelligent sequence to properly use the IEEE interface. The CP/M BIOS entry and exit states for the following commands are defined in Appendix 2.

The IEEE commands are:

Control out
Status in
Go to Standby
Take Control
Output Interface Message
Output Device Message
Input Device Message
Input Parallel Poll Message

IEEE-488 commands use no RAM other than the stack. Each command routine in BIOS determines status of the port by reading the status of the 6821 PIA chip. The PIA transmits signals in both directions, so to reduce the overhead in determining the current direction the PIA is attempting to communicate, it is always left in one of two modes:

the source handshake mode
the accepter handshake mode

(The PIA specification sheet will be helpful in determining these modes.)

Several of the IEEE commands require that the PIA be in the source handshake mode when called. The PIA is normally in the source handshake mode following the completion of any IEEE-bus information transfer, so this is not a major restriction. For instance, both the Status In and the Parallel Poll commands require that the PIA be in the source mode, which means that you can perform the detection-of-device requests using either serial poll or parallel poll only when the interface is idle.

To send data to a device on the IEEE bus, the controller makes the device a LISTENER, assumes the role of TALKER, and sends the data. To receive data from an external device, the controller must first make the device a TALKER and then assume the role of LISTENER. After this, the controller goes on "standby" and allows the two devices to communicate at their own rate.

The controller can regain control asynchronously by setting the ATN signal to true. But if a device-dependent message is true when ATN becomes true, other devices on the IEEE bus can misinterpret the interrupted byte as an interface message and produce chaos. Avoid the problem by taking control synchronously. If high-speed transfer of data between devices is not required and the computer can be tied up during the transfer, it is better to make the controller listen to the transfer while discarding the data. This procedure allows the controller to count transfers, look for EOI signals, or "time out" the TALKER before regaining control.

The IEEE commands are detailed in Appendix 2, with sample programs included for help in deciphering how we've put the BIOS jumps into effect for the IEEE bus. Within three months of introduction of the Osborne 1, users figured out methods of connecting hard-disk drives, printers, modems, and other popular devices using the IEEE interface.

The Centronics Interface

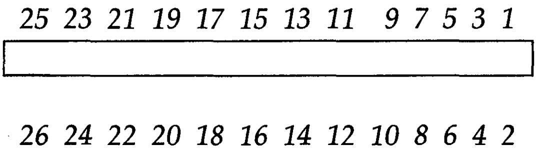
The Osborne 1 does not have a Centronics interface, per se, but we have done some clever programming via the IEEE interface to mimick Centronics' protocol.

To hook a Centronics printer to your Osborne 1 you must create a cable with the following pin connections:

Osborne IEEE Edge Connector		**Centronics Connector**	
pin 1	data 0	2	
2	data 4	6	
3	data 1	3	
4	data 5	7	
5	data 2	4	
6	data 6	8	
7	data 3	5	
8	data 7	9	
10	ground	30	
11	out strobe	1	
12	ground	19	
15	busy	11	
16	ground	29	
17	input ready	14	(not necessary on Epson)
18	ground	21	
19	in strobe	13	
20	ground	20	

NOTE

The pin connections on the Osborne IEEE-488 port are numbered in the following manner as you look at the front of the Osborne 1:



Generally speaking, the above signal definitions apply to most parallel printers, although the connector on the printer end may be entirely different—for instance, the IDS Paper Tiger uses a DB25 connector. One area which has provided for much confusion is the Centronics protocol definition. A number of printer manufacturers, Centronics included, do not use the full set of ground hookups defined, and in some cases, printer manufacturers who claim Centronics compatibility do not always attain it. If your dealer cannot help you in connecting a printer, WRITE to Osborne Computer Corporation Customer Service Dept. with a full description of the printer you are trying to connect, and we will attempt to help you get it working.

To use the printer connected to the IEEE interface, you must employ the **SETUP** program to configure the printer to the Centronics protocol. You need do this only once for each diskette. Alternatively, if you have two printers hooked up, or wish to change the printer being used from within a program, you can reset the IOBYTE as described earlier.

APPENDIX 1

Modifying WordStar

Modifying WordStar on the Osborne 1

Some Osborne 1 purchasers have asked Osborne Computer Corporation for help in modifying their copies of WordStar to work correctly with their printer. Unfortunately, over 500 different printers are available, some of which Osborne has no experience with. It is therefore virtually impossible for Osborne to provide the specific help many of you have requested.

At the same time, Osborne Computer Corporation is interested in providing as much support as possible to our customers. This document provides a general framework for making modifications to WordStar on the Osborne 1. Osborne will also work with FOG (First Osborne User Group) and other user groups to help bring specific modifications to the attention of Osborne 1 users. In 1982, we also began publication of a users' magazine (Portable Companion). With the help of the user groups and dealers, the magazine has become a further vehicle for disseminating information.

We're working on bringing our user support up to the highest level in the industry; if you'd also like to join an Osborne User Group, please consult the magazine or contact our customer service department for the group nearest you.

Modifying WordStar

If you want to modify WordStar, it is likely that you wish to make modifications to accommodate your printer. In this document, we will concentrate on making printer changes.

To modify WordStar, you'll need the following items:

Your WordStar diskette.
Your CP/M Utility diskette.

Your CP/M System diskette.
The assembly-language listing in this document, or the MicroPro Customization Notes.
A blank, formatted diskette.
The manual for your printer.
A hexadecimal/decimal/ASCII chart (helpful, but not necessary).

Make sure that you have these materials in front of you and that you've read through the documentation so that you're familiar with where to find specific information when you need it.

Step 1: Configuring the CP/M Printer Handler

Place your CP/M System diskette in drive A and the WordStar diskette in drive B. We assume that you're working with copies of both diskettes so that any mistakes you might make can be quickly reversed.

Press RESET on the front of the Osborne 1. Press the RETURN key to start CP/M. When the HELP program identifies itself, press the ESC (for ESCAPE) key to get to CP/M. Then type:

SETUP RETURN

When SETUP asks which disk you wish to configure, press **B** to indicate that you wish to reconfigure your WordStar diskette. After a few moments, the screen will clear and fill with a listing of the current settings.

We have now reached the moment of the first decision to be made: how is your printer connected to the Osborne 1? You must change setting A (PRINTER) and sometimes setting B (BAUD RATE) to reflect how your printer is connected to the computer.

If your printer is a low-cost dot-matrix printer, such as an Okidata, a Centronics, or an Epson, you will either select STANDARD SERIAL or CENTRONICS,

> depending on which interface option you purchased. Normally, these printers also run at 1200 baud, if you have a serial interface and the proper cable.
>
> If your printer is a high-quality daisywheel printer, it will probably connect to the serial port; a few models might connect like a Centronics parallel printer, however, so you'd better check. You may use the incremental spacing and other special printing capabilities of a daisywheel printer if you specify 300 BAUD, STANDARD SERIAL, or if you know enough about special printer protocols to use the XON/XOFF or ETX/ACK selections at 1200 baud. Using 300 baud will *always* work, so we suggest that you start with this rate.
>
> If your printer is intended for a Commodore or Hewlett-Packard computer, it may feature an IEEE-488 interface. In this case, specify IEEE as the printer type; baud rate need not be set.

If you don't know how your printer connects to the computer, you'll have to consult your dealer or the person who sold you the printer. Osborne Computer Corporation cannot supply you with the cabling or interfacing information.

While you're using SETUP to choose CP/M options, make sure that the ARROW KEYS are set for WORDSTAR (and not CP/M), that AUTO HORIZONTAL SCROLL is ON, and that SCREEN SIZE is set for 128. With the versions of WordStar being shipped, these options must be set correctly for WordStar to work as described in the User's Reference Manual.

You might also want to add some function keys while you're using SETUP. Commands that we find handy to have programmed into the function keys are:

^QQ^B	continuously reformat document
^OR	set right margin
^OL	set left margin
^OC	center line
^KD	save file

We suggest that you figure out your ten most frequently used commands and program them into the function keys; you don't have to, of course, but if you learn to use them properly, they do save time.

When you've finished making sure that all the options are properly configured, press **RETURN** (to exit), then press **B** to indicate that the configuration should be saved on drive B.

The disk drive, as usual, will whirr and clack, then shut off. You should see the CP/M prompt (A >).

Step 2: Figuring Out the Special Codes for Your Printer

You need to know certain things about the way your printer operates. We can't list all the different printers and the specific information you'll need to add or change in your source listing to allow WordStar to work properly with your printer.

What we will do is list each piece of information you need to modify, and discuss what each means. You'll have to sleuth through your printer's manual to find the specifics you need, but at least you'll have an idea of what you're looking for.

Usually, you're looking for a chart labeled "Print Control Codes." This chart will tell you the sequence of special characters the printer must receive to perform each function. These characters are usually multiple-character sequences (two or three characters in length) and often begin with the ESCAPE character. Normally, the control codes will be given in both decimal and hexadecimal format—we'll want the hexadecimal numbers to enter into WordStar.

The listing has two distinct patterns for the way WordStar wants to see character sequences:

1. The first number must be the number of characters in the sequence (zero characters if the function is not to be implemented), followed by the actual character sequence. For example, the Epson MX-80 uses the sequence **ESCAPE E** to change to enhanced print. This sequence looks like this:

   02h,1Bh,45h,0,0,0,0,0

   where 02h is the number of characters in the sequence,

   1Bh is the ESCAPE character (in hex),
   45h is the E (in hex), and the
   0s are there to take up the rest of the space dedicated to the function.

2. A single number or character is entered, and WordStar will supply the rest of the information needed. This is often called a "flag" or "toggle" function—the value "flags" which type of printer is being used or what functions are available.

Okay, here's the information you need to find out about your printer:

HOW TO OVERPRINT: The first thing you need to know is the instructions that tell WordStar how to create "overstrike" effects. An overstrike is simply two characters printed one on top of the other. WordStar uses overstrike to create boldface print, by printing the same character over and over; to underline, by printing the character, then the underline; and to print one character above another, as in foreign-language accents.

If you have a daisywheel printer, you'll just need to tell WordStar what kind it is.

If you own a dot-matrix or other non-daisywheel printer, you'll need to determine whether it is capable of backspacing or returning the carriage to the left margin without issuing a line feed. The latter method is preferred. The former is necessary for Selectric and other printers that cannot issue a carriage return without moving down a line on the paper.

If your printer can't backspace or issue bare carriage returns (no line feed), you will not be able to use WordStar's overstrike capabilities.

BOLDFACE PRINTING: Some printers do a better job than others of striking the same character over and over to create boldface print. The second item you need to know is the number of times you wish WordStar to print the same characters for all text between the **^PB** commands. Daisywheel printers should always have this set at 02. WordStar will automatically make this setting, so don't change it.

DOUBLESTRIKE PRINTING: The same criteria apply to doublestrike as to boldface. The difference between boldface print and doublestrike print should be readily apparent: boldface should be darker and bolder. Increase the number for this function so that the density of the print is somewhere between that of normal printing and boldface printing. Daisywheel printers should have this function set at 02.

ADVANCE A LINE: Some printers perform "automatic line feeds." This means that every time the computer sends a carriage return to the printer, the printer will perform a carriage return *and* a line feed. Normally, printers should perform a carriage return only, and perform a line feed only when the computer issues a line feed.

Therefore, to advance a line on the paper, WordStar usually has to send two characters: a carriage return followed by a line feed. If your printer performs auto line feeds, you'll need to get rid of the line feed in this function.

Another problem with some printers is that they take much longer to perform a carriage return and line feed than they do to print any other characters. The result is that these printers lose characters unless WordStar is told to wait a few moments after sending line feeds. You can't do this, but you can tell WordStar to send some extra characters—called "nulls" (00 hex)—which both WordStar and the printer will ignore. Each null sent by WordStar counts as a character only in the sense that it takes WordStar time to send it. If you send enough nulls, the printer will be ready to start printing the real characters once they begin arriving.

In other words, you may want to send a carriage return, line feed, and then several nulls with this function if you have a printer that performs carriage returns slowly—Selectric printers, for example.

RETURN TO SAME LINE: This function is much the same as the previous one. The difference is that this one doesn't issue a line feed, as you don't want the carriage moved to the next line, just to the left margin. The same comments about nulls mentioned for the previous function apply here.

ADVANCE ONE-HALF LINE: If your printer is capable of advancing the paper one-half line at a time, you may want to change this function so that you can print superscripts and subscripts. Do not change this function from all nulls if your printer is a daisywheel or capable of moving the carriage up *and* down on the paper (see below). You'll have to look through your printer's manual for the "character sequence" that performs this function, and then enter that sequence.

BACKSPACE: If you specify that your printer backspaces to overstrike, you need to enter into this function the character sequence your printer recognizes as the backspace command (usually ^H or 08 hex). If you have a daisywheel printer or specify that the printer should use carriage returns to perform overstrikes, you may simply fill this function with zeros.

ALTERNATE CHARACTER SET: If your non-daisywheel printer is capable of printing different-size characters, you need to enter the character sequence that tells the printer to change to the nonstandard size in this function.

Many dot-matrix printers have several different-size characters available. You'll have to pick one size as your normal character set and one as your alternate. It's usually a good idea to pick the alternate set as the smaller of the two—i.e., 16 characters per inch versus 10 per inch for normal. WordStar can't tell how large your characters are, and if you switch in the middle of a line to a larger character set, you may end up with text running off the right side of the page.

NORMAL CHARACTER SET: See discussion immediately above. If you specify an alternate character set, you must also specify a normal character set.

ROLL CARRIAGE UP PARTIAL LINE: To perform superscripts on non-daisywheel printers, entering the character sequence for "carriage up a partial line" is the preferred method of creating superscripts. If you enter a character sequence in this function, you must also enter a character sequence in the next function.

ROLL CARRIAGE DOWN PARTIAL LINE: This is the complement of the last function. The amount that the carriage is "rolled down" must equal the amount it is rolled up, otherwise you will end up with jagged lines when printing subscripts and superscripts.

USER PATCHES 1 THROUGH 4: WordStar allows four additional character sequences to be sent to the printer through definable "user functions." With non-daisywheel printers, it makes sense to put any additional character sizes (other than those entered above) into these functions, just in case you wish to use them. With printers such as the Centronics 737, you might want to specify that your printer enter the "proportional spacing" mode by employing a user function.

RIBBON CHANGE: If your printer allows you to switch between the black portion of the ribbon and another color, usually red, enter the character sequence that tells the printer to change to the alternate ribbon color in this function. Otherwise, fill it with zeros.

RIBBON CHANGE BACK: If you specify a ribbon change function, you need to tell WordStar the sequence of characters your printer recognizes; this sequence instructs the printer to return to the normal ribbon color. Enter that sequence in this function.

INITIALIZATION: You may send any sequence of up to 16 characters to your printer before WordStar first begins to send a file for printing. At the least, send the printer a carriage return. This will assure that the printer always starts at the left edge of the paper. Some users may wish to send a form feed (12 hex) to make sure that WordStar always starts printing at the top of a form, no matter what instructions preceded.

CONCLUSION: You may also elect to send your printer a sequence of characters at the end of each printing session. Some printers, for example, need to have their motor turned off by the computer. If you turn off the printer's motor using this function, make sure you turn it on in the initialization function, above.

STRIKEOUT CHARACTER: You may specify which character is used for strikeouts. WordStar comes with the "-" (hyphen) used as the strikeout character, but some people prefer a lowercase or uppercase X. Make sure at least some character is entered here or else you will never see strikeouts, even if you enter them into your document.

UNDERSCORE CHARACTER: You may also specify which character to use for underlining. The obvious choice is the underline (_), but some printers use special underlining characters.

PRINTER ROUTINE: WordStar uses one of five different ways of sending characters to the printer. Only three of these make sense on the Osborne 1:

CP/M list device
user subroutine
alternate console

You set up your WordStar diskette in step 1 for the first option, using the CP/M list device. In almost every instance, this is the only choice novice users should attempt to use. The other two possibilities both require some knowledge of either assembly language or the CP/M IOBYTE. The Customization Notes list these last two routines and show the areas to change; just make sure you know what you're doing if you begin tinkering with these two routines; otherwise, you may spend a lot of time going backwards (i.e., not printing!).

Step 3: Using INSTALL

Now comes the process of actually changing WordStar.

Place your CP/M Utility diskette in drive A and the WordStar diskette in drive B. Press RESET on the front of the Osborne 1 and press RETURN to load CP/M. Type **INSTALL RETURN** to load the WordStar installation program.

A copyright message will appear, and INSTALL will ask you:

```
Do you want a normal first-time INSTALLation of WordStar?

(Y = yes; N = display other options):
```

WordStar was installed by Osborne Computer Corporation the first time, so DO NOT answer with a **Y**; press N (for no). A new message will appear listing your options. Rather than explain each of them, choose option **"B"**—the only one that really makes sense given our sample installation.

INSTALL now asks:

```
FILE NAME OF WORDSTAR to be INSTALLed?
```

to which you should reply with the name of the WordStar program file: **B:WS.COM**. You must supply the **B:** because INSTALL needs to be told that your WordStar diskette is in drive B, not drive A.

Next, INSTALL asks:

```
File name for saving INSTALLed WordStar?
```

You will have to reply with **B:WS.COM** (i.e., the same name as the program you wish to modify) because there is not enough space on drive B to save a second copy of WordStar.

The next message you'll see is a "terminal choice menu." The proper response is to specify **U** indicating no change. Modifying the terminal information in WordStar makes no sense on the Osborne 1.

Finally come the printer options in INSTALL. The first question is what type of printer you have:

Printer Menu

(More specific information is displayed after you enter your choice)

```
A   Any "Teletype-like printer (i.e., almost any printer)

C   "Teletype-like" printer that can BACKSPACE

D   DIABLO 1610/1620 daisywheel printer

E   DIABLO 1640/1650 daisywheel printer

F   QUME Sprint 5 daisywheel printer

G   NEC Spinwriter 5520/5520 thimble printer  [ ? ]

H   "Half-line-feed" printers

M   I/O MASTER/OEM printer combination

U   no change
```

```
Z    none of the above

PLEASE ENTER SELECTION (1 LETTER):
```

This is where your sleuthing will start to pay off. The information you found earlier in the "How to Overprint" section is the primary data you need to make your selection:

> If you have a daisywheel printer, select the choice (D–G) that comes closest to the type of daisywheel printer you own.
>
> If you have a non-daisywheel printer and it can backspace, you may want to choose option C, whereas all others should choose option A.

When you pick an option, INSTALL will present some information about your choice and ask you if you are sure about your choice. Press Y if you want to continue, N if you want to reconsider your last choice.

Next, INSTALL asks you two questions about how WordStar is to communicate to your printer. You must answer these questions, even if you earlier told CP/M how to communicate to your printer.

Communications Protocol

WordStar needs a "communications protocol," a special method of communicating to the printer, for some types of printers. It does not make sense to have WordStar provide this protocol; however, CP/M can do it. If you have a daisywheel printer, specify the appropriate protocol using SETUP and specify **"NONE required (or handled outside WordStar)"** to the question INSTALL is now asking.

If you have a non-daisywheel printer, the appropriate choice is the same one: N for **"NONE required."**

Print Driver

WordStar also needs to know which method you wish it to use to find the instructions to print. Of the options shown, only **L** (CP/M list device) or **S** (user-installed printer driver) makes sense. Novices should always choose the CP/M list device; if you have access to the Customization Notes and know Z80 assembly language, you may want to code your own routine.

Special Modifications

INSTALL now asks you if your modifications are complete. If you have a daisywheel printer, you should reply **Y**, otherwise reply **N**.

Should you continue making changes, the following modifications are still open to you:

NAME	DEFAULT	DESCRIPTION
BLDSTR:	02	Number of strikes for boldface
DBLSTR:	02	Number of strikes for doublestrike
PSCRLF:	02,0D,0A,0,0, 0,0,0,0,0,	Advance to next line sequence
PSCRLF:+11	01,0D,0,0,0,0,0	Return carriage for overprint
PSHALF:	0,0,0,0,0,0,0	Half-line-feed sequence
PBACKS:	01,08,0,0,0,0	Backspace character sequence
PALT:	0,0,0,0,0	Alternate character-pitch sequence
PSTD:	0,0,0,0,0	Standard character-pitch sequence
ROLUP:	0,0,0,0,0	Superscript character sequence
ROLDOW:	0,0,0,0,0	Subscript character sequence
USR1:	0,0,0,0,0	User-defined character sequence
USR2:	0,0,0,0,0	User-defined character sequence
USR3:	0,0,0,0,0	User-defined character sequence
USR4:	0,0,0,0,0	User-defined character sequence
RIBBON:	0,0,0,0,0	Ribbon alternate character sequence
RIBOFF:	0,0,0,0,0	Ribbon standard character sequence

PSINIT:	01,0D,0,0,0 0,0,0,0,0,0,0,0, 0,0,0,0	Initialization sequence
PSINIT:+17	0,0,0,0,0,0,0, 0,0,0,0,0,0,0, 0	Ending sequence
SOCHR:	2D	Character used for strikeout
ULCHR:	5F	Character used for underlining

Default values: each hex number specified is one byte stored in WordStar. You may not use more bytes than are assigned to the default values, although you can use fewer by merely entering 00 hex for each unimportant byte.

The above table corresponds (in order) to the information you had to look up about your printer earlier. To substitute the information needed to control your printer in place of the default values listed above:

1. Type the name of the routine you wish to change.

   PALT: RETURN

   tells INSTALL you wish to change the alternate character-pitch sequence. INSTALL replies:

   ```
   ADDRESS: xxxxH OLD VALUE: xxH NEW VALUE:
   ```

2. Examine the "old value" and see if it is the same as for your printer. If it is, you need only press RETURN; if it isn't, enter the hex value you looked up in your printer's manual earlier. Remember, sometimes INSTALL needs a sequence of information, which must be provided in the order the printer expects it, preceded by the number of characters in the sequence.

Example:

> for the Epson MX-80, you might enter 02, meaning two characters are in the sequence, to tell your MX-80 to switch to a different character pitch.

Press RETURN after entering the correct hex value for your printer.

3. If more than one character needs to be sent to the printer for the function being modified, press RETURN a second time and INSTALL will show you what is stored in the second memory location of the function sequence. Again, enter the appropriate value for your printer. Repeat this process for all characters in the sequence, then go on to the next function (i.e., type the function's name when you see the prompt **"LOCATION TO BE CHANGED"**).
4. When you are through, enter **0** instead of a function name, and INSTALL will echo your selections and give you one last chance to change your mind.

Don't panic if your changes don't work the first time. Since you're working on a copy of WordStar, you don't have to worry about goofing things up—the worst you'll do is waste a few minutes.

APPENDIX 2

IEEE-488 Implementation

The Osborne 1 IEEE-488 Implementation

All commands (C1, 2, 3, 4, 25) of the IEEE-488 can be performed using eight calls (defined below). The easiest way to include IEEE-488 capability is to add these commands to the BIOS jump table. These commands are defined at a low level and must be used in an intelligent sequence, the commands are general purpose and can be used by different hardware implementations. There are, however, some restrictions that impose limits which are not part of the IEEE standard. Some restrictions can be overcome with increased software. The SRQ input can generate an interrupt, but commands assume the interrupt is not used. It would require a more than simple change to allow SRQ interrupts. A background scan of SRQ should satisfy most applications.

IEEE-488 command routines use no RAM other than the stack. Routines determine status by reading the current state of the PIA.

The software was made considerably more complicated by the use of bidirectional signals from the PIA. To reduce the overhead of determining the current direction of all signals, the PIA is always left in one of two modes: the source handshake mode, or the acceptor handshake mode. Several of the commands require that the PIA be in the source handshake mode when called. The PIA is normally in the source handshake mode following the completion of an IEEE bus transfer, so this is not a major restriction. Both Status In and Parallel Poll commands require that the PIA be in the source mode. This means that detection of device requests using either serial poll or parallel poll can be done only when the interface is idle.

To send data to a device on the IEEE bus, the controller makes the device a LISTENER, then assumes the role of a TALKER and sources the data. To receive data from a device, the controller makes the device a TALKER, then assumes the role of a LISTENER and accepts the data. It is also possible for the controller to enable both a TALKER and a LISTENER, then go to standby and allow the two devices to talk at their own rate. The problem is how to regain control.

The controller can take control asychronously by setting ATN true. If a device-dependent message is true when ATN becomes true, the interrupted byte could be misinterpreted by other devices as an interface message and produce unintended state transitions. The problem can normally be avoided by taking control synchronously. If high-speed transfer of data between the devices is not required, and if the computer can be tied up during the transfer, it is better to have the controller LISTEN (acceptor handshake) to the transfer while discarding the data. This allows the controller to count transfers, look for EOI signals, or timeout the TALKER before regaining control.

The "GOTO Standby" command has three uses. If the controller is going to allow an unmonitored data transfer between two other devices on the bus (without listening, as described above), this command is used to relinquish bus handshake control to the devices. Secondly, this command can be used after the completion of a bus transfer, causing the controller to float all signals (except REN) so that a second controller, also on the bus, can take control. This command is also used to initialize the interface following certain device handshake errors. There is no formal transfer of control as defined in the IEEE Specification, but it does allow an operator the ability to share IEEE devices on the bus.

Commands having loops that might cause the interface to "hangup" when a device on the bus is either busy or malfunctioning have error exits. Therefore, the programmer can define separate timeout limits, report the error, or recall the command and ignore the error.

The timeouts are very short, some as short as 100 microseconds, and are meant to occur often. Some IEEE peripherals use this timeout to indicate that they are not ready to receive data. When outputting to a slow device, e.g., a 300-baud printer, the printer will continuously not be ready for the next character. The program could be written to wait as long as required, even allowing for a printer that is off-line. While the program is waiting for the printer, it is possible for the computer to perform background functions following each timeout. If the command times out, the device can be UNLISTENED. This capability is necessary for devices such as modems and keyboard printers operating in full-duplex mode.

The following table summarizes CP/M BIOS entry and exit states for all IEEE-488 commands:

	CP/M BIOS CALL	ENTRY STATE	EXIT STATE
E13FH	Control Out	Any	Source, ATN High
E142H	Status In	Source	Source
E145H	GOTO Standby	Source	Source, ATN High
E148H	Take Control	Source, ATN High	Source, ATN Low
E14BH	Output Interface Message	Any	Source, ATN Low
E14EH	Output Device Message	Source	Source, ATN High
E151H	Input Device Message	Any	Acceptor, ATN High
E154H	Input Parallel Poll Message	Source	Source, ATN Low

Table A-1 IEEE-488 Exit and Entry States

CP/M BIOS calls receive parameters in registers C or BC, parameters are returned to A or HL. Only registers used to return results and status are modified by command execution.

Control Out

Bit No.	7	6	5	4	3	2	1	0
C Register						ENR	REN	IFC

IFC: If "1", the interface logic is initialized and IFC is driven low for 100 microseconds.

ENR	REN	
0	X	No action
1	0	REN set high
1	1	REN set low

When the PIA is reset, all internal registers are cleared. All I/O pins except REN become inputs and are pulled high by a pull-up register. REN becomes an input and floats, but has no pull-up register. Signals IFC, ATN, and REN power up till the active low state is achieved, but all other signals float. If IFC is set, the PIA is initialized to the source handshake mode in the standby state (ATN high), then the IFC signal is pulsed low to initialize all devices on the bus. The signal REN can be set or cleared independently.

Status In

No input parameters

Bit No.	7	6	5	4	3	2	1	0
A Register								SRQ

SRQ: Set to "1" if one or more devices are requesting service by driving SRQ low.

This command reads the service request signal. This signal is used by a device requiring attention and also to request that the current sequence of events be interrupted. If SRQ is "1" (low on the IEEE bus), then a device is requesting service; this signal is connected to PIA input CA1 and could be used to generate an interrupt. However, the software has not been written to support interrupts; to detect a request, the application program must periodically call Status In to read SRQ.

Since the CA1 input of the PIA can only sense signal transitions, not signal levels, a trick must be used to read the SRQ signal. An open-collector driver controlled by the PIA output PB2 (ENABLE EOI/DAV) is OR-tied to the output of the SRQ input buffer. If the SRQ signal is low, the buffer presents a high to the PIA. When signal PB2 is driven low, and input CA1 is also driven low, a transition is simulated that the PIA will detect. Conversely, if the SRQ signal is high, the buffer presents a low to the PIA. When signal PB2 is driven low, input CA1 remains low and there is no transition to detect.

This command can be called only when the interface is in the source handshake mode. To use the SRQ interrupt, all command routines would have to be modified to protect the interrupt state during their execution. Interrupts would have to be disabled during command execution.

GOTO Standby

No input or output parameters

This command causes the controller to enter the controller standby state where all signals are allowed to float. This command can be used as follows:

1) After completion of a bus transaction to free the bus for use by another controller.
2) To pass control to a device that the controller has previously made a TALKER.
3) To clear an interface error condition.

This command can only be called when the interface is in the source handshake mode. The only action performed by GOTO STANDBY is to clear the ATN signal and float the data bus.

Take Control

Bit No.	7	6	5	4	3	2	1	0
C Register								TCA

TCA: If "1", the controller will regain control of the bus asynchronously.

If "0", it will regain control synchronously.

Bit No.	7	6	5	4	3	2	1	0
A Register	ERR							ATO

ATO: Set if DAV remains active for longer than 100 microseconds during "take control synchronously."

ERR: Set if any error bit is set.

This command should only be used following a GOTO Standby that allows independent transfer of data between devices on the bus. The asynchronous take-control can be used safely only when there is no activity on the bus. Normally it is used to take control from a device that has malfunctioned and will not give up control synchronously. Use the synchronous take-control unless there is a special circumstance that requires asynchronous takeover.

If the data-valid timeout error is returned (ATO), it means that a LISTENER is slow in accepting data, or the TALKER is slow removing data. This command can be called repeatedly until the takeover is performed successfully.

Output Interface Message

Message byte in the C Register

Bit No.	7	6	5	4	3	2	1	0
A Register	ERR					ATO	RTO	NDP

NDP: No device present
RTO: Device not ready for data timeout
ATO: Data-not-accepted timeout
ERR: Set if any error bit is set

This command outputs the contents of Register C as an interface message (with the ATN signal active). If ATN is not active when called, ATN is first driven low and is left low following the data transfer.

The command first checks the ENABLE-DATA-OUT bit (PBO) to determine if the interface is in the source handshake mode or the acceptor handshake mode. If this bit is set, then the interface is in the acceptor mode and must be switched to the source mode. Before starting the source handshake, the routine checks if the DAV signal is being driven low by the interface. If it is being driven low, then the command is being reentered following an ATO timeout error and the first portion of the source handshake is branched over.

The handshake begins with the command placing register C contents on the bus. The routine then checks the NRFD signal to determine if the devices on the bus are all ready for data. If this signal remains low for more than 100 microseconds, the command returns the RTO timeout error. This does not mean that a hardware error has occurred, only that the devices on the bus are not all currently ready for data. When the NRFD signal becomes high, the routine checks the NDAC signal. If this signal is also high, it means that there is no active device on the bus and the NDP error is returned. If the NDAC signal is low, DAV is driven low to indicate that there is valid data on the bus. The command then checks the NDAC signal to determine if all the devices on the bus have accepted data. If this signal remains low for more than 1000 microseconds, the ATO timeout error is returned. Again this does not mean that a hardware error has occurred, but it should be less common than the RTO timeout errors, since few IEEE-Std 488 devices use this portion of the handshake to control their input rate. When the NDAC signal becomes high the DAV signal and data is removed, and the A register is cleared. The command then ends.

The NDP error occurs when there are no devices on the IEEE-Std 488 bus, or power is off at all devices. This error leaves data on the bus and should be followed by a GOTO Standby command in order to float the bus.

The RTO error occurs when a device does not become ready for data within 100 microseconds. Most IEEE-Std 488 devices can accept interface control messages at or near the full rate of the interface, but there are exceptions. This error can be handled in any of several ways. It can be ignored by continuously recalling the command until all devices become ready, or the command can be called a fixed number of times (effectively lengthening the timeout) before responding to the error, or the error can be reported immediately. This error also leaves data on the bus and should be followed by a GOTO Standby command to float the bus.

The ATO error occurs when a device does not accept data that it has already agreed to accept. If this error persists, a GOTO Standby command followed by an UNLISTEN may clear the error, or it may be necessary to command an IFC and initialize the interface. It cannot be determined whether valid data was transferred if the handshake is aborted here.

Output Device Message

Data byte in C Register

Bit No.	7	6	5	4	3	2	1	0
B Register								EOI

EOI: If "1", the EOI signal will be driven low when the data is on the bus.

Bit No.	7	6	5	4	3	2	1	0
A Register	ERR					ATO	RTO	NDP

NDP: No device present
RTO: Device not ready for data timeout
ATO: Data-not-accepted timeout
ERR: Set if any error bit is set

This command outputs the byte in register C as device-dependent data (with ATN SIGNAL high). If ATN is active when called, ATN is first driven high and is left high following the data transfer.

This command is called only when in source handshake mode. If bit 0 of register B is "1", then the EOI signal is driven low. Before starting the source handshake the command checks whether the DAV signal is being driven low by the interface. If it is being driven low, then the command is being reentered following an ATO timeout error and the first portion of the source handshake is branched over. The handshake begins with the interface replacing the data byte on the bus. The command then checks the NRFD signal to determine whether all LISTENERS are ready for data. If this signal remains low for more than 100 microseconds, then the routine returns the RTO timeout error. This does not mean that a hardware error has occurred, but only that a device on the bus is not currently ready for data.

When the NRFD signal becomes high, the command checks the NDAC signal. If this signal is also high at this time, it means that there is no active LISTENER on the bus so the routine returns the NDP error. If the NDAC signal is low, the command responds by driving DAV low to identify valid data on the bus. The command then checks the NDAC signal to determine whether all the LISTENERS have accepted the data. If this signal remains low for more than 1000 microseconds, the command returns the ATO timeout error. Again, this does not mean that a hardware error has occurred, but this error should be less common than the RTO timeout error since few IEEE-Std 488 devices use this portion of the handshake to control their input rate. When the NDAC signal becomes high, the command removes the DAV signal and the data, clears the A register, and returns to the calling program.

The NDP error occurs when none of the devices on the bus have been made a LISTENER. This could be caused by using a nonexistent IEEE bus address, or the device addressed may not be powered up. This error leaves data on the bus and should be followed by a GOTO Standby command to float the bus.

The RTO error occurs when a device does not become ready for data within 100 microseconds. Most IEEE-488 devices cannot accept device-dependent data at the full rate of the interface. This error can be handled in any of several ways. It can be ignored by continually recalling the command until all LISTENER devices become ready, or the command can be called a fixed number of times (effectively lengthening the timeout) before responding to the error, or the error can be reported immediately. As an example, a printer may only be able to accept data at a rate of 30 CPS. If the delay between transfers is much greater than this, it could mean that the printer is off-line or that the paper has jammed. It is possible to check a device and determine whether it is ready for data by attempting to transfer a data byte. If the command returns a timeout error, the device can be issued an UNLISTEN and the interface is then free for other transactions. Since this error leaves data on the bus, it should be followed by an UNLISTEN command to free the bus.

The ATO error occurs when a device does not accept data that it has already agreed to accept. If this error persists, a GOTO Standby command followed by an UNLISTEN may clear the error, or it may be necessary to command an IFC to initialize the interface. It cannot be determined whether valid data had been transferred if the handshake is aborted here.

Input Device Message

No input parameters

Data bytes in A and H Registers

Bit No.	7	6	5	4	3	2	1	0
L Register	ERR					NTO	VTO	EOI

EOI: Set to "1" if the EOI signal was low when the data was read
VTO: Data-valid timeout
NTO: Data-not-valid timeout
ERR: Set if any error bit is set

This command inputs a device-dependent data byte (with ATN high), reads the EOI signal, then returns the results in registers A, H, and L.

The command first saves the contents of register HL in register DE. Register HL will contain input data if the routine is being reentered following an NTO timeout error. The ENABLE DATA OUT bit (PB0) is checked to determine if the interface is in the source handshake mode or the acceptor handshake mode. If this bit is clear, then the interface is in the source mode and is switched to the acceptor mode. When the interface is in the source handshake mode, the ATN signal will also be low. Signals NRFD and NDAC are driven low before the ATN signal is set high.

Before starting the acceptor handshake, the routine checks if the NDAC signal has been set high by the interface. If it is high, then the command is being reentered following an NTO timeout and the first portion of the acceptor handshake is branched over. The handshake sequence is initiated with the command setting signal NRFD high to indicate that it is ready for data. The command then checks the DAV signal to determine whether the TALKER has placed data on the bus. If this signal remains high for more than 100 microseconds, then the command returns the VTO timeout error. This does not mean that a hardware error has occurred, rather that the TALKER does not currently have data available for transfer. When the DAV signal becomes low, the command drives signal NRFD high to indicate that it is no longer ready for data. It then reads data and the EOI signal. Next the command sets signal NDAC high to indicate that it has accepted data. The command then checks the DAV signal to determine whether the TALKER has removed its data. If this signal remains low for more than 1000 microseconds, the command returns the NTO timeout error. Again, this does not mean that a hardware error has occurred, but it should be less common than the VTO timeout error since few IEEE-Std 488 devices use this portion of the handshake to control their output rate. When the DAV signal becomes high, the routine drives NDAC low and returns.

The VTO error occurs when a device does not make data available within 100 microseconds. Most IEEE devices will not supply data at this rate. This error can be handled in several ways. It can be ignored by continuously recalling the command until data becomes available, or the command can be called a fixed number of times (effectively lengthening the timeout) before responding to the error, or the error can be reported immediately. As an example, a keyboard may transfer a character only when a key is pressed. The time separating key entries could be very long. To exit from this error state, it is necessary to UNTALK the slow-responding device. There is a possibility of losing a character from the device when this is done, so if the device supports another method to check for data available, it should be used. Also, if the TALKER places the data on the bus after the error was detected, but before the UNTALK begins, then the ATN will occur on the bus with the data and could be misinterpreted by other devices as an interface message; this will produce unintended state transitions.

The NTO error occurs when the TALKER does not remove the data after that data has been accepted by the controller. This can occur if the controller has enabled other LISTENERS on the bus, and not all devices have accepted the data, or if the TALKER is slow in removing the data. If this error persists, an UNTALK to the device must be sent. This could also cause unintended state transitions since the TALKER does have data on the bus. If the UNTALK is not accepted, an IFC should be used to clear this error state. The data returned when the error was reported is valid.

Input Parallel Poll Messages

No input parameters

A = Parallel poll response

This command can only be called in the source handshake mode. The controller may be in the Standby state, but ATN is left low following this command. The routine drives both the ATN and EOI signals low. All devices on the bus with parallel poll capability respond by driving their previously assigned data bit low. The command floats the internal data bus, then reads the parallel poll data. The command restores the source handshake, then ends. The GOTO Standby command should be called following this command to float the bus.

IEEE-488 Sample Programs

The sample programs ignore error returns from the IEEE-Std 488 BIOS calls, except to reenter the routine if any error condition exists. A real I/O driver should analyze errors and determine the correct action. Timeout errors might be ignored but the "no device present" error must generate an I/O error.

Make LISTENER

There are two alternative versions of the L interface function: one with, and one without address extension. The normal L function uses a 1-byte address. The L function with address extension (the LE function) uses a 2-byte address. This example implements the LE function.

```
;INPUT PARAMETERS:   C    PRIMARYADDRESS (0–31)
                     B    SECONDARYADDRESS (0–31)

ML:     LD    A,C                   ;FORM PRIMARY
                                    ;ADDRESS MESSAGE
        ADD   00100000B
        LD    C,A
ML10:   CALL  E14BH                 ;OUTPUT INTERFACE
                                    ;MESSAGE
        OR    A
        JR    NZ,ML10               ;REENTER ON ERROR
        LD    A,B                   ;FORM SECONDARY
                                    ;ADDRESS MESSAGE
        ADD   01100000B
        LD    C,A
ML20:   CALL  E14BH                 ;OUTPUT INTERFACE
                                    ;MESSAGE
        OR    A
        JR    NZ,ML20               ;REENTER ON ERROR
```

Make TALKER

There are two alternative versions of the T interface function: one with and one without address extension. The normal T function uses a 1-byte address. The T function with address extension (the TE function) uses a 2-byte address. This example implements the TE function.

```
;INPUT PARAMETERS     C      PRIMARYADDRESS (0–31)
                      B      SECONDARYADDRESS (0–31)

MT:       LD      A,C                         ;FORM PRIMARY
                                              ;ADDRESS MESSAGE
          ADD     01000000B
          LD      C,A
MT10:     CALL    E14BH                       ;OUTPUT INTERFACE
                                              ;MESSAGE
          OR      A
          JR      NZ,MT10                     ;REENTER ON ERROR
          LD      A,B                         ;FORM SECONDARY
                                              ;ADDRESS MESSAGE
          ADD     01100000B
          LD      C,A
```

Second-Level Software Description

```
MT20      CALL    E14BH                       ;OUTPUT INTERFACE
                                              ;MESSAGE
          OR      A
          JR      NZ,MT20
```

UNLISTEN

```
UL:       LD      C,00111111B                 ;UNLISTEN MESSAGE
UL10:     CALL    E14BH                       ;OUTPUT INTERFACE
                                              ;MESSAGE
          OR      A
          JR      NZ,UL10                     ;REENTER ON ERROR
```

UNTALK

```
UT:       LD      C,01011111B                 ;UNTALK MESSAGE
UT10:     CALL    E14BH                       ;OUTPUT INTERFACE
                                              ;MESSAGE
          OR      A
          JR      NZ,UT10                     ;REENTER ON ERROR
```

OUTPUT DATA

```
;INPUT PARAMETERS     C      DATA BYTE
                      B      EOI

OD:      CALL   E14EH                      ;OUTPUT DEVICE
                                           ;MESSAGE
         OR     A
         JR     NZ,OD                      ;REENTER ON ERROR
```

INPUT DATA

```
;OUTPUT PARAMETERS    A,H    DATA BYTE
                      L      EOI

ID:      CALL   E151H                      ;INPUT DEVICE
                                           ;MESSAGE
         BIT    7,L
         JR     NZ,ID                      ;REENTER ON ERROR
```

Higher-Level Functions

The following examples use the above routines to perform the standard interface functions. This is not meant to be a complete description of the capabilities of the interface.

OUTPUT DATA TO A DEVICE

```
         LD     BC, DEVICE ADDRESS
         CALL   ML                         ;MAKE LISTENER
         LD     BC,DATA                    ;DATA AND EOI
         CALL   DO
                ;ANY NUMBER OF DATA BYTES MAY BE SENT
         CALL   UL                         ;UNLISTEN
         CALL   E145H                      ;GO TO STANDBY
```

INPUT DATA FROM A DEVICE

```
        LD     BC,DEVICE ADDRESS
        CALL   MT                        ;MAKE TALKER
        CALL   ID                        ;INPUT DATA AND EOI
        ;ANY NUMBER OF DATA BYTES MAY BE RECEIVED
        CALL   UT                        ;UNTALK
        CALL   E145H                     ;GO TO STANDBY
```

Data Transfer Between Devices (Monitored)

This will make one device a TALKER and another device a LISTENER, then let the TALKER control the bus. The controller will monitor the bus and take control when it detects an EOI.

```
        LD     BC,TALKER ADDRESS
        CALL   MT
        LD     BC,LISTEN ADDRESS
        CALL   ML
LOOP:   CALL   ID                        ;INPUT DATA AND EOI
        BIT    0,L
        JR     Z,LOOP                    ;LOOP UNTIL EOI LOW
        CALL   UT                        ;UNTALK
        CALL   UL                        ;UNLISTEN
        CALL   E145H                     ;GO TO STANDBY
```

Data Transfer Between Devices (Unmonitored)

This will make one device a TALKER and another device a LISTENER, then let the TALKER control the bus. The controller will wait a fixed delay then take control synchronously.

Second-Level Software Description

```
        LD     BC,TALKER ADDRESS
        CALL   MT
        LD     BC,LISTEN ADDRESS
        CALL  ML
        CALL  E145H                  ;GO TO STANDBY
        ;FIXED DELAY WHILE DATA IS TRANSFERRED
        LD    C,0
LOOP:   CALL  E148H                  ;TAKE CONTROL
                                     ;SYNCHRONOUSLY
        OR    A
        JR    NZ,LOOP                ;WAIT FOR CONTROL
        CALL  UT                     ;UNTALK
        CALL  UL                     ;UNLISTEN
        CALL  E145H                  ;GO TO STANDBY
```

Serial Poll

This is an example of a serial poll. It assumes that the detection of the service request was performed separately. It is important that every device capable of generating the service request is polled to assure that two devices were not simultaneously driving the service request signal:

```
        LD    C,00011000B
LOOP1:  CALL  E14BH                  ;SERIAL POLL ENABLE
        OR    A
        JR    NZ,LOOP1               ;REENTER ON ERROR
        LD    BC,TALKER ADDRESS
        CALL  MT                     ;MAKE FIRST DEVICE
                                     ;A TALKER
        CALL  ID                     ;INPUT STATUS BYTE
        CALL  UT                     ;UNTALK

        ;ACT ON STATUS
        LD    BC,TALKER ADDRESS
        CALL  MT                     ;MAKE SECOND
                                     ;DEVICE A TALKER
        CALL  ID                     ;INPUT STATUS
        CALL  UT                     ;UNTALK
        ;ACT ON STATUS
        LD    C,00011001B
```

```
LOOP2:  CALL  E14BH              ;SERIAL POLL DISABLE
        OR    A
        JR    NZ,LOOP2           ;REENTER ON ERROR
        CALL  E145H              ;GO TO STANDBY
```

SEND LOCAL LOCK OUT (UNIVERSAL)

```
        LD    C,00000110B
        CALL  E13FH              ;SET REN LOW
        LD    C,00010001B
LOOP:   CALL  E14BH              ;SEND LOCAL LOCK
                                 ;OUT
        OR    A
        JR    NZ,LOOP            ;REENTER ON ERROR
        LD    BC,LISTEN ADDRESS
```

Second-Level Software Description

```
        CALL  ML                 ;FIRST DEVICE GO
                                 ;TO REMOTE
        LD    BC,LISTEN ADDRESS
        CALL  ML                 ;SECOND DEVICE GO
                                 ;TO REMOTE
        CALL  UL                 ;UNLISTEN
```

GO TO LOCAL (ADDRESSED)

```
        LD    BC,DEVICE ADDRESS
        CALL  ML                 ;MAKE FIRST DEVICE
                                 ;A LISTENER
        LD    BC,DEVICE ADDRESS
        CALL  ML                 ;MAKE SECOND
                                 ;DEVICE A LISTENER
        LD    C,00000001B
```

```
LOOP:   CALL  E14BH                   ;GO TO LOCAL
        OR    A
        JR    NZ,LOOP                 ;REENTER ON ERROR
        CALL  UL                      ;UNLISTEN
        CALL  E145H                   ;GO TO STANDBY
```

REMOVE LOCAL LOCK OUT (UNIVERSAL)

```
        LD    C,00000100B
        CALL  E13FH                   ;SET REN HIGH
```

PARALLEL POLL CONFIGURE

```
        LD    BC,LISTEN ADDRESS
        CALL  ML                      ;MAKE LISTENER
        LD    C,00000101B
LOOP1:  CALL  E14BH                   ;PARALLEL POLL
                                      ;CONFIGURE
        OR    A
        JR    NZ,LOOP1                ;REENTER ON ERROR
        LD    C,0110SPPPB
LOOP2:  CALL  E14BH                   ;PARALLEL POLL
                                      ;ENABLE
        OR    A
        JR    NZ,LOOP2                ;REENTER ON ERROR
        CALL  UL                      ;UNLISTEN
        CALL  E145H                   ;GO TO STANDBY
```

PARALLEL POLL UNCONFIGURE (UNIVERSAL)

```
        LD    C,00010101B

LOOP:   CALL  E14BH                   ;PARALLEL POLL
                                      ;UNCONFIGURE
        OR    A
        JR    NZ,LOOP                 ;REENTER ON ERROR
        CALL  E145H                   ;GO TO STANDBY
```

PARALLEL POLL DISABLE (ADDRESSED)

Second-Level Software Description

```
        LD      BC,LISTEN ADDRESS
        CALL    ML                      ;MAKE LISTENER
        LD      C,00000101B
LOOP1:  CALL    E14BH                   ;PARALLEL POLL
                                        ;CONFIGURE
        OR      A
        JR      NZ,LOOP1                ;REENTER ON ERROR
        LD      C,01110000B
LOOP2:  CALL    E14BH                   ;PARALLEL POLL
                                        ;DISABLE
        OR      A
        JR      NZ,LOOP2                ;REENTER ON ERROR
        CALL    UL                      ;UNLISTEN
        CALL    E145H                   ;GO TO STANDBY
```

DEVICE CLEAR (UNIVERSAL)

```
        LD      C,00010100B
LOOP:   CALL    E14BH                   ;DEVICE CLEAR
        OR      A
        JR      NZ,LOOP                 ;REENTER ON ERROR
        CALL    E145H                   ;GO TO STANDBY
```

DEVICE CLEAR (ADDRESSED)

```
        LD      BC,DEVICE ADDRESS
        CALL    ML                      ;MAKE FIRST DEVICE
                                        ;A LISTENER
        LD      BC,DEVICE ADDRESS
        CALL    ML                      ;MAKE SECOND
                                        ;DEVICE A LISTENER
        LD      C,00000100B
```

```
LOOP:   CALL  E14BH                         ;SELECTED DEVICE
                                            ;CLEAR
        OR    A
        JR    NZ,LOOP                       ;REENTER ON ERROR
        CALL  UL                            ;UNLISTEN
        CALL  E145H                         ;GO TO STANDBY
```

DEVICE TRIGGER

```
        LD    BC,DEVICE ADDRESS
        CALL  ML                            ;MAKE FIRST DEVICE
                                            ;A LISTENER
        LD    BC,DEVICE ADDRESS
        CALL  ML                            ;MAKE SECOND
                                            ;DEVICE A LISTENER
        LD    C,000010008B
LOOP:   CALL  E14BH                         ;GROUP EXECUTE
                                            ;TRIGGER
        OR    A
        JR    NZ,LOOP                       ;REENTER ON ERROR
        CALL  UL                            ;UNLISTEN
        CALL  E145H                         ;GO TO STANDBY
```

APPENDIX 3

SuperCalc Installation

SuperCalc Installation Procedure

Follow these steps to alter the screen dimensions, or to configure SuperCalc for compatibility with your printer:

1. Load SuperCalc on drive A. Press RETURN to load, then press RETURN again to start. Next, exit to CP/M with the command **/QY**. When the A> prompt appears, type:

   INSTALLS

   and RETURN. You'll see this message:

   ```
   This program will let you modify the SuperCalc™
   file on your disk. Do you wish to proceed (Y/N)?
   ```

2. Type **Y** for yes if you want to alter the default settings. Then this message is added:

   ```
   Enter the name of the SuperCalc™ file as:
   "d:filename" where "d" is the drive.

   Enter name: —
   ```

   After typing the file name, this list of options is offered:

   ```
   SuperCalc Install Program
   A. Edit Screen dimensions
   B. Edit Printer dimensions
   C. Edit Printer initialization string
   D. Edit Border character
   E. Save SuperCalc™ on disk
   X. Exit without changing SuperCalc

   Enter option letter :—
   ```

3. If you want to change the height or width of the current screen display, type A. The options for changing either of these conditions as well as their present settings will be presented as shown below:

   ```
   A. The current screen height is      :  24
   ```

```
B. The current screen width is      :  52
X. Exit to previous menu

Enter option letter :
```

To change the screen height, type A; to change the screen width, type B. After typing either A or B you will be asked to supply the new value. Enter the new screen size and press RETURN; the new value will be shown. When you have made the necessary alterations, simply type X and you will be returned to the initial menu.

4. If you want to alter the size of a printed page, type B from the initial menu. You will be provided with these options:

```
A. The current printer page length is      :  66
B. The current printer page width is       : 132
X. Exit to previous menu
```

Type the letter of the value you want to change, enter the new value, and press RETURN as explained earlier. The newly configured page dimensions will be shown. Type X to get back to the initial menu.

5. In some instances, a select string of code needs to be output to the printer before it is activated. This printer initialization string is sometimes required by certain printers or is needed to take advantage of certain print features such as compression (consult your printer manual). In order to configure this printer initialization string, type C from the main menu and the following message and prompt will appear:

```
The printer initialization string is a series of up to
8 hex bytes that will be output to the printer prior
to printing a worksheet.

A. Current printer initialization
   string  :  Unconfigured
X. Exit to previous menu
```

As indicated, the initialization string is currently unconfigured. Type A and enter the desired value, then type X to return to the main menu.

6. Selecting option letter D allows editing the border character; this is a displayable, printable ASCII character that appears on the column border on the terminal and both row and column borders when printing a worksheet.

```
A. Current border character          :  7Ch
X. Exit to previous menu
```

7. After making the required adjustments through this installation program, save the changes on disk by typing E. This message will appear indicating that your changes are being installed:

```
This will install the changes you have made into
your SuperCalc™ program
```

If for any reason you want to leave the install program without implementing any changes, type X and you will be returned to CP/M.

OSBORNE 1™

Reference Guide

Table of Contents

PART 2

This section of the Osborne 1 User Guide provides an in-depth compilation of facts regarding the software that directs your computer. A complete listing of all the software error messages that you may encounter while running Osborne 1 standard software is also provided.

The following terms and abbreviations are used throughout this section:

System	CP/M operating system
Logged drive	disk drive currently in use
Default	usual condition
Toggle	turns function ON or OFF dependent on its current state
Prompt	message prompting action
Scroll	moving the entire screen
Justification	margin alignment of text
Dot command	command characters preceded by a period
Block	section of text
Worksheet	SuperCalc grid delimited by coordinates
Cell	a single SuperCalc worksheet coordinate
Statement number	any valid CBASIC or MBASIC line number
Constant	a real number, integer, or string with a fixed value
Delim	a punctuation mark used for separations
Parm	defines specific parameters
(Exp)pression	a single constant, variable, or expression
Integer exp	an expression with integer value
Numeric exp	an expression with real, or integer numeric value
Real exp	an expression with a real numeric value
String exp	an expression with a string value
Variable	simple or subscripted variable type of information
hex	hexadecimal
byte	equivalent to one character
<ESC>	ESCAPE key
R/O	assignment of read-only attributes
x:	drive identifier (A: or B:)

<**cr**>	carriage return
^	represents pressing the control key (CTRL)
#	represents a number
K	kilobyte
{ }	indicates that the enclosed parameters may be repeated
[ ]	indicates that the enclosed parameters are optional

CP/M

Control Characters

Control characters are special characters in CP/M command lines. CP/M recognizes some control characters when they appear on a command line. You create each control character by holding down the control key (labelled CTRL) while simultaneously pressing another key. In some cases, CP/M may also recognize these control characters when you are running an application program.

In the examples below, the following nomenclature is used:

^ represents pressing CTRL.

n represents a number.

^C causes a general system restart (called a warm boot). To be recognized, this command must be the first on a CP/M command line. Always use this command after changing a diskette at the CP/M command level.

^E causes a carriage return and line feed at the console display, but does not send the command line for processing.

Example:

You type on the command line then press ^E <cr>
which takes you to the left margin ^E <cr>
of the next line, as in this example ^E <cr>
(this is not a valid CP/M command, by the
way).

^H backspaces one character position.

^J performs a line feed and sends the current command line to be processed.

^M is the same as a carriage return; it causes a line feed and carriage return and sends the current command line to be processed.

^P toggles the printer. If the printer is OFF, pressing **^P** will duplicate all subsequent console output on the list device. If the printer is already ON, issuing this command will stop sending output to the printer.

^R redisplays the current command line on the next line without deleted characters.

Example:

```
A> MBASIC PROGRAM:RANDOMIZE^R
   MBASIC PROGRAM:RANDOMIZE
```

^S stops the console display temporarily. You can press this command when CP/M is displaying any message or file dump, but it terminates execution of certain commands. Pressing **^S** again will restore execution of a program or command and restore output to the display.

^U deletes the entire command line that has been typed and places your cursor on the line immediately below the current command line.

Example:

```
A> MBASIC PROGRAM ^U
     ←Would position cursor here)
```

^X functions exactly as **^U** (i.e., it deletes the current command line), but does so by backspacing.

^Z ends input from the system console (keyboard). This control character is used in PIP and ED to indicate that the keyboard has transmitted the end of a file.

CP/M Commands (.COM files)

ASM—*is Digital Research's 8080 Assembler*

The file ASM.COM contains Digital Research's 8080 Assembler. This assembler can assemble any source program written using standard Intel-format mnemonics. Two new files are created: one with the same file name and an extension of .PRN contains the annotated source listing after assembly; another with the same file name and an extension of .HEX contains Intel hex-format object code.

The generalized process of using the assembler follows:

1. Create your source file using the **N** command in WordStar.
2. Use the assembler as indicated below to create a .HEX file.

3. Use DDT to run and debug your program; reedit and reassemble the source file as necessary.
4. Use LOAD to convert the finished program from hex format to an executable .COM-type file.

Format:

```
ASM x:filename.xxx <cr>
```

ASM loads and executes the assembler program. You can use any valid CP/M file name that has an extension of .ASM and contains assembly-language source code.

The three letters (xxx) following the period represent the drive of the three files the assembler uses or creates; xxx is not the extension of the file name. The first letter represents the disk drive that contains the source file. The second letter represents the disk drive on which the hex file should be placed. The third letter represents the drive on which the annotated listing file should be placed.

You may type any valid disk drive identifier (without the colon) for each of these letters. You may use "Z" in the second or third position to indicate that you wish the assembler to skip generation of the hex or listing file. You may use "X" in the third position to indicate that you wish the listing file to go to the printer and not to a disk file.

If you do not type the period and the three letters after the file name, the assembler will assume that all files are to be found or placed on the current default disk drive.

Examples:

ASM SP <cr>

assembles the file SP.ASM on the current default disk drive. A .PRN and .HEX file will be created on the default drive as well.

ASM SP.ABX <cr>

assembles the file SP.ASM that is on the primary drive (drive A), places the resultant .HEX file on drive B, and sends the annotated listing to the printer instead of a disk file.

DDT—*is a dynamic debugging tool*

The DDT program allows you to interactively test, modify and disassemble 8080 object programs. When DDT is loaded into memory it immediately moves to the top of the free memory space, then it installs some "hooks" that control all input, output, and execution until you return to the CP/M command level.

Formats:

DDT <cr>

This form loads DDT into memory and executes it. No file is loaded into memory with DDT.

DDT x:filename.typ

This form loads DDT into memory and executes it. Before control passes to you, the specified file is also loaded into memory.

The file type must be .HEX or .COM. Other extensions will not work.

Once DDT is in control of the system, the following commands are allowed (# represents hexadecimal number):

A# allows entry of an assembly-language mnemonic beginning at the hex address requested.

D#, # displays a hexadecimal and ASCII dump of memory beginning at the first and continuing to the second hexadecimal address requested. You may leave off the second address to display only one page of memory.

F#, #, # fills all memory inclusive between the first and second hexadecimal addresses requested with the third hexadecimal value indicated.

G# begins execution at the hexadecimal address requested. Note that neither DDT nor CP/M is protected. Execution of your program could result in a reboot of CP/M, or worse.

H#, # computes the sum and difference of the two hexadecimal numbers you specify. The first number shown will be the sum; the second, the difference.

Ifilename sets the file control block used by CP/M to reflect the file name specified. If the file also has an extension, be sure to type it.

L#, # lists in 8080 assembly-language mnemonics the machine code between the first and second hexadecimal addresses requested; you may leave off the second address to see only 12 lines of disassembly at a time.

M#, #, # moves the block of memory between, and including, the first two addresses to the area of memory beginning at the third location.

R or R# reads a .COM- or .HEX-type file into memory at its proper position if no number is specified; or it reads a file into memory with an offset bias of the hexadecimal value you type.

S# allows you to begin inserting machine code in hex at the hexadecimal address you supply.

T or T# sets the trace function so that the CPU state is displayed on the console; the optional decimal number indicates the number of instructions to be executed and which tracings are to be displayed.

U or U# is identical to the trace command except that the CPU state is not displayed.

Xregister allows you to examine the current CPU state if no register is specified; if a register is specified, its current state is displayed and any valid entry you type (other than a carriage return) will be entered as the new contents of that register. C, Z, M, E, and F represent flags. A, B, D, and H are registers. S is the stack pointer, and P is the program counter.

Examples:

DDT <cr>

loads and executes DDT; a "-" prompt indicates that DDT is waiting for a valid command.

DDT TNT.HEX <cr>

loads and executes DDT, which, in turn, loads TNT.HEX into the proper position in memory; the following display will appear:

```
DDT VER X.X
NEXT PC
0111 0000
```

The number under "PC" indicates that the program counter is set to 0000 hex. The number 0111 under "NEXT" indicates that the next free memory location is 0111 hex.

DIR—*displays a disk directory*

The DIR command displays a complete or partial directory of the contents of any currently active diskette.

Formats:

DIR <cr>

This is the general form to show entire directory of current default drive.

DIR X: <cr>

This form shows entire directory of specified diskette.

DIR x:filename.typ <cr>

Form to show partial directory of a diskette. Note that "?" and "*" can be used to create ambiguous file names. DIR * .* <cr> is the same as DIR <cr>.

Examples:

DIR <cr>

displays complete directory for the current default diskette.

DIR *.BAS <cr>

displays all files, regardless of the file name, that have the extension of .BAS and are on the current default diskette.

DIR *.BA? <cr>

displays all files, regardless of file name, that have an extension whose first two characters are BA. The third character in the extension may be anything. Files must be on the current default diskette.

DUMP—*displays disk file in hex*

The DUMP program displays the named file in Intel hex format. The file is listed in 16 bytes per line, with the absolute byte address displayed in hexadecimal at the beginning of each line.

Format:

DUMP x:filename.type <cr>

This is the only form allowed for this command; if a drive is not specified, the current default drive is used.

Sample Dump Display

```
0000 57 52 49 54 54 4E 20 42 59 20 54 48 4F 4D 20
0010 48 4F 47 41 3E 20 1A 1A 1A 1A 1A 1A 1A 1A 1A
 ^                            ^
absolute            Intel hex format display
byte address        of file contents
```

Example:

DUMP WASTE.PCB <cr>

will display an Intel-format dump of the file WASTE.PCB.

ED—*is Digital Research's editor*

ED is the editor Digital Research supplies with CP/M. This editor is primitive; WordStar is capable of performing the same task and is easier to use.

Format:

ED x:filename.type <cr>

General form used to invoke the editor. Once the editor has been loaded, several commands are available for use. If you must use this editor, consult one of the books now available on CP/M for details on its use.

Example:

```
ED MAC.MAN <cr>
```

instructs CP/M to load and execute the editor, with the file MAC.MAN to be referenced while the editor is in control of the system.

ERA—*erases a disk file*

The ERA command deletes an entry from the directory of a diskette. Note that the file will not actually be removed. Only the directory entry is removed. CP/M will reallocate the area of the diskette used by the deleted file as needed. It is often possible to reinstate a file after its deletion—if you have the proper software tools, and provided no information has been added to the diskette since the file was deleted.

Format:

```
ERA x:filename.typ <cr>
```

This is the only form of the ERA command CP/M recognizes. The diskette identifier may be omitted if the file to be erased is on the current default drive.

Example:

```
ERA SE.ME <cr>
```

erases the file SE.ME from the current default diskette.

LOAD—*converts hex file to a .COM file*

The CP/M assembler generates an Intel hex-format file containing the resultant object program. To convert this code into an executable file, use LOAD. LOAD reads a hex-format file and translates it into a .COM file. .COM files load and begin execution at 0100 hex. If your assembly-language program did not originate at this address, LOAD will abort without creating a new file. Your original hex-format file is left intact.

Format:

LOAD x:filename <cr>

This is the only form that invokes LOAD (the drive identifier may be omitted if the default drive is used). Note that the file type is not specified; LOAD always assumes the extension .HEX for the file you specify.

Example:

LOAD UP <cr>

creates the file UP.COM from the file UP.HEX on the current default drive.

MOVCPM—*creates a different size CP/M*

If you wish to create a CP/M operating system that uses memory differently than the one provided with your system, you must first use the MOVCPM program. CP/M always uses the top 8K bytes of memory within the memory space you specify in the MOVCPM command. Thus, if you ask for a 48K CP/M system, CP/M will occupy the memory space between 40K and 48K.

Wherever CP/M is located, it will, nevertheless, start execution at the standard CP/M location (0100 hex).

When you run MOVCPM, the reallocated version of CP/M can either stay in memory, or be immediately executed. If you do not desire immediate execution of the newly created CP/M, you may save it by using the SYSGEN command.

Formats:

MOVCPM # <cr>

Creates a CP/M version for the memory size specified by #.

MOVCPM # * <cr>

Creates a CP/M version for the memory size specified by #, but does not execute it.

Example:

MOVCPM 48 * <cr>

creates a 48K CP/M system (about 40K of free memory beginning at location 0100 hex) and leaves it in memory. It is unlikely that you will want to use the other forms of the MOVCPM, as they create special problems when you use programs or commands that reset the system.

PIP—*Peripheral Interchange Program*

The Peripheral Interchange Program supplied with CP/M manipulates and moves files between devices. The devices CP/M recognizes can be either physical (i.e., real) or logical (i.e., assigned). Following is a list of all devices CP/M recognizes.

PHYSICAL:

x:	(disk drives)
TTY:	(console, reader, punch, or list)
CRT:	(console or list)
UC1:	(user console #1)
PTR:	(reader)
UR1:	(user reader #1)
UR2:	(user reader #2)
PTP:	(punch)
UP1:	(user punch #1)
UP2:	(user punch #2)
LPT:	(list)
UL1:	(user list #1)

LOGICAL:

CON:	(console)
RDR:	(reader)
PUN:	(punch)
LST:	(list)

SPECIAL:

NUL:	(sends 40 nulls)
EOF:	(sends **^Z**)
INP:	(input obtained for PIP by call to 0103 hex)
OUT:	(output obtained for PIP by call to 0106 hex)

NOTE

You may intermix logical, special, or physical device names with the disk file names on the PIP command line.

Formats:

PIP <cr>

This general form loads and executes PIP.

PIP destination=sources[options] <cr>

Normal form loads and copies "sources" to "destination." There may be only one destination device or file, but there may be any number of sources, each separated by commas.

[Options] in the second format represents the following special parameters:

B specifies block-mode transfer; data is buffered until **^S** command is received.

D*n* deletes all characters that extend past the column number specified.

E echoes all transfers to the console.

F removes form-feed characters from the data being transferred.

G*n* instructs PIP to get the file from the user area.

H specifies that Intel hex-data transfer is used. The source must be an Intel hex-format file.

I ignores 00 records in Intel hex transfers.

L translates uppercase to lowercase letters.

N adds line numbers to the data being transferred.

O specifies an object-file transfer.

P*n* indicates that page breaks occur every *n* number of lines.

R instructs PIP to read "system" files, if necessary.

Q string ^Z quits copying when it encounters the indicated string.

S string ^Z starts copying when it encounters the indicated string.

T*n* expands the tab character to *n* number of spaces.

U translates lowercase to uppercase characters.

V verifies that data has been copied correctly when the destination is a disk.

W instructs PIP to write over a file with the attribute of R/O without asking.

Z makes the parity bit zero on input for each ASCII character encountered.

NOTE

If you load and execute PIP by simply typing its name, you receive a "" prompt. You may type any valid "destination = sources [options]" line. That task completed, the "*" prompt returns. You may enter as many PIP commands as you wish in this fashion. Press ^C to exit PIP.*

Examples:

```
PIP <cr>
```

loads and executes PIP.

```
PIP A:=B:*.* <cr>
```

copies all files from drive B to drive A.

```
PIP EDREAM.DOC=B:ANYFILE.TXT <cr>
```

copies ANYFILE.TXT from drive B to a file named EDREAM.DOC on the current default drive.

```
PIP LST:=X:EWRENCH.TXT <CR>
```

copies the file EWRENCH.TXT to the list device.

```
PIP CON:=ETTE.BAS[N] <cr>
```

copies the file ETTE.BAS to the console device and displays line numbers before each line.

REN—*renames a disk file*

REN changes the name of a disk file. Renaming a file only changes the portion of the directory on the diskette that stores the file name and file extension. No other change is made to the diskette.

Format:

```
REN x:newname.typ=oldname.type <cr>
```

This is the only form allowed. "Newname" is the name identifying the file, "oldname" is the file's original name.

NOTE

When using the REN command, you may change the file extension as well as the file name.

Example:

```
REN AME.IT=OLDNAME.WAS <cr>
```

renames the file OLDNAME.WAS to AME.IT.

SAVE—*saves memory to disk file*

To save the memory beginning at 0100 hex, you use the SAVE command. Because of the way

CP/M saves information on diskette, memory must be saved in 256-byte increments—referred to as one page of memory.

Format:

SAVE *n* x:filename.typ <cr>

This is the only form of the SAVE command allowed. *n* is the number of pages (in decimal, not hex) of memory to be saved.

NOTE

Since programs initially load into memory beginning at 0100 hex, be careful that you don't execute a program between the time you create something you wish to save and the time you save it. Also, you cannot use the SAVE command more than once consecutively, because sometimes the memory image changes during the SAVE operation.

Example:

SAVE 7 CHICAGO <cr>

saves seven pages (7*256 bytes) in a file named "CHICAGO." Note that the file does not have to have an extension.

STAT—*displays statistics*

The STAT command performs different functions. It can report on the current status of disk files. Such a report gives the following information:

SIZE: the virtual file size in records.

RECS: the number of records in a file.

BYTES: the size of the file in kilobytes.

EXT: the number of logical extents the file occupies.

Acc: the file access (read-only or read/write).

If you did not ask for a report on the current status of individual files, but asked for a report on the entire diskette, one line listing the amount of free disk space in kilobytes would be displayed instead.

A second use of STAT is to display the current status of devices.

A third use of STAT is to reassign devices, or assign a particular status (such as R/O) to a device.

Formats:

STAT x:filename.typ

returns statistics about specific file (or files if "*" or "?" parameters are used).

STAT x:DSK: <cr>

lists status of disk device.

STAT VAL: <cr>

lists possible device assignments.

STAT DEV: <cr>

lists current device assignments.

STAT logicaldevice=physicaldevice <cr>

assigns physical device to logical device.

Examples:

STAT <cr>

displays the amount of space remaining on the current default drive.

STAT *.BAS <cr>

displays the status for all files with the extension of .BAS.

STAT ISTICS.ON <cr>

lists status for the file ISTICS.ON.

STAT B:DSK: <cr>

displays the status of disk drive B:.

SUBMIT—*is the batch submit utility*

CP/M can accept command lines from a disk file instead of the console. The file containing your commands may have any valid CP/M file name, but must have the file extension .SUB to work correctly.

SUBMIT will accept parameters at the time it executes. Each parameter is identified within the SUBMIT file by a dollar sign ($), followed by a number. On the command line invoking SUBMIT, parameters you enter are assigned to SUBMIT variables in sequential order.

NOTE

CP/M 2.2 also includes another batch utility, called XSUB, which allows you to provide individual characters from a disk file.

Format:

SUBMIT x:filename parameters <cr>

This is a general form (parameters may be omitted). If present, parameters are separated by spaces.

Example:

If you type **"SUBMIT X:filename a:junkfile.ugh b:<cr>"** and the SUBMIT file (x:filename) has the following commands in it:

```
PIP $2:BACKUP.FYL=$1
DIR $2
ERA $1
```

The following would be executed:

```
PIP B:BACKUP.FYL=A:JUNKFILE.UGH
DIR B:
ERA A:JUNKFILE.UGH
```

SYSGEN—*creates a CP/M system image*

One way to place a copy of CP/M on the system tracks of a diskette is by using the SYSGEN utility; you can also use the COPY or BACKUP utility programs. SYSGEN gets a copy of CP/M either from memory after executing a MOVCPM command, or from a diskette containing the CP/M operating system.

The SYSGEN utility does not alter the directory or data saved on the diskette involved; it only writes information on the reserved system tracks.

Format:

SYSGEN <cr>

This is the only form allowed for invoking the SYSGEN command. Once invoked, SYSGEN prompts the user through the steps needed to save the CP/M system onto diskette. These prompts are:

***SOURCE* DRIVE (A or B).** Reply with a valid disk identifier if you wish to get the CP/M system from a diskette, or press RETURN if CP/M is in memory following a MOVCPM command.

Put *SOURCE* diskette in drive x, then press RETURN. This prompt indicates that CP/M expects you to place the diskette with the system from which you wish to copy on drive x.

Put *DESTINATION* diskette in x, then press RETURN. In this case, x represents the drive opposite that designated as source. Insert the diskette in the indicated drive and press RETURN to save the CP/M system.

***DESTINATION* (A, B or RETURN to exit).** At this point you can copy the system to another diskette in the drive of your choice, or press RETURN to leave the SYSGEN program.

TYPE— *displays a disk file in ASCII*

The TYPE command displays any file containing only displayable ASCII characters. TYPE does not format displayed files, so be prepared for long bunches of data without carriage returns to wrap around from one line to the next. If a file is longer than the screen can display at one time, the screen will scroll. You can stop scrolling by pressing **^S**, which pauses the display, or you can stop scrolling completely by pressing **^C**.

TYPE will display WordStar files you created using the **N** (nondocument) command. WordStar files created by the **D** (document) command are not displayed as entered, since they contain special control characters. Files created by programs running under CBASIC2 generally contain all ASCII data and are displayed as entered.

Format:

```
TYPE x:filename.type <cr>
```

This is the only form of TYPE command allowed.

Example:

```
TYPE RIGHT.TER <cr>
```

presents an ASCII display of the file RIGHT.TER.

USER— *changes disk access in the user's area*

CP/M allows you to specify any of 16 "user areas" and to assign files to them. These areas are not physical assignments on a diskette; they are only parameters stored in the directory. CP/M user areas are a security measure.

CP/M will only allow you access to "system" files (accessible to all user areas), or to the files that contain the user attributes matching your current user-area assignment. CP/M commands that do not involve system files will only operate files with user attributes matching your current user-area assignment. You may change user areas anytime you are at the CP/M command level and see the CP/M prompt. Correct and logical use of user areas makes sure that only authorized operators or programs access certain files.

NOTE

If you do not assign a user area, CP/M assumes that all files have user attributes of zero.

Format:

USER ***n*** **<cr>**

This is the only form of the USER command. *n* may be any number between 1 and 15, inclusive; it indicates the user area assignment you wish to make.

Example:

```
USER 0 <cr>
```

sets the current user area to zero.

XSUB— *batch submit utility*

SUBMIT tells CP/M to receive further commands from a disk file. XSUB can be used as a command within a SUBMIT file to indicate that individual characters pass to CP/M (or the program currently in control of the system) for processing.

XSUB only works if the program to which you wish to pass characters uses the CP/M-buffered console-input routines in a normal fashion (i.e., by your placing 0A hex in the C register and calling 0005 hex). Some programs, notably MICROSOFT BASIC-80 and several word processing packages, do not use these routines. XSUB passes individual characters to any program Digital Research supplies.

XSUB is only used as a command within a SUBMIT file, and it must be the first command within that file. The valid format as it appears in a SUBMIT file follows:

```
XSUB
MOVCPM $1 2$
SYSGEN
 ^M
$3
 ^M
 ^M
```

The above file generates and saves a new CP/M system; the size and disk drive of the system to be saved are entered on the SUBMIT command line (e.g., SUBMIT SUBFILE 48 * B: <cr>).

WordStar

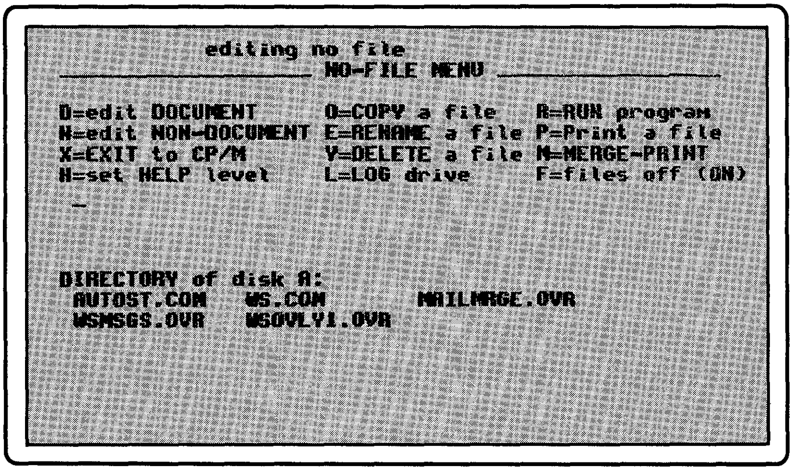
Getting WordStar Started

In order to start WordStar, place the WordStar diskette in drive A and a formatted diskette in drive B, then press **RETURN**. The diskette in drive B will be used to store text files you generate.

Once WordStar is running, the message

editing no file

appears at the top of the screen with a menu of WordStar operations below it. Each operation is identified by a single letter which you press to select the desired operation. Pressing **CTRL** is not necessary when you enter commands from the No-File menu—in fact this is the only menu from which WordStar uses non-control letters as commands. Here is the No-File menu you will see:



```
              editing no file
__________________ NO-FILE MENU __________________

D=edit DOCUMENT     O=COPY a file   R=RUN program
N=edit NON-DOCUMENT E=RENAME a file P=Print a file
X=EXIT to CP/M      Y=DELETE a file M=MERGE-PRINT
H=set HELP level    L=LOG drive     F=files off (ON)
 _


DIRECTORY of disk A:
 AUTOST.COM  WS.COM       MAILMRGE.OVR
 WSMSGS.OVR  WSOVLY1.OVR
```

Initiate the following operations by pressing the letter indicated:

CREATE OR EDIT A DOCUMENT—*creates or retrieves a document file*

D is used for general word processing. A file name is requested. Supplying a new file name causes a new file to be created under the specified name. Entering an existing file name causes the specified file to be fetched from the current drive and displayed on the screen.

EDIT A NON-DOCUMENT—*creates or retrieves a nontext file*

N is used to create or edit a data file for merge-printing. The non-document mode is generally used by programmers to create source-program files.

MERGE-PRINT—*initiates merge-printing of a file*

M initiates a merge-print operation. Merge-print merges files during printing, thereby generating form letters, boilerplate text, mailing lists, and large documents.

DISPLAY DIRECTORY—*toggles appearance of file directory*

F turns the file directory OFF or ON again. The menu displays the current status. When the file directory is ON, the names of all text files on the logged drive are displayed.

CHANGE LOGGED DISK DRIVE—*activates alternate disk drive*

L selects the "logged" or "active" drive; WordStar assumes that all text files are on the diskette in this drive. When you press **L**, a message asks you to select the drive to activate. Type the drive letter followed by a colon, and press **RETURN**.

RUN A PROGRAM—*runs a program from the No-File menu*

R runs a program without exiting from WordStar. You can execute CP/M programs, such as XDIR and STAT, by supplying their file names to the COMMAND? prompt.

HELP SET—*establishes the level of assistance displayed on the screen*

H selects the level of information the menus display. As you become more experienced with WordStar, you may want to decrease the level of assistance displayed in the menus at the top of the screen.

EXIT TO SYSTEM—*relinquishes control to CP/M*

X exits WordStar and returns control to the CP/M operating system.

PRINT A FILE—*toggles printing of a named file*

P initiates and halts printing of a text file. The name of the file to print is requested, followed by prompt questions regarding the print operation. The current print status is displayed on the screen.

DELETE A FILE—*deletes the specified file*

Y deletes a file from the diskette in the currently active drive. A file name is requested, and the named file is subsequently deleted.

FILE COPY—*copies a file*

O copies a file from source to destination. The name of the file to be copied, and the name of the file where the copy is to be transferred are requested. If the destination file name exists, it will be erased unless the command is abandoned.

FILE RENAME—*renames a file*

E assigns a new name to a specified file. The existing file name is requested; when the name is supplied, the new file name is requested. Supplying the new file name and pressing **RETURN** completes the renaming process.

Block Operations

An entire section of text may be moved, copied, deleted, or written to another file. All these operations are performed by block commands. To manipulate a section of text, you must first "block" it. To block a section of text, place the cursor at the beginning of the text to be blocked and type **^KB**; a < B > will appear on the screen to indicate the position of the beginning marker. Next, move the cursor to the end of the text you wish to enclose within the block and type **^KK**. The entire marked block will be displayed in half intensity so it is easy to distinguish.

You can set a beginning and end block marker in the middle of a paragraph to manipulate a sentence, or in the middle of a sentence to manipulate a word.

You can have only one marked block in your text file at a time. A new **^KB** or **^KK** command will replace any previous block beginning or end marker, if one exists. The marked block is always the implied source for any block operation. Those block commands that require a destination assume the cursor position as the location. Following are the BLOCK commands:

BLOCK MARK BEGINNING—*defines the beginning of a block*

^KB marks the beginning of a block at the cursor position. The beginning block marker is displayed as a < B >. Alternatively, **^KB** hides the displayed marker.

BLOCK MARK END—*defines the end of a block*

^KK

marks or hides the ending of a block. It is used with the beginning block marker to enclose a section of text so you can perform block operations on it.

BLOCK COPY—*copies a block from source to destination*

^KC

copies the currently marked block of text to where the cursor is positioned. The original text remains unaltered. The block markers move with the text.

To copy a text block, place the cursor at the desired destination and type **^KC**. The current block is then copied to the cursor position. The cursor stays at the beginning of the copy.

The block markers are transferred with the copy of the block and remain displayed. The command **^KH** hides the block markers following a block operation.

You may make as many copies of the block as you desire by typing **^KC** as many times as necessary. Copies may be made in different locations by moving the cursor to the desired position between copy commands.

Using **^QV** after a block copy returns you to the source of the block.

After copying a block, you may use the REFORM command, **^B**, to reformat the text.

NOTE

Unlike Block markers, Place markers do not move with the marked block.

BLOCK DELETE—*removes a block from a file*

^KY deletes the currently marked block. The block must be visible (not hidden) for **^KY** to delete it. Since large amounts of text can be deleted by accident, it is recommended that a block be hidden when you're not operating on it.

When a block is deleted, both block markers are hidden and left at the position of the deleted text. You can use the **^QV** command to move the cursor to the delete location.

BLOCK MOVE—*moves a block to the desired position*

^KV moves a block to the cursor position. Place the cursor at the desired destination and type **^KV**. The cursor is left at the beginning of the moved text.

The block markers move with the block and remain displayed. The command **^KH** hides the block markers after the block operation.

Following a BLOCK MOVE, the **^QV** command returns to the source of the block. Also, the REFORM command, **^B**, can be used to reformat text after the block is moved.

NOTE

There is a limit to the size of block you can move or copy. If a BLOCK-TOO-LONG error occurs, divide the block into smaller sections and perform the BLOCK operation on each section.

BLOCK WRITE TO FILE—*sends content of block to a named disk file*

^KW

transfers the contents of a block to another file on the logged drive, or drive specified. The name of the file where the block is to be transferred is requested, the contents of the block are then written to the named file. The entire contents of any existing file with the same name will be erased. To prevent unintended erasures, if the file name that is specified already exists, WordStar responds:

```
FILE d:name.typ EXISTS....OVERWRITE? (Y/N):
```

Pressing **Y** causes the BLOCK WRITE function to write over the previous contents of the specified file; pressing **N** causes the file name to be requested again.

BLOCK WRITE lets you extract text from a document and save it as a separate document. BLOCK WRITE may also be used to move a section of text large distances within a file: write the block to a temporary file, then move the cursor where you want the block moved and read the file containing the block with **^KR**.

BLOCK HIDE/REDISPLAY—*toggles display of a block*

^KH hides a text block so that no BLOCK operations can be performed on it. Though hidden, the text remains blocked until another section of text is blocked. Alternatively, the **^KH** command redisplays the hidden marked block making it once again subject to BLOCK operations.

Changing the Logged Disk Drive

The standard procedure for starting WordStar involves activating the disk drive where files will be stored. After you place the WordStar program's diskette in drive A and press **RETURN**, the A drive is activated (logged). What this means is that, unless you append the drive identifier B:, any files you create or edit will be stored on the program's diskette in drive A.

"Logging onto" the B drive provides the most convenient method of managing your files. You should log onto drive B before issuing any other command from the No-File menu. Subsequently, all your files will be stored and retrieved from the logged drive B. Here is a summary explanation of the command used to log drives:

CHANGE LOGGED DISK DRIVE—*activates alternate disk drive*

^KL (**L** from the No-File menu) changes the logged drive where files will be stored. The currently active drive is identified, and you are asked which drive to log. The file directory reflects the contents of the currently active drive.

After you type **^KL** (or **L**), the name of the drive where files are currently being stored is displayed. You are asked to identify the disk drive to activate. Enter the drive letter followed by a colon (**A:** or **B:**). Files will subsequently be written to, and read from this drive.

Remember, WordStar will always read from and write to files on the currently logged disk drive unless you explicitly specify the other drive by appending a disk identifier (A: or B:) to the front of a file name.

Cursor Motion

Cursor-motion commands move the cursor within a document. Basic cursor movement is accomplished using six control characters positioned according to the direction they move the cursor:

```
        ^E
^A  ^S  ^D  ^F
        ^X
```

The arrow keys, as well as a variety of other commands, also move the cursor to specific locations in the file. Following are the cursor-motion commands:

CURSOR LEFT A CHARACTER—*moves cursor left one character*

^S, ^H,

or $\boxed{\leftarrow}$ each move the cursor one character position to the left. The cursor will move to the end of the preceding line if it is located at the beginning of the current line. These commands are used to backspace over characters and to make corrections.

CURSOR LEFT A WORD—*moves cursor left a word*

^A moves to the beginning of the word to the left of the cursor. A word is a string of characters separated by a punctuation mark (. , : ; ! ?, a space, or a carriage return).

CURSOR LEFT EDGE OF SCREEN—*moves cursor to left edge of screen*

^QS moves the cursor to the far left of the screen.

CURSOR RIGHT A CHARACTER—*moves cursor right one character*

^D
or [→] move the cursor one character position to the right. If the cursor is positioned at the end of the current line, it will go to the beginning of the next line.

CURSOR RIGHT A WORD—*moves cursor one word to the right*

^F moves the cursor to the beginning of the next word.

CURSOR RIGHT END OF LINE—*moves cursor to the end of a line*

^QD moves the cursor to the end of the current line of text. To get the cursor to move beyond the actual characters at the right end of a line, you must space or tab over to the desired column.

CURSOR UP A LINE—*moves the cursor up one line*

^E or **↑** moves the cursor to the next line above. The cursor remains in or near the same column.

CURSOR DOWN A LINE—*moves the cursor down a line*

^X or **↓** moves the cursor down to the beginning of the next line in the document. The cursor remains in the same column, but will move to the left to avoid landing beyond an end of a line; the cursor will also jog around print-control characters when necessary.

CURSOR TO BEGINNING OF FILE—*positions cursor at file's beginning*

^QR moves the cursor to the beginning of the file being created or edited. If the document involved is a large one, and you are near the end, use the FILE SAVE command, **^KS**, since it is faster and uses up less temporary diskette file space.

CURSOR TO END OF FILE—*positions cursor at file's end*

^QC moves the cursor to the end of the file. The cursor ends up in the position following the last character of the document.

CURSOR TO SCREEN BOTTOM—*moves the cursor to the screen bottom*

^QX moves the cursor to the bottom of the text displayed on the screen. The cursor remains in its current column position.

CURSOR TO SCREEN TOP—*moves the cursor to the screen top*

^QE moves the cursor to the top of the currently displayed text.

CURSOR TO BLOCK BEGINNING—*moves cursor to block's beginning*

^QB moves the cursor to the beginning of the currently marked block. If the text block is hidden, the beginning marker is redisplayed.

CURSOR TO BLOCK END—*moves cursor to the end of a block*

^QK moves the cursor to the end of the currently marked text block. If the block is hidden, the end-block marker will be redisplayed.

CURSOR TAB—*moves the cursor to the next tab stop*

^I or TAB advances the cursor to the next tab stop when INSERT is OFF, or inserts spaces up to the next tab stop when INSERT is ON.

CURSOR TO PLACE MARKER—*moves cursor to indicated place marker*

^Q0 to ^Q9 moves the cursor to any of the ten place markers. Each PLACE MARKER is identified by a number. To send the cursor to a previously set place marker, type **^Q** and the number of the marker (0–9).

CURSOR FIND—*moves cursor to a specified string of characters*

^QF moves the cursor to a specified word or phrase. The word or phrase is requested through a prompt. The cursor moves to the first occurrence of the specified word or phrase. **^QF** can be used with the FIND/REPLACE AGAIN command, **^L**, to find all occurrences of a given word. When you type **^QF**, WordStar asks for the word to be located using the prompt

```
FIND?
```

This question will appear below the menu, moving the top of the file display area down one line. Respond by typing any sequence of characters that you wish to locate, then press **RETURN**. (This function is further discussed in the FIND command.)

Send the cursor to the position it occupied before the last FIND or REPLACE by using the CURSOR-TO-LAST-FIND command, **^QV**.

CURSOR TO PREVIOUS POSITION—*moves cursor to previous position*

^QP

moves the cursor to where it was when the last command was issued. The **^QP** command is frequently used to continue editing after saving a file with **^KS**. This command is also used following a paragraph REFORM, **^B**, to return to the position where you were making editing changes.

CURSOR TO SOURCE—*moves cursor to source of last BLOCK or FIND operation*

^QV

moves the cursor to the position it occupied before the last FIND or REPLACE operation, or to the source of the last block operation.

Deletions

DELETE A CHARACTER RIGHT—*eliminates characters at the cursor*

^G deletes one character at the cursor position. Text, spaces, and carriage returns are deleted. Characters to the right of the deletion are drawn to the cursor position and replace the deleted characters.

DELETE A CHARACTER LEFT—*eliminates characters to the left*

^_ deletes one character to the left of the cursor. When the left end of the line is encountered, the cursor eliminates the carriage return and starts deleting on the next line up.

DELETE A WORD RIGHT—*eliminates a word from the right*

^T deletes a word or portion of word to the right of the cursor. If the cursor is in the middle of a word, the part of the word to the right of the cursor is deleted.

You can delete spaces between words by placing the cursor between the characters you wish to join and pressing **^T**. If the cursor is set at the end of a line, the carriage return and any following spaces will be deleted.

DELETE TO BEGINNING OF LINE—*deletes to beginning of line*

^Q^- deletes all characters from the cursor position leftward to the beginning of the line. This command, however, does not delete the carriage return at the end of the line.

DELETE TO END OF LINE—*deletes to end of line*

^QY deletes all characters to the end of the line from where the cursor is positioned. Carriage returns and any overprint lines will not be deleted at the end of this line.

DELETE A LINE—*eliminates the current cursor line*

^Y deletes the entire line containing the cursor. The line below moves up and takes the place of the deleted line. Screen continuation lines and associated overprint lines are also deleted.

DELETE A BLOCK—*eliminates the currently marked block*

^KY deletes the currently marked block of text. (See BLOCK DELETE.)

DELETE A FILE—*deletes a specified file*

^KJ (or Y from the No-File menu) asks for the name of the file to be deleted and then deletes the specified file.

Display Commands

DISPLAY PAGE BREAK—*toggles appearance of page break line*

^OP hides or displays the line used to illustrate where one page ends and the next begins. It also changes the status line to display the number of the cursor character (FC=cursor character number from beginning of file) and the line number (FL=line number from beginning of document). The page break line is ON until the **^OP** command is issued. The current status of the page break line is displayed on the **^O** prefix menu.

DISPLAY PRINT-CONTROL CHARACTERS—*toggles appearance of print-control characters*

^OD hides or displays print-control characters. Print-control characters are initially displayed while you're entering them into text; however, the **^OD** command can conceal them from the screen display and thus show how your text should look when it is printed. You can examine the current status of the print-control feature on the **^O** prefix menu.

DISPLAY RULER LINE—*toggles appearance of ruler line*

^OT displays or hides the ruler line. The ruler line is normally displayed until hidden.

DISPLAY DIRECTORY—*toggles appearance of file directory*

^KF (or F from the No-File menu) hides or displays a directory listing all files contained on the diskette in the currently logged drive. The prefix menu indicates whether the directory display is ON or OFF.

File Manipulations

You can enter a file or exit WordStar from the No-File menu. The No-File menu also provides options that allow you to copy, delete, read, or rename files using the appropriate command. You can also undertake these file manipulations from within an open file. To manipulate files between drives, append a drive identifier followed by a colon to the file name. You can toggle the directory of the currently active drive ON or OFF by using the FILE DIRECTORY command **^KF**, or temporarily display it during a file manipulation by pressing **^F** after initially issuing a file manipulation command. Following are the commands used to manipulate files:

EDIT A DOCUMENT—*creates or retrieves a file*

D creates a new file or opens an existing file for editing. A file name is requested. If the specified file does not exist, then a new file is created. If a file with the specified name exists, it is retrieved so you can edit it. The file may be on another drive, in which case the drive letter and a colon must precede the file name.

EDIT A NONDOCUMENT—*creates or retrieves a nontext file*

N creates or retrieves a nontext file for editing. Dynamic pagination is disabled, and a different set of defaults is in effect. This command is typically used to prepare input for other text formatters, to enter data for application programs, or to edit program source files. Do not use the **N** command for general word processing.

NOTE

Programmers should not reform (^B) the contents of nondocument program files.

EDIT ABANDON—*closes file without saving current version*

^KQ abandons the file being created or edited without saving a copy. A backup copy will remain only if the file is being edited after previously being created. If the file is just being created, no copy of it will exist following this command.

EXIT TO SYSTEM—*relinquishes control to CP/M*

^KX

leaves WordStar and returns control to the CP/M operating system. When you issue **^KX**, the current file is saved before leaving WordStar. (Similar to the No-File command **X**.)

FILE COPY—*copies a named file*

^KO

(or O from the No-File menu) copies a file from its source to a specified destination. Prompts ask for the name of the file to copy and the destination where the copy is to be sent. You can copy from one drive to another by preceding the file name with the letter and colon of the drives involved. (Performs the same function as the CP/M program PIP.COM.)

When you type **^KO** or **O**, the following prompts occur:

```
NAME OF FILE TO COPY FROM?
NAME OF FILE TO COPY TO?
```

Enter the name of the file that you wish to copy and press **RETURN**. Next, enter the name of the file where the copy is to be transferred. To make copies from one drive to another, add the appropriate drive identifier and a colon to the front of the file name, (e.g., **A:copyfrom, B:copyto**).

FILE DELETE—*erases a named file from the directory*

^KJ (or Y from the No-File menu) erases a named file from the current file directory. A drive identifier specifies a file on the inactive drive. This command performs the same function as the CP/M command ERA (also see DELETE A FILE).

FILE READ—*reads a named file into the currently open file*

^KR transfers and inserts the contents of a specified file to the cursor position of the file being created or edited. A file name is requested by WordStar as follows:

```
NAME OF FILE TO READ?
```

The contents of the indicated file is inserted into the current file at the cursor position.

FILE RENAME—*renames a disk file*

^KE (or E from the No-File menu) asks for the name of the file to rename and then assigns the new name to the specified file. These prompts occur:

```
NAME OF FILE TO RENAME?
NEW NAME?
```

To change the name of a file, supply its name, and press **RETURN**. Then enter the name with which you want the file identified in the future.

This command performs the same function as the CP/M command REN.

Find Functions

The FIND command, **^QF**, moves the cursor to a given word or phrase within the file. The FIND, REPLACE command, **^QA**, locates a specific word or phrase and replaces it with another. After the desired word or phrase has been located and/or replaced, you may proceed to the next occurrence of the word or phrase by issuing the FIND/REPLACE AGAIN command, **^L**.

FIND—*locates a given word or phrase*

^QF

finds the first occurrence of a specified word or phrase. A prompt asks for the word or phrase to search for:

```
FIND?
```

Reply by typing the word or phrase you wish to locate, then press **RETURN**. The cursor moves to the indicated word or phrase. Certain options are defined in the FIND REPLACE command below.

FIND, REPLACE—*locates and replaces a given string with another*

^QA

finds the first occurrence of a specified word or phrase and replaces it with another. Prompts ask for the word or phrase you're searching for, as described above. After you have entered the string that you wish to locate and have pressed **RETURN**, WordStar asks:

REPLACE WITH?

Respond by entering the replacement characters and pressing **RETURN**. WordStar then asks:

OPTIONS (? FOR INFO)

The "OPTIONS" question allows you to specify certain options, such as matching whole words only, ignoring the distinction between uppercase and lowercase letters, or searching backwards instead of forwards. A question mark (?) will display a list of FIND options; use it to help refresh your memory. You can ignore the "OPTIONS" question by pressing **RETURN**, or you can answer it with one or more of the following option codes:

> #, when specified with the FIND command, locates the #th occurrence of the specified word or phrase. Used in the FIND, REPLACE command, it locates and replaces the specified word or phrase that number of times.
>
> **G**, when used with the FIND, REPLACE command, replaces every occurrence of the specified word or phrase from the cursor position to the end of the file. A (Y/N) prompt allows selection of each replacement as it appears. When used with the FIND command, it will search for the last occurrence.
>
> **N** replaces words or phrases without asking the (Y/N) question.

U causes the find or replacement operation to ignore the distinction between uppercase and lowercase letters.

W matches only whole words during a FIND or REPLACE.

^P (control character) Control characters can be entered in response to the OPTIONS question. Some specialized ones follow:

B Instead of the search progressing towards the end of the file, as it usually does, this option causes the file to be searched backwards to the beginning of the file.

A matches any single character.
S matches any character other than a letter or digit.
O precedes a character so that a match will occur with any character but the one following **^O**.
N causes a match to be made with a carriage return or line feed.

NOTE

You can eliminate the "OPTIONS" question by pressing ***ESC*** *following either the "FIND" or "REPLACE" questions.*

After you've issued the FIND, REPLACE command (**^QA**), WordStar will search for the word or phrase to replace. On finding the word or phrase, WordStar will display the following prompt at the upper right side of the screen:

```
REPLACE (Y/N)?
```

The cursor flashes on and off to indicate that a decision is required. If you want to replace the word or phrase that is located, press **Y** for YES. If you do not wish to replace that particluar occurrence of the word or phrase, press any other key. To repeat the most recent REPLACE command from the current cursor position, use the abbreviated FIND/REPLACE AGAIN command, **^L**.

NOTE

You may use the INTERRUPT command, **^U**, *to stop a FIND or REPLACE operation while it is in progress.*

FIND/REPLACE AGAIN— *continues a previous FIND or REPLACE function*

^L repeats the most recent FIND or REPLACE command and supplies identical responses for the options.

Flag Characters

All the columns on the screen, except the rightmost column, are available for text. This column is reserved for FLAG characters that indicate the status of text on the current file line as follows:

(blank) in the last column indicates that the line ends with a soft carriage return. This condition may be changed following a WORD WRAP or REFORM operation.

< indicates that the line ends with a hard carriage return. WORD WRAP or REFORM operations do not change this line break.

- means that the following line will be printed over the current line. This PRINT function creates special effects.

• indicates that the current screen line is below the existing text. This character will also appear at the end of the last text line if there is not a carriage return at the end of this line.

: appears if the current text line is above or before the beginning of the document.

+ indicates that the next screen line is a continuation of the initial line.

P appears only when the PAGE BREAK display is on: it indicates that a new page begins with the next line.

? is displayed when a line contains an unrecognized or possibly erroneous DOT command. It also appears while a DOT command line is being typed. You may ignore this character until entry is complete.

J indicates that the line ends in a line feed without a carriage return. This format is nonstandard and is never created during normal WordStar usage.

M indicates a line contains a MERGE-PRINT DOT command.

HELP Commands

HELP menus are shown at the top of the screen. You can eliminate them by degree as you become more adept at using the system. Also, WordStar provides further information to assist you in learning; certain commands display detailed information about the more involved WordStar functions. The **^J** prefix menu explains the following list of subjects:

^JD explains print directives such as DOT commands and PRINT controls.

^JI explains command index for entering text.

^JM explains margins, line spacing, justification, and tabs.

^JB explains paragraph reform.

^JP explains place markers.

^JV explains moving text.

^JR explains the ruler line.

^JS explains the status line.

^JF explains flag characters.

HELP SET— *establishes the level of assistance displayed on the screen*

^JH, or H from the No-File menu, displays the current help level and requests a new setting. You can set the help level between 0, the least amount, or 3, the greatest amount of help. The amount of information displayed on the help menus corresponds to the help-level setting.

After you type the HELP SETcommand, a description of help levels and the current help-level setting are displayed. The display requests a new help-level setting.

As you gain experience using WordStar, you may reduce the level of assistance to coincide with your experience and ability. Help level 0 provides the least assistance and gives you the most screen area for file display.

Hyphens

WordStar has two kinds of hyphens: the soft hyphen, which indicates a syllable break, and the hard hyphen, which separates words or phrases.

A soft hyphen at the end of a text line separates a word that is too long to fit on the current line. The hyphen divides the word into syllables and continues it on the next line. The soft hyphen prints only if the divided word appears at the end of a line; if the word ends up in another position following some operation such as a REFORM, the hyphen will not be printed.

Generally, you enter soft hyphens by using the HYPHEN HELP feature, but you may enter them explicitly by turning ON the SOFT-HYPHEN ENTRY. Soft hyphens that you type when the SOFT-HYPHEN ENTRY is ON will divide a word only if it appears at the end of a line.

Hard hyphens, on the other hand, are used whenever a fixed divider is required between characters, strings, or phrases. A hard hyphen will always be printed, no matter where it appears in the text. A hard hyphen is entered automatically if the SOFT-HYPHEN ENTRY is OFF and HYPHEN HELP is not engaged. **^P-** unconditionally enters a hard hyphen.

To distinguish soft from hard hyphens, type **^OD**, which turns the print-control display ON and OFF. When this display is OFF, soft hyphens will not show up in the file document.

HYPHEN HELP— *toggles HYPHEN HELP feature ON or OFF*

^OH

turns HYPHEN HELP ON or OFF. When ON, the PARAGRAPH REFORM process pauses and positions the cursor at each instance where a word can be hyphenated. The **^O** prefix menu displays whether HYPHEN HELP is ON.

HYPHEN HELP checks that the word contains two syllables, and selects the proper position for the hyphen. You can then decide if the word should or should not be hyphenated at the selected position.

When using HYPHEN HELP, make sure that margins, line spacing, and justification are properly selected. Place the cursor at the beginning of a paragraph and press **^B**. When the hyphen position is located, the following message is displayed:

```
TO HYPHENATE, PRESS -. Before
pressing the -, you may move the cursor:
^ S = cursor left, ^ D = cursor right.
```

You can enter a hyphen at the suggested location, or you can move the cursor to the position where you want the hyphen to appear. If you do not want to hyphenate the word at all, press **^B** and the REFORM process will continue down the text.

SOFT-HYPHEN ENTRY— *toggles interpretation of hyphens*

^OE

turns SOFT-HYPHEN ENTRY ON or OFF. When ON, hyphens are temporary and will not be printed unless they fall at the end of a line. Soft hyphens are highlighted, but the **^OD** command causes only hyphens that will be printed to be displayed in half intensity.

Interrupt Execution

INTERRUPT— *stops command execution*

^U

stops any commands currently in progress. You can also enter this command in response to prompt questions, such as "FIND?", to abort the command making the request. When you press **^U**, the following message is displayed:

```
* * * INTERRUPTED * * *
```

All commands issued before the INTERRUPT are aborted and must be reentered if needed.

Insertions

INSERTION— *toggles character insertion ON and OFF*

^V

toggles the INSERTION function to either insert text to the left of the cursor position or replace text at the current cursor position. When INSERTION is ON, it inserts typed characters to the left of the cursor while the cursor moves text to the right. When INSERTION is OFF, typed characters replace those at the cursor position. The status line displays the current state of insertion.

You can determine if INSERTION is ON or OFF by looking at the STATUS LINE at the top of the display for the words INSERT ON. (Use **^<LEFT ARROW>** to see the INSERT message at the upper right of the screen.)

INSERT CARRIAGE RETURN— *establishes a fixed carriage return*

^N
or ^M

insert fixed carriage returns. Any text to the right of the cursor moves to the beginning of the next line. Neither WORD WRAP nor any other operation can alter a hard carriage return. A hard carriage return will always appear in the printed version of a document.

The difference between these two commands is that **^M** moves the cursor with the text, whereas **^N** leaves the cursor in its current position.

Layout

You can format a document in many ways using the WordStar layout commands. These commands allow you to control the way text is displayed. All the following commands can affect already existing text when they're used with the PARAGRAPH REFORM command:

REFORM PARAGRAPH— *reorganizes a paragraph with new specifications*

^B

reforms the paragraph below the cursor so that words are spaced evenly after editing. You can also use the REFORM command to change margins or line spacing, justify or unjustify text, or assist in hyphenation.

JUSTIFICATION— *toggles interpretation of text alignment*

^OJ

when ON, aligns text with the right margin. When JUSTIFICATION is OFF, lines of text end in various columns. The **^O** prefix menu displays the current state of justification. You determine if this function is ON or OFF by pressing **^O** and looking at the menu.

WORD WRAP— *toggles carriage return requirements between lines*

^OW

turns the WORD WRAP feature ON or OFF. This feature, which is normally ON, allows entry of text without the need for carriage returns except between paragraphs. Turn WORD WRAP OFF if you want to terminate every line with a carriage return as you would on an average typewriter.

CENTER CURSOR LINE— *centers current line between margins*

^OC

centers the line containing the cursor within the margins. This command is generally used to center headings. To use this command, place the cursor anywhere on the line you wish to center and type **^OC**. This command deletes any spaces and tabs set at the beginning of the line, then enters the appropriate number of hard spaces needed to center the line.

LINE SPACING— *sets the line spacing*

^OS

sets the number of blank lines that separate text lines. When you type **^OS**, WordStar will request a number between 1 and 9 that represents the number of carriage returns placed between text lines.

The LINE SPACING command also determines the number of line advances following every RETURN. You can change line spacing at any time by entering **^OS**, and reformatting the document (using the REFORM command).

RULER LINE— *toggles display of the ruler line*

^OT

toggles the display of the RULER LINE below the menu. This dotted line shows the margin and tab formatting that is in effect. Left and right margins are illustrated by an "L" and "R". Exclamation marks indicate variable tab stops. Decimal tab stops are shown as # signs.

When a margin is set at a tab stop, the tab symbol is displayed. If the margin is temporarily moved in with the PARAGRAPH TAB command, **^OG**, the ruler display will show the extent of the temporary margin.

Tabs set outside the margins are not displayed until the margins are released or WORD WRAP is OFF. When the margin is set larger than the screen display, the RULER LINE doubles to show the current margin setting. You can hide the RULER LINE by typing the DISPLAY/HIDE

RULER LINE command **^OT**. When the RULER LINE is hidden, a line of S's is displayed to separate the file directory from the file document.

You can specify margins in a text file by using the RULER LINE to identify where to set the margins. To enter a RULER LINE that will set the margins, type a line into the document with an exclamation point (!) at each column where a tab should be set, a number sign (#) at each column where a decimal tab should be set, and a hard hyphen (-) in every other column between the desired left and right margins. You can then enter this RULER LINE into the document by placing the cursor anywhere in the line and typing the MARGINS FROM FILE LINE command, **^OF**.

When you have entered this line, the desired tabs and margins will be set while all other default settings are cleared. The RULER LINE may be kept from appearing in the printed document by preceding it with two periods (see PRINT DOT COMMENTcommand).

NOTE

The ^O menu shows whether the following features are currently ON or OFF:

HYPHEN HELP
VARIABLE TABBING
PAGE BREAK DISPLAY
WORD WRAP
JUSTIFICATION
PRINT-CONTROL DISPLAY
SOFT-HYPHEN ENTRY
RULER DISPLAY

Margin Arrangement

MARGIN LEFT— *establishes the left margin*

^OL sets the left margin between column 1 and column 240. You can specify the left margin setting by entering the column number of the new margin, or pressing ESC to set it at the cursor column. The current column number is displayed on the STATUS LINE. To set the left margin at the cursor position, press ESC following the **^OL** command.

MARGIN RIGHT— *establishes right margin*

^OR sets the right margin. To answer the request for a right margin, you may enter a column number, or press ESC to specify the current cursor position.

NOTE

You may change margins at any time by reforming the existing text with the new margin settings.

MARGINS FROM FILE LINE—*mimics existing margins*

^OF sets the margins to match those of the existing document. To set the margins with this command, place the cursor anywhere in the existing line and type **^OF**. The margins automatically set to the width of the current file line.

MARGIN RELEASE— *disengages existing margins*

^OX temporarily disengages the current margin settings. Text entered following the **^OX** command can extend beyond the established margins. The margins remain released until the cursor returns within the bounds of the original margin setting. MAR REL appears on the STATUS LINE while the margins are released. You can reset the margins with the same **^OX**.

Place Markers

PLACE MARKERS—*mark a position for later reference*

^K0– ^K9

mark a position within the text where the cursor may be sent. There are ten PLACE MARKERS (0–9). You can set any of these markers within the file and subsequently reference them. The numbered marker will show up at the specified position, but it is not actually part of the document. Each numbered marker may be returned to by using the CURSOR TO MARKER command, **^Q**, followed by the number of the marker (0–9).

You can hide a PLACE MARKER by moving the cursor to the desired location and issuing the same command used to set the marker. In other words, this command acts as a toggle that alternately sets or hides the marker. Though hidden, a set marker is still in effect and will be redisplayed after it is accessed.

Print-Control Characters

You can insert control characters into a text file—by typing **^P**, followed by the print-control character—to control printing.

Displayed print-control characters tend to distort text. However, editing commands ignore print-control characters, and they are not printed.

Print-Control Toggle Commands

Toggle-control characters are those that must be placed on both sides of the affected text. The first toggle character initiates a control effect and the second toggle character terminates the effect. Listed here are the toggle commands:

STRIKEOUT TOGGLE

^PX prints dashes over the specified characters. Use this command to illustrate deleted text in the revised version of a document.

SUBSCRIPT TOGGLE

^PV prints the enclosed characters as subscripts. The subscripted characters will be positioned below the surrounding text. Determine the degree of subscript using the DOT command **.SR**. On printers without fractional lines, the next line must be blank.

SUPERSCRIPT TOGGLE

^PT prints enclosed characters as superscripts so they will appear slightly higher than the surrounding text.

BOLDFACE TOGGLE

^PB offsets slightly and overstrikes on daisywheel printers or any other printer capable of incremental motion. **^PB** multistrikes each character on teletype printers.

DOUBLE-STRIKE TOGGLE

^PD strikes each character twice with no offset. This command produces a lighter version of "boldface." This control character, used with a carbon ribbon, produces an extremely sharp impression of the entire document.

UNDERSCORE TOGGLE

^PS is placed on both sides of the section of text you want to underline. Only nonblank characters are underlined.

Other Print-Control Commands

Following are the control-character commands that control the printer:

LEFT/RIGHT, HEADING/FOOTING CONTROL

^PK is used with heading and footing DOT commands to produce headings, page numbers, etc., that print on the left-hand side of even-numbered pages and on the right-hand side of odd-numbered pages. **^PK** formats headings and page numbers so they will always appear on the side of the page farthest from the binding in loose-leaf binders. Use this print-control character with the DOT commands **.HE** and **.FO**.

ALTERNATE CHARACTER PITCH

^PA is used with daisywheel printers to change the character width from 10 (pica) to 12 characters per inch (elite).

STANDARD CHARACTER PITCH

^PN selects a standard 10-characters-per-inch width (pica) on daisywheel printers.

BACKSPACE

^PH makes the next character overprint the preceding character on the line. Use it to place accent marks over letters or to create special symbols by overprinting multiple characters. This control character is placed where the backspace is desired; it may be affected when text is reformed or justified.

NONBREAK SPACE

^PO prints a space, but the space is not treated as such for line breaks or justification during line formatting.

PHANTOM RUBOUT

^PG prints the character on a daisywheel printer that is associated with code 7F hex. This code prints a "not sign," "double underline," or graphic that is associated with the hex code.

PHANTOM SPACE

^PF prints the character, normally a space code, associated with hex code 20 on daisywheel printers.

STOP PRINT

^PC halts printing. This function gives you a chance to change ribbons or type fonts. You can use this control character within a line as often as needed. When printing stops, the prompt **PRINT PAUSED** appears on the status line. You can restart the printer by typing the PRINT command **^KP** (or **P**).

TAB

^PI displays and prints spaces to advance to the next multiple of eight columns; normally you don't enter it into the text except in VARIABLE TAB mode.

USER PRINT FUNCTION (1–4)

^PQ, ^PW, ^PE, ^PR are USER PRINT FUNCTIONS for special printer operations that WordStar does not otherwise perform. You have to establish each of these functions when you install WordStar. Each function can send a sequence of one to four characters to the printer.

FORM FEED

^PL causes a form feed to be entered into the text.

CARRIAGE RETURN

^PM is the same as a carriage return. Causes the current line to print over the preceding line. The flag character (-) appears in the rightmost column to indicate an overprint.

LINE FEED

^PJ is the same as line feed.

Print DOT Commands

The DOT commands are special characters you embed in text to control the final format of the printed text. DOT commands alter default formats. DOT commands are displayed but they are not printed.

A DOT command consists of a period in the first column of a line, a two-letter code, and optionally a number or some other argument. When you enter a period in the first column of a line, WordStar expects a DOT command and the left margin temporarily disengages; this is indicated by a ? prompt on the current screen line.

Most DOT commands may be placed anywhere in a text file, but the dynamic page break display requires that certain DOT commands appear at the beginning of a file.

The following five sections describe the various DOT commands:

Vertical Page Layout

The following vertical-page-layout DOT commands have to appear at the beginning of the file for page breaks to correctly interpret them.

LINE HEIGHT

.LH *n* sets the line height in 1/48ths of an inch on daisywheel printers and provides an alternative or supplement to the single, double, or triple spacing the PRINT CONTROL command **^OS** gives. Don't use this DOT command on printers that can't print incrementally. The default is 6 lines per inch.

PAPER LENGTH

.PL ***n*** determines the number of lines per page, including the top and bottom margins. The paper length must match the forms specification. The default is 66 lines.

TOP MARGIN

.MT ***n*** specifies the number of lines from the top of the paper to the beginning of the text. The default is 3 lines from the top.

BOTTOM MARGIN

.MB ***n*** specifies the number of lines not to be used for text at the bottom of the page. The page number or footing, if present, is printed within the bottom margin. The default is 8 lines.

HEADING MARGIN

.HM ***n*** determines the number of blank lines that will appear between a page heading and the body of the text.

FOOTING MARGIN

.FM ***n*** sets the number of lines between the last text line of the page and the page number or footing. The default for the HEADING and FOOTING margins is 2 lines.

Horizontal Page Layout

Most horizontal formatting is an integral part of text editing and does not involve DOT commands. However, the following DOT commands cover special print operations.

PAGE NUMBER COLUMN

.PC ***n*** determines the column at which the page number is printed when neither the **.FO** or the **.OP** command is in effect. You can place the page number to the left or right of a page, but you must take the current character pitch into account.

PAGE OFFSET

.PO ***n*** sets the number of columns that the entire document will be indented from the printer's left margin, to offset text from the tractor-feed holes at the left of the paper. This feature allows you to load narrow paper near the center of wide printer carriages. The default is eight columns.

Pagination

PAGE

.PA starts a new page unconditionally.

CONDITIONAL PAGE

.CP*n* starts a new page if there are less than *n* lines left on the current page. This keeps blocks of text together and suppresses pagination after a title, in the middle of a table, etc.

Page Heading, Footing, and Page Number

TEXT HEADING

.HE begins a line that will serve as a heading for each page until another heading is specified. Headings may be changed as often as necessary. To print a heading on the first page, put an **.HE** command in front of all text in the file.

TEXT FOOTING

.FO The rest of a line beginning with this command serves as a page footing for the current and following pages of a document. A document may contain numerous footing commands. Text specified in the most recently encountered footing command is printed.

If no footing is specified, or if an **.FO** command has no text following it, page numbers will be printed in the footing line at the column specified by the PAGE NUMBER COLUMN command, **.PC**. Page numbers are not automatically printed when a TEXT FOOTING DOT command is in effect. You can place a # symbol at the location where you wish a page number to appear.

The following 3 characters have special meaning within TEXT HEADING and FOOTING commands:

> # prints the current page number. Use it to position page numbers wherever you want, at the top or bottom of the page.
>
> \ prints the next character without special interpretation so that control characters may be printed as text.
>
> **^K** is used with the **.HE** or **.FO** DOT commands. This control character directs printing to format a heading or page number depending on whether a page number is odd or even. All spaces following the **^K** character are ignored if the page number is even so that the heading or page number will be printed on the right-hand side of odd-numbered pages and on the left-hand side of even-numbered pages. Use this feature if your document will be printed on both sides.

OMIT PAGE NUMBERS

.OP suppresses printing of page numbers when no footing has been given. This DOT command has no effect if footing has been specified.

NUMBER PAGES

.PN turns page numbering back ON following a previous **.OP** command that turned it OFF. Page numbering will begin with the number 1, unless otherwise specified. (See **.PN** *n* below.)

PAGE NUMBER

.PN ***n*** turns page numbering back ON following a previously issued **.OP** DOT command. Page numbering will begin with the number you specify after **.PN**. The page numbers are printed at the bottom of the page unless a # character specifies otherwise.

Miscellaneous DOT Commands

CHARACTER WIDTH

.CW ***n*** sets the character width in increments of 1/120ths of an inch. The default standard pitch is 10 characters to the inch, and the default alternate pitch is 12 to the inch. This command only works with printers that have programmable character widths. CW 12 is the default.

SUB/SUPERSCRIPT ROLL

.SR ***n*** specifies the amount in 1/48ths of an inch that the carriage rolls before printing a superscripted or subscripted word. The default is 3/48ths of an inch or .SR3.

ON (1) OR OFF (0) MICROJUSTIFICATION

.UJ turns off microjustification which is normally ON. Microjustification spreads words evenly by adding soft spaces. When microjustification is OFF, the text will be printed as it appears on the display with soft spaces and returns. Turning microjustification OFF may be useful to make a columnar table print with the columns aligned as they appear on the screen, even with soft spaces inadvertently supplied by WORD WRAP or REFORM.

ON (1) OR OFF (0) BIDIRECTIONAL PRINT

.BP either enables or prevents the printer from printing back and forth across the page. Use it when you have a problem with the printer.

IGNORE TEXT–

.IG allows display, but not printing, of commentary text in a file line.

After using the appropriate print-control characters and DOT commands to direct the printer, use the print command described below to print your file.

Printing a File

PRINT A FILE—*toggles printing of a file ON and OFF*

^KP (or P from the No-File menu) outputs the contents of a file to the printer. The same **^KP** or **P** commands halt the print operation. The current status of this command is displayed in the **^K** prefix menu.

These PRINT A FILE commands toggle printing so that, when first issued, the name of the file to be printed is requested. When this file name has been supplied, several print option questions are asked. You may simply press **RETURN** in response to each question if you wish to use the default settings, or press **ESC** to prevent the options from being offered. These option questions are:

DISK FILE OUTPUT (Y/N) sends the file to a diskette.

START AT PAGE NUMBER (RETURN for

beginning) lets you specify a page number where printing should begin.

STOP AFTER PAGE NUMBER (RETURN for end) asks for a page number where printing should stop.

USE FORM FEED (Y/N) outputs form-feed controls to the printer.

SUPPRESS PAGE FORMATTING (Y/N) sends an exact replica of the screen display, including DOT commands, to the printer if you answer YES. No formatting is performed on the text before it is printed when page formatting is suppressed.

PAUSE FOR PAPER CHANGE BETWEEN PAGES (Y/N) stops the printer at the end of each page if you answer with YES. This option allows you to change paper between pages.

Ready printer, press RETURN indicates that text is ready to be printed. The specified file will output to the printer when you press **RETURN**.

To halt printing, reissue the **^KP** (or **P**) command, and the following messages appear:

```
TYPE Y TO ABANDON PRINT,
N TO CONTINUE, ^ U TO HALT
```

If you respond with **Y**, the print operation will stop. Pressing **N** causes the print operation to resume immediately. Pressing **^U** temporarily suspends printing; subsequently you can use the **^KP** (or **P**) command to resume printing.

The STATUS LINE will display the prompt message PRINT PAUSED while printing is suspended.

Prompt-Question Control Characters

Special control characters entered in response to prompt questions perform specific functions. These control characters follow:

^X or ^Y erases the entire answer in order to enter another answer (no file menu).

^S, ^H, or ← erase one character to the left (no file menu).

^D moves the cursor to the right and displays any previously erased character. This function will also display the character that was entered the last time that the question was asked (no file menu).

^R restores the erased answer, or the answer you entered the last time you received the current question (no file menu).

^F displays a file directory of the currently logged disk drive for the duration of the command (main menu, with **^K**).

^Z or ^W scrolls the file directory up (**^Z**) or down (**^W**) to bring additional files into view (main menu, with **^K**).

- enters a SOFT HYPHEN if the SOFT-HYPHEN ENTRY has been turned ON. Also permits using SOFT HYPHENS in response to a "FIND?" question prompt. (Use **^P** to search for a hard hyphen.)

^N, ^S, ^A, or ^O have special meaning only with the "FIND" question prompts. (See the FIND FUNCTION.)

^U interrupts and terminates the command in progress. (See the INTERRUPT command.)

Repeat a Command

REPEAT NEXT COMMAND—*repeatedly executes a command*

^QQ

(command) causes the command following it to be executed repeatedly until you press the SPACE bar. You can specify a rate of execution between 0 (slowest) and 9 (fastest). First type the command **^QQ**, then type the command that is to be repeated.

Run a Program

RUN A PROGRAM—*runs a program from the No-File menu*

R

runs a specified program. Enter the name of the program in response to a prompt, and the indicated program executes. You can examine the amount of disk space, move or copy files, etc. Pressing **R** displays the following prompt:

```
COMMAND?
```

Answer this prompt question by entering the name of the program you want to run (ie., XDIR) and then press **RETURN**. Precede the program file name with a drive identifier if it exists on a drive other than is currently logged.

Save Procedures

You can transfer a file to a disk and save it by using one of the following SAVE commands.

SAVE DONE—*saves file and returns to the No-File menu*

^KD saves the file currently being created or edited, then transfers to the No-File menu.

SAVE REEDIT—*saves file without closing it*

^KS saves the current file, then returns to the beginning of the saved file. This command should be used periodically to protect your file from accidental loss by saving it in increments. After the file has been saved, editing of the same file can continue.

The cursor will go to the beginning of the saved file, but you may return it to the position it occupied when you saved the file with the CURSOR PREVIOUS POSITION command, **^QP**. Also, the SAVE REEDITcommand is the fastest method of moving long distances to the beginning of a file.

SAVE EXIT—*saves file and exits to CP/M*

^KX saves the current file and exits from WordStar to CP/M. This command performs the same function as the EXIT TO SYSTEM command, **X**, from the No-File menu, except that **^KX** saves the file before leaving WordStar.

Scroll Commands

The SCROLL commands, which follow, move the screen display down or up a line, or move the entire display down or up one screen's distance.

SCROLL DOWN LINE—*scrolls display down a line*

^W moves the entire screen display down one line.

SCROLL DOWN SCREEN—*scrolls downward a screen's distance*

^R moves displayed text down one screen's distance so the text above is displayed. This command causes the portion of the file that is being displayed to move upward and disappear while text below takes it place.

SCROLL UP LINE—*scrolls up a line*

^Z moves the screen display up one line.

SCROLL UP SCREEN—*scrolls upward a screen's distance*

^C causes text currently occupying the screen to be replaced by an equal amount of text below it.

CONTINUOUS SCROLL—*scrolls upward or downward till stopped*

QW or **QZ** scrolls the display continuously up or down.

NOTE

REPEAT NEXT COMMAND, ^QQ, may be placed before any of the scroll commands to move the display continuously in either direction (see REPEAT NEXT COMMAND).

Status Line

The STATUS LINE at the top of the screen displays various information pertinent to the file being edited. The information displayed from left to right on the STATUS LINE is: the currently active drive, a colon, the name of the file being edited, the number of the page being edited, the line number, and the column number where the cursor is positioned.

When you issue a command, it will be displayed at the upper far left of the STATUS LINE until the command has been executed. If the INSERTION function is ON, it will be indicated at the upper far right of the STATUS LINE. (See INSERTION.)

Certain messages are displayed on the status line when appropriate. These messages are:

> **WAIT** is displayed when a file is being read or written. No data should be entered while this message is displayed.
>
> **MAR REL** indicates that margins have been released.
>
> **LINE SPACING *n*** shows what line spacing is in effect, unless it is set to the default of one line per inch.
>
> **PRINT PAUSED** is displayed when printing is suspended.
>
> **REPLACE** is displayed when the FIND REPLACE function is in effect.

Tab Arrangement

TAB SET—*establishes tab stops*

^O TAB or

^OI

sets a tab stop anywhere on a line to format a document or arrange columnar data. To set the tab, type the SET TAB STOP commands, **^O and TAB** or **^OI**, and a prompt question will ask for the column number where the tab should be set.

Typing a column number will set the tab at the specified column. Pressing **ESC** will set the tab at the cursor column, indicated on the STATUS LINE. All tab stops that are in effect are displayed on the RULER LINE as an exclamation point (!).

NOTE

Decimal tabs will align columns of numbers on the decimal point, or right align text on a tab stop. After you tab to a decimal tab stop, characters entered move to the left, pushing the entire field to the left of the decimal tab setting. The cursor will remain at the tab position. This right alignment function may be terminated at any time by typing a period.

Decimal tabs may be set by using either of the methods for setting tab stops. When using the **^OI** command, you type a # sign before entering the column number or pressing **ESC**. The decimal tab may also be set in the RULER LINE by placing a # sign instead of a ! sign at the column where you want the decimal tab stop.

Decimal tabbing is only active when variable tabbing is ON. You may determine if the VARIABLE TABBING is ON or OFF by pressing **^O** and examining the menu.

TABBING VARIABLE—*toggles from fixed to variable tabs*

^OV turns variable tabbing ON or OFF. When ON, variable tab stops are in effect and spaces are entered into the files for tabs. When OFF, fixed tabs are in effect. Fixed tabs are not usually used during standard word-processing operations; this feature is used when writing computer programs. When variable tabbing is OFF, tab characters **^I** (09 hex) are used in the file and are displayed with fixed stops every 8 columns; multiple spaces are entered into the file when using variable tabbing. Variable tabbing should be turned OFF when programs are being developed under the CP/M text editor or Micropro Wordmaster.

When the variable-tabbing mode is turned OFF, each tab is a single character that edits differently than those used when the variable-tabbing mode is ON. The cursor cannot be placed within the white space on the screen, representing the TAB; the cursor advances over the tab. Text that is inserted before a tab will appear in front of the tab until enough text has been entered to force the text to move to the next tab position.

TAB ADVANCEMENT—*moves to the next tab stop*

^I or TAB advances the cursor to the next tab stop. If no tab stops are encountered on a line, the cursor advances to the first tab on the following line. Only tab stops set within the current margins are used unless WORD WRAP is OFF or the margins have been released.

When INSERTION is ON, **^I** or **TAB** inserts spaces to the tab stop and positions the text to the right of the tab stop.

When INSERTION is OFF, the TAB command will advance the cursor over the existing text. A document line will be extended with spaces or a hard carriage return as an advance to the next tab stop occurs.

TAB PARAGRAPH—*temporarily relocates the margin to the next tab*

^OG temporarily sets the left margin in one tab stop from its present setting. This command indents a paragraph or other section of text. This temporary margin setting will remain in effect until you press **RETURN**, issue another margin command, or move the cursor above (before) the indented text. **^OG** commands issued in succession further indent text.

TAB CLEAR—*disengages specified tab stops*

^ON clears the tab at a specified, prompted location. You can release all tabs by typing **A**, then pressing **RETURN** in response to the prompt.

MailMerge DOT Commands

MERGE-PRINT—*initiates merge-printing of files*

M initiates printing in the same manner as the print command, but additionally interprets MailMerge DOT commands. You are asked for the name of the file to be merge-printed, and are provided with the following options:

```
NAME OF FILE TO MERGE-PRINT?

DISK FILE OUTPUT (Y/N)?

START AT PAGE NUMBER (RETURN for
beginning)?

STOP AFTER PAGE NUMBER (RETURN for
end)?

NUMBER OF COPIES (RETURN for 1)?

USE FORM FEEDS (Y/N)?

SUPPRESS PAGE FORMATTING (Y/N)?

PAUSE FOR PAPER CHANGES BETWEEN
PAGES (Y/N)?

Ready printer, press RETURN:
```

The various DOT commands used to control merge-print operations are described below:

DATA FILE—*specifies the data file*

.DF identifies the data file that contains information to be printed in place of keywords when the calling file is merge-printed.

Format:

.DF d:filename [CHANGE]

The named data file provides the text for keywords in the calling file. The data file will be expected on the currently active drive unless another drive is specified.

Each record in a data file contains fields of information to be supplied for one printed document, such as a form letter. A comma separates each field of information to be printed for a keyword from the next. A data record provides replacement text for all keywords in the calling file. An **.RV** command in the calling file lists keywords in the sequence that text from the data file will be assigned.

The calling file is reprocessed once for every record in the data file. It ends when data has been taken from the last data-file record. You can access only one data file at a time.

To display a prompt message asking for a diskette change, place the word **CHANGE** at the end of the **.DF** command. CHANGE, the name of the expected data file, and the drive where it is expected are all displayed when the CHANGE option appears in a **.DF** command.

Examples:

.DF DATA.FYL

specifies that text variables listed in **.RV** will come from a data file named DATA.FYL.

.DF B:DATA.FYL2 CHANGE

asks that the diskette in drive B be changed.

READ VARIABLES—*reads data for variable keywords (with .DF)*

.RV

lists keywords in the order that text fields from the data file are selected and assigned to them.

Format:

.RV keyword, keyword

The sequence with which keywords are listed in the **.RV** command must correspond to the order that text fields are listed within records in the data file.

The first text field in a record line is assigned to the first keyword listed in **.RV**, and so on. Usually one **.RV** is sufficient to list all keywords in a file, but you can use multiple **.RV**s when necessary.

If a text field is missing from a record in the data file, **.RV** will use the next available text field. Extra text fields are ignored. Processing of a document always starts at the beginning of the next record line.

Example:

.RV NAME, ADDRESS, STREET, COMPANY

ASKS VARIABLES—*requests data to be assigned to keywords*

.AV

asks for data to be entered from the keyboard that will be assigned to a corresponding keyword. Each **.AV** DOT command corresponds to a keyword and an optional message. Data entered at the keyboard is printed in place of the keyword wherever it appears in the file.

Format:

.AV [''message''], variable identifier,
[max-length]

The keyword and a question mark ask for data to be input from the keyboard. Optionally, a prompt message may be displayed: type the desired message within quotation marks in front of the keyword.

You can specify the maximum number of text characters an **.AV** prompt accepts. To do so, place a comma and this maximum number at the end of the **.AV** command. No more than the maximum specified will be displayed or printed.

The following CONTROL characters—**^S** (erase character), **^Y** (erase answer), and **^R** (restore previous answer for this question)—can edit keyboard entries. Press **RETURN** after you supply the appropriate data for the displayed prompt.

To halt printing while **.AV** is asking for data, enter the requested input, then press **RETURN** and **P** in quick succession. This maneuver causes the "STOP" print command, **P**, to be received before the next **.AV** is processed.

Examples:

.AV "Enter First, Last Name", NAME

.AV "City, State", ADDRESS

.AV ZIPCODE, 5

SETS VARIABLE—*establishes text for keywords*

.SV assigns a fixed piece of information to a keyword.

Format:

.SV keyword, data

Data represents text that is permanently assigned to the keyword. Data, up to 200 characters long, may include keywords, providing they have had text previously assigned. You can enter a carriage return into the data with a **^N**.

Examples:

.SV DATE, Dec 20, 1981

.SV PARTY1, John Doe

.SVADDRESS, 22334 55th St. ^NHayward, CA, 94545

.SV PARTIES, &Partyl& and &Party2&

FILE INSERT—*references a file to be inserted*

.FI inserts and processes a named file in its entirety at the point where the **.FI** command is encountered. All commands in the inserted file are processed, including those that reference further insertions.

Format:

.FI d:filename [CHANGE]

Files you insert using **.FI** should end with a carriage return to separate text in the inserted file from text in the calling file. After the inserted file has been processed, processing of the calling files continues where it left off.

Inserted files may include **.FI** commands. This process of referencing files is called nesting. Files can be nested to a maximum of eight levels. An unlimited number of files can be nested when **.FI** is the last command in each file. A keyword in the **.FI** command allows you to enter a file name from the keyboard; you must set up the calling file with the appropriate **.AV** command.

Examples:

.FI DOC.FYL

.FI B:LETTER.FYL

.FI CHAPTER &KEYWORD&

REPEAT PROCESSING—*reprocesses a file*

.RP causes a file to be reprocessed.

Format:

.RP [n]:

A file containing the **.RP** command is processed until all the text in the associated data file has been used; or until it has been processed the specified number of times. If the number specified is larger than the number of records in a data file, processing continues reusing the records of the data file.

Example:

.RP 20 <causes file to be processed 20 times>

DISPLAY MESSAGE—*displays a message on the screen*

.DM displays a message on the screen. If you don't specify a message, a blank line appears. Each message is displayed on the next free line. If the screen is full, existing messages scroll up a line to make room. Keywords can form part of the message, providing you've previously assigned text to the keywords.

Format:

.DM (message)

Examples:

.DM (produces blank line on screen)

.DM This file prints form letters

.DM Printing letter to &NAME&

.DM Load special paper and press P

.DM Insert data diskette in drive B:

CLEAR SCREEN—*clears the screen*

.CS clears all accumulated messages from the screen. An optional message can be displayed following clearance of the screen by placing the desired message after **.CS**.

Format:

.CS [message]

Messages that follow a **.CS** command are displayed on the first blank line at the top of the screen. When the screen fills with messages, the entire display scrolls up one line to make room for the next message. You can place references to keywords in the message, providing you've previously defined data for the keywords.

Examples:

.CS clears message area of screen

.CS Press **RETURN**, P to stop, or enter data for next letter

PRINT FORMATTER—*controls print-time line formatting*

.PF turns the print-time line formatter ON, OFF, or to default.

Formats:

.PF OFF:

This form suppresses print-time line formatting. Inserted text is printed without being formatted to accommodate the length of keywords.

.PF ON:

You must turn print-time line formatting ON before you can use other DOT commands to format printed text. Print-time line formatting remains ON until you issue a **.PF OFF:** or a **.PF DIS:** command.

Print-time line formatting with other DOT commands reformats text before it is printed.

.PF DIS: (default)

This form leaves print-time line formatting to the discretion of merge-print. Print-time line formatting is automatically turned ON when a variable identifier is detected, and is turned OFF when the next carriage return, line feed, form feed, or end of file is encountered.

RIGHT MARGIN—*sets right margin*

.RM affects the right margin setting during merge-printing.

Formats:

.RM ***n:***

A number between 1 and 240 specifies the column at which the right margin is set for the printed text.

NOTE

Print-time line formatting must be ON before ***.RM*** *will have any effect. (See the* ***.PF*** *command.)*

.RM DIS: (default)

This form uses the right margin you specified when you created the file.

LEFT MARGIN—*sets left margin*

.LM specifies the left margin setting.

Formats:

.LM ***n:***

When **.LM** and a number between 1 and 240 occur during merge-printing, the left margin is affected as specified. If the document contains hanging indentations or text that extends to the left of a desired margin setting, don't use the **.LM** command.

NOTE

Print-time line formatting must be turned ON before ***.LM*** *commands will work. Print-time line formatting is turned ON when a variable identifier is detected in the current paragraph, or when the DOT command* ***.PF ON:*** *is encountered.*

.LM DIS: (default)

.LM DIS: uses the left margin setting you specified when you created the file.

LINE SPACING—*sets line spacing*

.LS establishes line spacing during merge-printing.

Formats:

.LS ***n***

You can specify line spacing between 1 and 9. **.LS** has no effect unless print-time line formatting is ON, as described for **.LM**.

.LS DIS: (default)

.LS DIS: causes document to be printed with the line spacing used when you created the document file.

INPUT JUSTIFICATION—*determines input justification*

.IJ determines whether the input scanner interprets input as justified.

Formats:

.IJ ON:

.IJ ON: interprets input text margins as justified.

.IJ OFF:

.IJ OFF: assumes that input text will not be justified.

.IJ DIS: (default)

.IJ DIS: leaves input text right margins as you specified them when you created the file.

Small variations in the right margin indicate ragged right. A constant right margin and soft spaces between words indicate justification.

NOTE

When you intend to justify output from ragged-right input, or vice versa, use the .IJ DOT command.

OUTPUT JUSTIFICATION—*determines output justification*

.OJ

right-justifies printed text.

Formats:

.OJ ON:

.OJ ON: prints text with the right margin aligned.

.OJ OFF:

.OJ OFF: prints text with a ragged right margin.

.OJ DIS: (default)

.OJ DIS: prints text using the right margin you specified when you created the file.

NOTE

.OJ has no effect unless print-time line formatting is ON, as described for ***.LM****.*

WordStar Default Values

	D Option	N Option
Left margin	column 1	column 2
Right margin	column 50	column 50
Variable tab stops	6, 11, 16, etc.	9, 17, 25, etc.
Word wrap	ON	OFF
Justification	ON	OFF
Ruler display	ON	OFF
Page break display	ON	OFF
Print-control display	ON	OFF
Soft-hyphen entry	OFF	ON
Hyphen-Help	ON	OFF
Insert mode	ON	OFF

SuperCalc

Cursor Movement

There are several ways to move the worksheet cursor to a new active cell. You can use the four arrow keys. Similarly, you can use the alternate diamond keys. Hold down the **CTRL** key while you press the **S**, **D**, **E**, or **X** key to move left, right, up or down in that order.

Alternately, you can use the = (address) command—also called the **GOTO** command—to move directly to the designated cell. The SuperCalc program will ask for the cell coordinates. When you type them, the display on your screen will change. If the designated cell is already on the display, it will show as the active cell. If not, the window will move to show the new active cell at the upper-left corner. There is a special case: if you type only = and press **RETURN**, the window will adjust to show the current active cell at the upper left.

Special Function Keys

? for help

When you use the SuperCalc program and need information about your current entry options, press **?**. The display screen will change to show you a list of entries that you can make relative to your present position within SuperCalc. This help function is available at any time and in any mode. Press any key to return to the previous display.

The ! key

The ! command forces recalculation. In the manual-calculation mode, this command is the only way to have the program recalculate values. In the automatic mode, it provides an additional recalculation.

The ; key

; moves the worksheet cursor from one portion of a split window display to the other portion. See the **WINDOW** command.

The ESC key

The current-cell key is the **ESC** (escape) key. When you press it, the SuperCalc program puts the location of the active cell onto the entry line for you to use in a command or expression. After you press **ESC**, the arrow and alternate diamond keys control the worksheet cursor. If you move the worksheet cursor, the active cell address on the entry line changes dynamically to reflect the new location. When you press **ESC** again, the address stops changing, and the arrow and diamond keys are again available for editing.

Pressing **,** after the active-cell address is a special case. SuperCalc places another active-cell address after the colon. The address before the **,** is fixed; the address after the **,** is still changeable.

The new active-cell location is temporary. When you press **RETURN** to enter the command or expression, the worksheet cursor returns to the prior active-cell location. If you are entering data into a cell, it will go into that prior location.

Data Entry

SuperCalc accepts numbers, formulas, and text. Ordinary numbers can have 16 significant digits plus a decimal point. Scientific, or exponential, numbers can have 16 significant digits and a decimal point, all raised to a power of ten. The limit is the 63rd power of 10. Text can have up to 110 characters. Formulas can have up to 110 characters and can include arithmetic expressions, relational expressions, functions, and references to cells.

Once you begin to type a command or data on the entry line, the four arrows—and the alternate diamond keys—no longer move the worksheet cursor around the worksheet. Instead, you can use them to edit information on the entry line. You can always correct your commands or data while they are on the entry line. The **EDIT** command allows you to use the edit process, and enter the changed contents into the active cell after you have committed an entry to the worksheet.

Left and right arrows (**^S** or **^D**) move the data-entry cursor without erasing an entry so you can position the cursor where you want to make the change.

Because a cell can contain 110 characters—longer than entry line can show—SuperCalc will scroll your entry during the edit process, allowing you to examine any portion of it. Wherever the cursor is, you can enter a new character to replace the old one. The cursor then moves right one location.

Each time you press the down arrow (or **^X**) it deletes a character. The cursor stays in position.

The up arrow (or **^E**) inserts a new space at the cursor location each time you press it. The cursor stays in place, and spaces fill out to the right of it. The space(s) can then be filled with additional characters.

Remember, what you see on the entry line is what is entered into the active cell. When you finish making your changes and enter the data or execute the command, SuperCalc takes everything on the entry line, not just the material to the left of the cursor.

Data-Entry Limit

Numbers: 16 significant digits, plus optional decimal point and optional sign for ordinary numbers. 16 significant digits, plus decimal point and optional sign for exponential numbers (scientific notation). These 16 digits can be raised to the 63rd power of 10.

Largest ordinary number.	9999999999999999
Smallest ordinary number.	−9999999999999999
Largest exponential number.	9.99999999999999e62
Smallest exponential number.	−9.99999999999999e62

Text: 115 characters

Example: **"Expenses, January**

Formulas: 116 characters

Example: **7+A5,9+5*E7,SUM(B1:B9), MIN(A4,D4,G4)**

Distinguishing Numbers, Text, and Formulas When Entering Data

Numbers start with digits (0–9), +, −, or a period. An entry beginning with a period is assumed to be a decimal entry beginning with zero and a decimal point.

Text starts with a quotation mark (").

Formulas can start with the same characters as numbers—0–9, +, −, or period. They can also start with an open parenthesis —(. You can put arithmetic expressions, relational expressions, functions, and references to cells within formulas.

About Numbers

Numbers are ordinarily right-justified; optionally, they're left-justified. The way numbers appear when they're displayed depends on the display format you selected, not on the way they looked when you entered them into the cell. The display format doesn't affect the content of the cell.

Display options allow you to display numbers in the following ways:

- general (ordinary numbers if they fit column display width; otherwise, exponential);
- exponential (scientific notation), rounded if necessary; integer (integers only; if there is a decimal number, round up or down to make it integer);
- dollar amounts, rounded to the nearest cent; .00 is appended to whole numbers;
- graphic display, using asterisks to show relative values in bar-graph form.

For any display format, if the numeric display cannot fit into the column, then >>>> fills the column.

You can widen a column to display a number or text in full by setting column width from 1 to 126.

NOTE

Text is ordinarily left-justified; optionally it is right-justified. If text is too large for the column, the text display continues into the adjoining blank cell(s) to the right. If it cannot continue into adjoining columns, it is cut off at the right.

NOTE

Ordinarily the resulting values rather than formulas are displayed; optionally, you can see the formula with the ***F*** *option in the* ***GLOBAL*** *command. The formula however, is shown on the status line. When the formula is displayed in a cell, it can continue into adjoining blank cells as text does.*

Status, Prompt, and Entry Information

SuperCalc uses the prompt and status lines near the bottom of your screen to send you messages. You use the entry line to respond. Here is a more detailed look at some of the things you may find on those lines:

Status Line

The status line is the first of the three lines. This line displays information about the active cell. The information displayed includes: the "Current Direction" that you have been moving, the active-cell location, the active cell's specific format and protection, and the textual contents of the active cell.

Here is an example of a status line:

> **>A5 L$TR P Text="February**

Here is what it means:

> \> is the current direction of the worksheet cursor, set by the last arrow key pressed. It may be >, <, v, or ^.

A5 is the active-cell location. Data entered will go into that cell. Commands that use the current column or row will use the column or row containing that cell; in the example, column A and row 5.

L$TR shows the active-cell format settings; numbers are left-justified, $ is the format, and text is right-justified. (The detailed reference for /Format gives full information on these settings).

P shows data protection of the active cell. This area is blank if the cell is unprotected.

Text="February indicates the contents of the Active Cell —in this case, text. **"Rtxt="** indicates repeating text. Numbers or formulas are shown as **"Form"**; for example, "FORM =12*B9."

SuperCalc also uses the status line to display error messages and certain informational messages. These special messages disappear and the status information reappears when you press any key.

Prompt Line

The middle of the three information lines serves a dual purpose: while you are entering a command, this line "prompts" you by outlining the choice of possible entries you may make. For example, after you have issued the **DELETE** command, the prompt line reads:

```
R(ow) or C(olumn)
```

This tells you that you must next tell the SuperCalc program whether you wish to delete a Row or a Column. If you then ask for Row, the prompt line changes to:

```
Enter Row Number
```

to ask you which row number to delete.

Whatever the prompt is, if you press **?** a short but detailed explanation of your options will be displayed on the screen.

When you finish typing your command, the middle line reverts to its other function: global status. It tells you about your worksheet's current status. An example is:

```
Width: 9 Memory:29 Last Cell:J10 ? for HELP
```

This information is:

> **Width: 9**—column width. This is the display width of the column that contains the active cell. The standard, or default, setting is 9, but you can specify a different width. You can set all columns to the same width or set different widths for different columns. If you change the default setting, the status line lists the display width that you select.
>
> **Memory: 29**—available memory in kilobytes (a kilobyte is the memory sufficient to hold 1024 characters or digits). This number changes as you add data to the worksheet.
>
> **Cell: J10**—this tells you the lower right-hand corner of an imaginary block that just contains all your worksheet. In other words, J is the right-most column that you have used, and 10 is the lowest row (biggest row number).
>
> **? for HELP**—this reminds you that pressing **?** will always give you an explanation of the options you have at that moment. If you press **?**, you will receive an explanation of your choices.

/ precedes most commands. If you press it, the prompt line will change to list possible entries and the **?** symbol.

As you proceed within commands, or make other possible entries, the prompt line will change to show you your current choices.

Entry Line:

This is the line where you tell SuperCalc what to do by typing your commands or data. Your experience with the tutorial chapter and the information in this Reference Guide will give you all the information you need to type in the desired data.

SuperCalc Command Entry

Type all SuperCalc commands with **/** and the first letter of the command. The remaining letters in the command are automatically supplied on the entry line. For example, **/ B** causes the command word BLANK to be displayed on the entry line. The prompt line lists the choices available to you for that command. When you enter **/**, the prompt line shows the possible one-letter entries. After you choose a command, the prompt line changes to show the choices available for that particular command. Whenever you wish further information about your options, you can press **?**.

Some commands exhibit a sequence of prompts and entries before the command is executed. An example is the command to copy from one location to another. If you enter one of these multilevel commands, you can back out of your current entry by using the back (left) arrow. In fact, you can back entirely out of the command, level by level, till you return to the desired level.

You can edit commands just as you do data by using the in-line editor. Remember that when you press **RETURN**, everything visible on the entry line will be executed—not just the part of the command to the left of the cursor.

A few commands effect only the current cell, column, or row. Most allow you to specify which cell, column, or row to be affected in the command line. You can type column addresses as either capital or lowercase letters; SuperCalc converts lowercase column entries to capitals. If you want to specify the current cell, column, or row (as appropriate) in such commands, simply press the comma to enter the current location into the command line.

The current-cell key (**ESC**) can also enter the current cell, column, or row into the command line (if only the column or row is needed, the other part of the current cell location is ignored). Once you press **ESC**, you can move the active cell temporarily to a new location. Its address changes on your entry line, and you can use it in your command. By pressing **:** you can develop two cell addresses, such as B5:E5.

RETURN follows up a command, causing it to be executed. In some cases, a comma can also end a command because pressing the comma enters the last item of information needed in the command line. The command is complete, so SuperCalc executes it.

All commands consist of / and a single letter. SuperCalc's interpretive prompting fills out the rest of the word, and the prompt line lists the options available. These commands are summarized here and described in detail in the tutorial chapter.

Commands Involving Formulas

Some commands move formulas to new locations. It is usually desirable to adjust formulas for their new locations. For example, suppose cell D4 has the formula +B4*C4. If the contents of cells B4, C4, and D4 move to T7, T8, and T9, the formula in T9 should read +T7*T8. SuperCalc ordinarily makes such adjustments automatically.

Some commands optionally allow you to move formulas without adjustment, or query whether each cell reference for each formula should be adjusted. Some commands also have an option to move values only; formulas do not transfer, only their values move.

Commands Involving Formula Adjustments

DELETE, **INSERT**, and **MOVE** all cause automatic formula adjustment. They have no options. Deleting a column or row that contains a cell on which a formula, outside the range of the deletion depends, will cause an error.

COPY and **REPLICATE** allow formula adjustment. Adjustment is automatic, unless you specify otherwise by selecting one of the options. The options allow you to disable formula adjustments, or to choose whether SuperCalc should adjust individually for each outside reference.

LOAD adjusts formulas if you're loading the material into a worksheet location different from the one where it originated. In this case, you have the same options as in **COPY** and **REPLICATE**.

SuperCalc Commands

Data Commands

/BLANK—*blanks contents of a cell or range of cells*

Blanks the contents and clears the cell format of a cell, partial column, partial row, or block. Also clears the formatting of the cell if it has been formatted individually (that is, at the E(ntry) level; see **FORMAT**).

Prompt:

```
Enter Range
```

Formatting for a column or row is not affected, even if every cell in it is blanked. Only the

FORMAT command can change the format for a column or row. Protected cells will be bypassed.

Examples:

/**Blank, c7** <**cr**>

/**Blank, c7:c12** <**cr**>

/**Blank, c7:h7** <**cr**>

/**Blank, c7:h7** <**cr**>

/EDIT—*transfers cell contents to entry line for editing*

The edit command lets you edit the contents of a specified cell, then place them back in the active cell. If the active cell is protected, you cannot edit.

Prompt:

From? Enter cell

Specify a cell in response to the prompt; **,** indicates the current or active cell. The cell contents come to the entry line, replacing the command on the line.

Edit with the in-line edit function. Use the arrow or diamond keys to move the cursor nondestructively left and right to characters you want to change. The character that will be altered is the one above the cursor. You can replace characters one-for-one by simply typing new characters over them. You can delete characters, including blanks, by pressing the down arrow (or **^X**). You can insert blanks by pressing the up arrow (or **^E**). Then if you wish, you can replace the blanks by typing other characters over them.

Example:

The active cell contains Janaurry.

/**E** and **,** bring this example to the entry line. Use the left arrow to move the cursor to the second "a" in Janaurry and type **ua**. Move the cursor right to one of the "r"s in Januarry, then press the down arrow to delete it, and press **RETURN**. (Remember, pressing **RETURN** puts the entire entry into the cell no matter where the cursor is positioned.)

The active cell now contains January.

/FORMAT—*specifies format for a given portion of the worksheet*

The Format command affects the worksheet **G**(lobally); by specific **C**(olumn), **R**(ow), **E**(ntry) cell; or range (col/row:col/row), in one or more of the following ways:

I(nteger) notation, **G**(eneral format), **E**(xponential) notation to the tenth power, (**$**) dollar format, **R**(ight) or **L**(eft) justification, (*) asterisk fill relative to value. **D**(efault) is: the general display, numeric right-justified, text left-justified, and column width 9.

Prompts:

Enter Level: G(lobal), C(olumn), R(ow), or E(ntry)

Specify the portion of the worksheet to be affected; **,** will specify the current column or row. If you press **E**, you can specify a single cell or a range of cells; that is, a partial column or partial row. Using **E** to specify formatting at the cell level provides the highest priority of formatting.

The next prompt message you receive depends on the level of formatting you specified.

A level of **G** or **C** has this prompt:

```
Define Formats: (I,G,E,$,R,L,TR,TL, * ,D, column width)
```

A level of **R** or **E** has the same prompt, except that "column width" is not included because it is not a valid choice.

You may select as many of the formats as you wish. Here is a list of the possible format choices:

I displays numbers as integers. This rounds decimal fractions up or down to convert them to whole numbers.

E (exponential) displays the number in scientific notation, as a power of 10. For example: 1776 is 1.77e3, 1,000,000 is 1.0e6; round if necessary.

G (general) displays the number as an ordinary number if it fits in the column width; otherwise, it displays the number as an exponential number.

$ (dollar amount) rounds to the nearest cent and appends .00 to whole numbers. No dollar sign is displayed.

* (graphic display for numbers) uses asterisks to show the relative sizes of numbers. Allows bar graph display.

R,L (right-justify, left-justify) is for numbers.

TR,TL (text right, text left-justify) is for text.

0–126 is the column width for the specified column or for the worksheet.

D (default) resets to the next level of formatting. See note 2 below.

When your entries are contradictory, the SuperCalc program will act on the one entered last. For example, if you enter R,L,I,G, then L and G will take effect, and SuperCalc will ignore R and I.

NOTE

1. *Format does not apply to data entry. The contents of a cell remain as entered; format specifies how the contents are displayed.*

2. *Where formats differ, the order of precedence is first the cell (**E**), then row (**R**), column (**C**), and finally worksheet or global (**G**). Cell formatting overrides any format for the column or row where the cell is. Where row and column intersect, row formatting overrides. Any of these override the global settings.*

   *When the program starts up, these global format settings are in effect: general numeric display (**G**), numeric right justify (**R**), text left justify (**TL**), and a column width of 9.*

Examples:

/Format, **C, E, 12** **<cr>**

/Format, **R, TR,** **<cr>**

/Format, **G, $, 11,** **<cr>**

/Format, **E, E,** **<cr>**

Worksheet Adjustment Commands

/DELETE—*erases data from a specified column or row*

When you type the command **/D** you are asked whether to delete a column or row.

Prompts:

```
R(ow), C(olumn) or F(ile)?
```

If you reply **R** , the prompt becomes:

```
Enter Row Number
```

You may then type a number from 1 to 254, or type **,** for the current row.

If you reply **C** , the prompt will be:

```
Enter Column Letter
```

You may enter a letter designation from A to BK, or type (**,**) for the current column.

This command deletes the contents and formatting of the specified row or column. The command will not execute if a protected cell is in that row or column.

The rest of the worksheet makes the following adjustments:

- Rows below the deleted row move up, and all row numbering adjusts. If row 4 is deleted, row 5 moves up and becomes the new row 4, and so on.
- Columns to the right of the deleted column move left. If column D is deleted, column E moves and becomes column D, and so on.

Examples:

/Delete, **R,5,** <**cr**>

/Delete, **C,E,** <**cr**>

All formulas on the worksheet are automatically adjusted as necessary. The adjustments preserve references to cell contents by giving their new location. For example:

Row 3 is deleted.
A prior reference was SUM(B2:B5)
That reference becomes SUM(B2:B4)
The contents that were at B5 are now at B4.

A reference to B3 would cause an error if column B or row 3 were deleted, because the contents vanish, and there can be no new reference to them. SuperCalc cannot assume that this is a special case, one where you want the old formula to refer to the new contents of cell B3. For example:

Cell A6 has the formula SUM(B3,F3,G3).
Column B is deleted.

Cell A6 will now display ERROR because the contents of B3 have vanished. To correct the error, you must correct the reference to B3 in cell A6.

/INSERT—*inserts an empty column or row where indicated*

The **INSERT** command inserts a column or row where needed. The inserted column or row replaces the specified column or row while the rest of the worksheet adjusts by reassignment of parameters.

Prompts:

R(ow) or C(olumn)?

If you reply **R** , the new prompt is:

Enter Row Number

You may select a number from 1 to 254, or type , for the current row.

If you reply **C** , the prompt is:

Enter Column Letter

You may enter a letter or letters from A to BK, or type a comma (,) for the current column.

This command inserts a new row or column of empty cells between existing rows or columns. A new row appears above the specified row; a new column appears to the left of the specified column.

The rest of the worksheet adjusts. Columns move right, rows move down. The contents of each column or row are preserved but have a new designation. The contents, if any, of the last row (254) or column (BK) are discarded. The command will not execute if that last row or column contains a protected cell.

Examples:

/Insert **R,5** <**cr**>

/Insert **C,D** <**cr**>

All formulas on the worksheet are automatically adjusted as necessary. The adjustments preserve references to cell contents by giving their new location. For example:

Row 3 is inserted.
A prior reference was SUM(B2:B5).
That reference becomes SUM(B2:B6).
The contents that were at B5 are now at B6.

A prior reference to B3 itself will become a reference to B4 when a new 3 is inserted.

/MOVE—*relocates a column or row of data*

The **MOVE** command transfers the contents from one column or row to another.

Prompts:

```
R(ow or C(olumn)?
```

If your reply is **R** , the prompt is:

```
From? Enter row number
```

You may enter a number from 1 to 254, or type a comma (,) for the current row.

If you reply **C** , the prompt is:

```
From? Enter column letter
```

You may enter a column designation from A to BK.

After you have specified a row or column, SuperCalc will ask the destination of the move. The prompt is:

```
To? Enter column letter
```

Reply with a row or column designation, whichever is appropriate. Pressing , or the current-cell key (**ESC**) will designate the current row or column.

The **MOVE** command adjusts the worksheet without destroying any data or performing any formatting. It moves a specified column left or right and inserts it in a new location, or moves a specified row up or down and inserts it in a new location. The columns, or rows between, move to fill the old location. They move in the opposite direction of the basic move.

Examples:

/**M**ove R,5,12 <cr>

/**M**ove C,E,A <cr>

All formulas on the worksheet adjust automatically as necessary. The adjustments preserve references to cell contents by giving their new location. For example:

Row 3 is moved to row 5.
The former rows 4 and 5 move up to become new rows 3 and 4.
The former row 3 becomes row 5.

A prior reference was SUM(B2:B5) —That reference becomes SUM(B2:B4). The contents of B5 are now at B4.

Copying and Replicating Rows or Columns

/COPY—*duplicates data from source to destination*

The **COPY** command allows a one-to-one copy of a cell, partial column, partial row, or block to a new location. Options give a choice of formula adjustment or copying values only.

Prompts:

```
From? (Enter Range)
```

Specify a cell, partial column, partial row, or block.

The next prompt is:

```
TO? (Enter Cell), then Return; or "," for Options
```

Copy makes a one-to-one copy of the source into a destination of the same shape and size. Enter a single cell address to give the new location:

For a partial column, give the upper cell.
For a partial row, give the left cell.
For a block, give the upper left cell.

Press **RETURN**, or if you wish a choice of options for copying formulas, press the comma key.

If you press **RETURN**, then all the formulas are copied and automatically adjusted; that is, all references to other cells are adjusted for their new location, if possible.

If you press **,** to select options, SuperCalc will enter the cursor's location as a destination. Delete if not wanted. It will prompt you with:

```
N(o) Adjust, A(sk for Adjust). V(alues)
```

N—Copies formulas exactly as they are.

A—Allows you to choose for each reference to another cell address within a formula whether to copy it as is, or to have the SuperCalc program adjust it.

V—Copies the values only, without formulas.

When you choose the (A)sk option, each formula that qualifies for possible adjustment is displayed on the entry line. Its source and destination address are shown on the prompt line. SuperCalc positions the cursor at each cell reference on the entry line, and asks you to reply **Y** or **N**. Y means yes, automatically adjust. N means no adjustment, transfer as is.

Examples:

/Copy, b9, c12 <cr>

copy cell to cell.

/Copy, b9:b15, e9 <cr>

copy partial column to partial column.

/Copy, b9:g9, h12 <cr>

copy partial row to partial row.

/Copy, **b9:g15, k20 <cr>**

copy block to block.

/Copy, **b9, c12,N <cr>**

copy without adjustment.

/Copy, **b9, b15, e9, A <cr>**

copy, ask for individual choice of adjustment.

/REPLICATE—*transfers source until specified range is filled*

The **REPLICATE** command makes a one-to-many copy of a cell to a group of cells, a partial column to a group of partial columns, or a partial row to a group of partial rows. Options give a choice of formula adjustment or replicating values only.

Prompts:

```
From (Enter Range)
```

Specify a cell, partial column, or partial row, followed by a comma.

The next prompt is:

```
To? (Enter Range), then Return;
or "," for Options.
```

Replicate makes a one-to-many copy of its source into a new destination that is larger than the source:

A cell into a partial column or partial row.

A partial column into a group of partial columns. The destination address is given as the left and right cell addresses on the top row of the destination group. The partial column will be copied once for each cell in that portion of the row.

A partial row into a group of partial rows. The destination address is given as the upper and lower cell addresses for the left column of the destination group. The partial row will be copied once for each cell in that portion of the row.

Specify the destination and press **RETURN**; then if you wish a choice of options for copying formulas, press ,.

The options are the same as those for **COPY**. If you press **RETURN**, formulas are adjusted automatically. The options are: no adjustment (**N**), whether to adjust for values only (**A**), or leave formulas behind (**V**). (See /**COPY** above for details.).

Examples:

/**Replicate, b12,e3:e8** <**cr**>

replicates a cell into a partial column.

/**Replicate, b12,e3:j3** <**cr**>

replicates a cell into a partial row.

/**R**eplicate, b3:b7,d3:j3 <cr>

replicates a partial column into a group of partial columns. In this example, the partial column is five cells deep. The result will be a block of cells repeating that partial column seven times. The top of that block is on row 3.

/Replicate, b3:e3, g5:g7 <cr>

replicates a partial row into a group of partial rows. The partial row here is four cells across. The result will be a block of cells repeating the partial row three times. The left side of that block is column G.

/Replicate, b12, e3:e8,N <cr>

replicates without adjustment.

/Replicate, b12, e3:j3, A <cr>

replicates and asks for individual choice of adjustment.

NOTE

*As a special case, /**REPLICATE** can make a one-for-one copy just as /**COPY** does. /**COPY** cannot make multiple copies. /**COPY** can, however, do something that /**REPLICATE** cannot do; it can copy a block.*

Data Protection Commands

/PROTECT—*provides protection against alteration of data*

The /**PROTECT** command shields the contents and formatting of specified cells from alteration. You can't enter or edit data in protected cells.

Prompt:

Enter Range

/BLANK, **/FORMAT**, **/COPY**, **/REPLICATE**, and **/LOAD** all bypass protected cells—that is, the commands operate on surrounding cells but leave the protected cells unchanged. **/DELETE** will not work if a protected cell is in the specified row or column.

There is one exception: the **/ZAP** command overrides protection.

Examples:

/Protect,c3 <**cr**>

/Protect,c3:c9 <**cr**>

/Protect,c3:g3 <**cr**>

/Protect,c3:g9 <**cr**>

/UNPROTECT—*allows exposure of previously protected cells*

This command removes protection from a cell, partial row or block.

Prompt:

Enter Range

Allows you to change cell contents or format. There is no error if you try to remove protection from something that is not protected.

Examples:

/Unprotect,c3 <**cr**>
/Unprotect,c3: <**cr**>
/Unprotect,c3:g3 <**cr**>
/Unprotect,c3:g9 <**cr**>

LOAD, SAVE and EXECUTE Commands

/LOAD—*loads and displays part or all of a disk file*

The /**LOAD** command reads a SuperCalc data file from a diskette and loads it into memory; the worksheet displays the contents of the file. You may load all or part of a worksheet at a location you specify. Options give a choice of formula adjustment or loading values only.

Prompts:

```
Enter File Name (or RETURN for directory)
```

Enter the name of the desired file with the drive designation, unless you want the file loaded from the SuperCalc diskette. The file name must have the .CAL-type "CAL". This extension is assumed, and you do not have to enter it. Do not leave blank spaces in the file name. For example:

SALESFEB <**cr**>

would load the file from the SuperCalc program diskette into drive A.

B:SALESFEB <**cr**>

would load the file from the B disk drive.

You receive a choice of loading the entire file or a specific portion of the file. The following prompt displays:

```
A(ll) or P(art)?
```

If you reply **A**, the entire worksheet is loaded into the original location.

If you reply **P**, then further questions appear on the prompt line:

From? (Enter Range)

Specify the position of the saved worksheet that you wish to load.

To? (Enter Range) then RETURN or "," for options.

Enter the cell address at the upper left of your destination, which may be a new location for that portion of your worksheet. Press **RETURN** if you wish automatic adjustment of formulas for the new location; otherwise, press **,** for options. The options are: **N**(o Adjustment), **A**(sk for Adjust), or **V**(alues) only. (See /**COPY** for an explanation of these options.)

NOTE

If there are protected cells in the destination area, they will remain unchanged.

Examples:

/Load, **QUARTER3 <cr>**

/Load, **B;QUARTER3 <cr>**

/SAVE—*stores data from the current worksheet to disk*

The /**SAVE** command stores the worksheet contents and all settings on a disk file. Options give a choice of saving all contents or values only.

Prompts:

```
Enter File Name (or < RETURN > for directory)
```

Enter the name you have chosen for saving your worksheet. Also enter the drive designation if you do not want to write it to the disk in the default drive (A). The SuperCalc program will automatically give the file the .CAL file-type extension. You do not need to enter it as part of the file name. The next prompt is:

```
A(ll), V(alues) or P(art)
```

A specifies that all cell contents will be saved; **V** specifies that values will be saved without formulas. For either case, all of these are saved: format settings, global options, title locking, window splitting, and active-cell location. **P** allows you to save only part of of your current worksheet. When saving part of the worksheet, you can decide whether to save the entire portion **A**; or just the values **V**.

NOTE

If you specify the name of an existing file, the program will display the following prompt:

```
File already exists:
C(hange name),B(ackup) or O(verwrite)?
```

Examples:

/Save, **WORK5 <cr>**

/Save, **B:WORK5 <cr>**

/XECUTE—*executes a group of commands from a disk file*

The /**Xecute** command causes the commands in a named file to be executed one after another. The /Xecute command allows you to execute a WordStar text file or SuperCalc file of command strings. When you enter /X the prompt line changes to:

Prompt:

```
Enter File Name (or < RETURN > for directory)
```

If you press **RETURN**, you will be given the option to display the directory (explained in the /Delete command). If you enter a file name, the SuperCalc program reads each of the commands in the specified file a character at a time. If the file is not in the proper format or a command is in error, an error message is displayed on the status line and the Xecute command is abandoned. You can terminate the command at any time with **^Z.**

NOTE

The default extension for command files is .XQT. If your file has no extension, you must still place a period after the file name.

Example:

```
/X TEST1 <cr>
```

The WordStar or SuperCalc file named TEST1 would look like this:

```
/ZY
/FCA,20
/LB:BALANCE,A
/GF/GM/FGD,$
```

Worksheet Display Commands

/TITLE—*provides method for fixing titles*

Title allows you to lock columns, rows, or both into their place on the display window. Locked information will not scroll; however, other information on the screen can scroll. Title lock uses the current row and column as the coordinates to be affected.

Prompt:

```
H(oriz), V(ert), B(oth), or C(lear)?
```

H locks the current row and all rows above it.

V locks the current column and all columns to the left of it.

B locks both the current row and column and all rows above and columns to the left.

C removes the title lock.

A replaces a prior title lock with a new one.

/WINDOW *splits the screen into two worksheets*

The **/WINDOW** command splits the display window into two parts. Each portion can have separate format settings and options. The screen is split at the current row or column.

Prompt:

```
H(oriz), V(ert), C(lear Split), S(ynch), or U(nsynch)
```

H The screen splits horizontally; the current row moves down and a second border replaces it. The active cell moves up one cell in its column.

V The screen splits vertically; the current column moves right and a second border replaces it. The active cell moves left one cell in its row.

NOTE

In both these cases, there is an alternate active cell in the original location. You can switch between the two active cells by pressing ; as they move independently.

C Clears the split screen. The portion that was above or to the left is the primary screen; it is now displayed in full.

S Synchronizes scrolling in the two portions.

U Unsynchronizes scrolling; the two portions will scroll independently.

Within the two portions of the screen, you can set formatting and global options independently. It is possible to show the same data with different formatting and options—for example, to show the same column as values and as formulas.

When the split is cleared, the options and formats for the primary screen remain. The primary screen is the portion above or to the left.

Data Display and Printing Commands

/OUTPUT *sends worksheet contents to the printer or disk file*

This command writes part or all of the worksheet to the printer, the terminal, or a disk text file. You can write out a partial column, partial row, or block. If you write the report to a disk file, you can use WordStar to add further information or modify formats before printing, or to include the SuperCalc report within other text.

Prompts:

```
D(isplay) or C(ontents) report?
```

The worksheet information can be written out in the way it is displayed, or as the actual cell-by-cell contents. If you choose **D**, for display, the entire worksheet is output. If you choose **C**, for contents, the following prompt appears:

```
Enter range
```

After you specify the portion of the worksheet to output, a prompt asks whether you want the data output to the printer, console, or disk. You may also change the default printer settings:

```
P(rinter), S(etup), C(onsole), D(isk),
```

Type **P** to send the data to the printer, **C** to display it on the screen, or **D** to send it to a disk file. Type **^Z** to stop the output. If you type **S**, for setup, the following options are provided:

```
          Select printer control:

L  = Change page length        (now # lines)
     (Length = 0 for continuous form)
W  = Change page width         (now # chars)
S  = Manual setup codes
P  = Print report

       CNTRL-Z to cancel /O command
```

You may change one or more of the above parameters, then type **P** to output the report, or **^Z** to cancel the entire process.

GLOBAL Options, QUIT, ZAP

/GLOBAL *manipulates screen formatting and calculations*

The **/GLOBAL** command lets you view formulas on which values are based, change the appearance of the screen display, and specify the order and sequence of calculations.

Prompt:

```
F(orm),N(ext),B(order),T(ab),R(ow),
C(ol),M(an),A(uto)?
```

If you respond to the prompt by pressing **F**, the display window will show the formulas contained in the cells instead of the values that result from the formula calculations. If formulas are currently being displayed, pressing **F** will display the values.

If you respond to the prompt by pressing **N**, the cursor will "auto-advance in the "current direction" after the data is entered into a cell. If auto-advance of the cursor is already in effect, then pressing **N** causes no auto-advance of the cursor after the data is entered into a cell.

Pressing **B** will suppress the display of the worksheet border. If you already suppressed the border display, then pressing **B** will restore the border display. ("Border" refers to the column and row designations across the top and down the left side of your display window.)

Pressing **T** activates the Tab mode, or deactivates it if SuperCalc is already in the Tab mode. In the Tab mode, advancing between cells skips all empty or protected cells. Therefore, you can never select a protected or an empty cell as the active cell in this mode.

Options **R**, **C**, **M**, and **A** concern recalculation.

> **R** means recalculate by rows, from the top down. (Rows are recalculated left to right.)

C means recalculate by columns, from the left across. (Columns are recalculated from top down.)

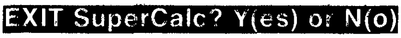
A means recalculation is automatic (default).

M means recalculation occurs at your request, whenever you press the ! key.

/QUIT—*exits from SuperCalc and returns to CP/M*

The /**QUIT** command leaves SuperCalc and relinquishes control to the CP/M operating system. You get a chance to save your work on diskette before the transition occurs.

Prompt:



```
EXIT SuperCalc? Y(es) or N(o)
```

If you reply **Y**, you return to CP/M as indicated by the A> prompt. If you reply **N**, you return to SuperCalc. Any other reply is ignored.

If you have work you could lose when you quit, SuperCalc gives you a chance to save the work before exiting.

Example:

/Quit <**cr**>

/ZAP—*clears the entire worksheet of data*

The /**ZAP** command clears the contents and formatting from the entire worksheet.

Prompt:

```
Y(es) to clear everything, else N(o)
```

All cells become empty. All format settings and modes of operation revert to their standard settings. Everything starts fresh, as if you had just started up the SuperCalc program.

ZAP is the only command that can override protection of cells.

NOTE

Remember, when you ZAP the worksheet, nothing remains.

Examples:

/ZAP, Y

/ZAP, N

SuperCalc Built-In Functions

ABS (value): Provides the absolute value.

AVERAGE (list): Provides the arithmetic mean of the nonblank values in the list.

COUNT (list): Returns the number of nonblank entries in the list.

ERROR, NA: Displays ERROR or NA (not available) for the cell having this function and for any cell with a formula referring to this cell.

EXP (value): Raises "e" exponentially. The value is the exponent.

OR (expression 1, expression 2): Results in "true" (value of 1) if either expression 1 or expression 2 is "true" (nonzero); otherwise, results in "false" (value of O).

AND (expression 1, expression 2): Results in "true" (value of 1) if both expression 1 and expression 2 are "true" (nonzero); otherwise, results in "false" (value of O).

NOT (expression): Results in "true" (value of 1) if expression is "false" (zero); otherwise, results in "false" (value of O).

IF (expl,exp2, exp3): If expression 1 is true, then use expression 2; otherwise, use expression 3. Expression may be combined with AND or OR NOT to form expression 1.

INT (value): Returns integer portion of value. The value is not rounded. Do not confuse this function with **/FORMAT,I** which will round off numerical entries.

LOOKUP (value, column/row range): Searches the range for the last value less than or equal to the search value given. Returns the adjacent value from the column to the right of the search column or the row below the search row. Assumes the search range is in ascending order of values.

Ln (value), LOG 10 (value): Provides the natural log, log base 10, of the value.

MAX (list), MIN (list): Provides the maximum or minimum value in the list.

NPV (discount, column/row range): Nets the present value of a group of cash returns at the given rate of discount. The cash amounts are assumed to be projected for equal time periods, such as every year; and the discount rate is for that interval. The first cash entry is discounted once, the second twice, and so forth, and added to form the total value.

PI: Returns Pi to 16 significant digits.

SIN (value), ASIN (value), COS (value), ACOS (value), TAN (value), ATAN (value): Trigonometric calculation of the value. ASIN is arcsine, etc. Trigonometric results are give in radians.

SQRT (value): Returns the square root of the value.

SUM (list): Returns the sum of the values in the list. Here is a quick explanation of what "value," "range," and "list" mean in this context: Value is a constant, the value of a cell, or a combination of these values made by using the arithmetic operators.

Formulas and Functions

Formulas specify calculations and comparisons. Formulas use values in other cells (which may be themselves the result of formulas), constants, and built-in functions. These values are combined using arithmetic and relational operators:

+	addition
-	subtraction
*	multiplication
/	division
^	raising to a power
=	is equal to
<>	is not equal to
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to

Examples:

Constants: **12,5.9,3.4e3**

Cell values: **A12, B19, BK54**

Combinations: **12+5.9,B19-3,7,A12*B14,(9+E5)/4**

The combinations are also called "expressions." They are evaluated from left to right; * and / are evaluated before + and -. Use parentheses to group terms in your expressions so that SuperCalc will evaluate them as you wish. Some examples follow:

5+4*3+1=18 (that is, 5+12+1)

(5+4)*3+1=28 (that is,9*3+1)

5+4*(3+1)=21(that is, 5+4*4)

(5+4)*(3+1)=36 (that is, 9*4)

Here are some examples of functions with values:

BS(A12),SQRT(9.5*E7),LN(3.5e4),TAN(C5+E5)

Range is simply a partial column or partial row, such as B4:B12 or B4:H4. Here are some examples of functions that use both a value and a range:

LOOKUP(7,C5:J5)

LOOKUP(A4,D3:d12),

NPV(.18,D12:H12)

NPV(B4,G3:G8)

A list can have values, expressions, and ranges. Here are some examples:

SUM(A12,B9,D5)

SUM(C12:E12,H3:H7)

SUM(MAX(C12:E12)

COUNT(E3:E12,F8:J8)

AVERAGE(B7,B8:h8,C12:C20)

Sample worksheets provided on your SuperCalc disk give examples of these formulas in actual use. This material will help you understand how you can put the formulas to work; it is especially useful for **IF**, **LOOKUP**, and **NPV**.

Practical Suggestions

1. Keep your work in the upper left of the worksheet grid.
2. Keep your work in a rectangular shape. Try to avoid having long columns or rows projecting outside the basic shape.
3. Do not blank cells, protect cells, or format cells in the area below or to the right of the area that you actually need. Especially, do not put data below or to the right of the area you actually need.
4. When you have extra or interim work on the screen that you can get rid of, use the following procedure to free that space completely:

   a. /**DELETE** or /**BLANK** the material you do not need.
   b. Move the rest of the work to the upper left of the grid, and adjust it as you wish it to display.
   c. /**SAVE** your work.
   d. /**ZAP** the screen.
   e. Reload. You are now using the minimum space required for your worksheet.

Worksheet Display

A command, text, or formula too long for the entry-line information on the entry line will scroll left when it reaches the end of the line. You can enter a command, text, or formula that is too long to display in its entirety. You can then use the in-line editor to examine any part of the entry by moving the cursor to the left or right. The information will scroll to show the hidden part of the line. When you want to enter the line, press **RETURN**. SuperCalc will take the entire entry, not just the portion to the left of the cursor.

Column Width Greater Than Screen Width

You may sometimes want to make the width of a column greater than the width of the screen. In such cases, you can scroll to see all of the display. If you have a printer with a wide carriage, you can use the output command to print the full width of the information. This feature can be useful for long text notes, explanations, or graphic display of numeric values.

To See the Same Information in Different Formats

The window command lets you look at the same information simultaneously in different formats. Split the single display window into two smaller windows. After you have split the screen, you can move one window so that it shows the same information as the other. Each part of the screen can have its own format settings for entries, rows, columns, or the entire worksheet. Each can have its own GLOBAL options settings. By using this technique, you could display both values and formulas for the same cell contents.

When you set formats or GLOBAL options for a split screen, remember that the portion above or to the left of your screen is "dominant." That is, when you cancel the split, the settings that were in effect for the upper or left window will remain in effect for the entire single display window.

Building Worksheets

Combining worksheet portions to build entirely new worksheets is possible. The /**SAVE** command saves the entire worksheet, but the /**LOAD** command can load all or part of a worksheet. It can place the part loaded at any worksheet location. This means that you can construct the nucleus of a new worksheet from parts of one or more existing worksheets.

When you have a fully developed worksheet with data, you can save it both with and without data. For example, you have developed a monthly report, which you save. Then you blank all the variable contents of the report, which you save. Then you blank all the variable contents of the report and save only the information that will not change, such as: titles, formatting, the general layout of the sheet formulas, and any constant values. Next month you can load this file, fill in the new information, and save it as your current monthly report.

Using PROTECT to Build New Worksheets from Old

The /**BLANK**, /**COPY**, /**LOAD**, and /**REPLICATE** commands all bypass protected cells, leaving their contents unchanged while changing surrounding cells. You can use this capability to combine information in detail, protecting key information and then surrounding it with new information by using LOAD, COPY, or REPLICATE.

Summing a Partial Column or Partial Row

When developing a worksheet, you may often insert new columns or rows within a range covered by a SUM formula. This can be awkward. Inserting or deleting at the top or bottom of an existing column or at the left or right of an existing row can mean redoing your formula. For example, you wish to insert a new row 12 and have to change the formula SUM(C2:C12) to SUM(C2:C13).

Here is a way to avoid this difficulty. Include a header or title at the top or left and an extra cell at the bottom or right within your sum. For a column, the extra cell could have "--------- as a total line. For example:

```
                                 C
 1: January Receipts
 2:                                                  35
 3:                                                 405
 .:
 .:
 .:
 9:                                                  38
10: ______________________________________________________
11: SUM(C1:C10)
```

Text C1 and C10 have a zero value. Including them in the sum makes no difference. You can insert or delete rows from 2 through 9 and have the SUM formula automatically adjust to the new situation.

Security

Security includes protecting your work from accidental loss or change and protecting confidential information in your worksheet.

Protecting Your Worksheets

The CP/M operating system allows you to specify files or entire disks as "read only." Designating your worksheet files this way means others can examine them or print reports from them, but cannot change or erase them.

The SuperCalc option to save values only offers another protection. Your full worksheet may have important proprietary information within its formulas or lookup tables.

After you have saved a full copy for yourself, you can save a Values-Only worksheet for others to use. In that worksheet, you may wish to remove lookup tables.

Similarly, you can use the output command to put a Values-Only copy of selected portions of your worksheet on a disk file for others to use. They can print that file or use the system text editor to include it in their own text file.

Save Your Work Often

It is important to save your work frequently while you are entering data or building worksheets. This practice insures you against losing the time and effort you have invested. It protects you against problems that are completely out of your control —such as power failures or hardware problems with your disk drive.

The Update option of the /**SAVE** command gives you a convenient way to do this. Every time you save your work, use the same name—for example, TRIALBAL. The first time you save your work, it is stored on the disk as TRIALBAL.CAL. The second time you save it, SuperCalc will tell you that there is a file of that name and ask you what you want to do. If you choose the Update option, your new worksheet will be saved as TRIALBAL.CAL and the earlier one will become TRIALBAL.BAK, your backup file. Whenever you use the Update option after that, SuperCalc will give you the two most recent files as filename.CAL and filename.BAK; it will erase any earlier files.

Having the backup file can be convenient; you may want to go back to that file in case a change does not work out in actual operation. You can use CP/M operations to change your file names so that filename.BAK becomes filename.CAL. Or you can directly load the file by giving its full name including the .BAK extension.

Standard or Default Settings

SuperCalc uses standard settings for display and formatting and standard modes of reference. These are also called default settings or modes. You can change these settings by choosing among the available options described earlier. For convenience, here is a list of the standard settings and standard modes.

You can change the following default settings by using the **/FORMAT** command:

Column Width: 9

Numeric Display:
Right-justified.

Standard numeric format. (Cells that contain formulas will have their values displayed; if the number is too large to fit into the column, the number will be displayed in scientific notation.)

Text Display:
Left-justified.

You can change the following default settings by using the **/GLOBAL** command:

> **Border Display:** Row numbers (1–254) and column designations (A–BK) are always displayed. (When the screen is split, the row numbers and column designations are displayed for both windows.)
>
> **Calculation:** Automatic calculation takes place upon reception of new or altered data followed by **RETURN**.
>
> **Order of Calculation:** Calculation is performed by rows, from left to right and top to bottom.

Numeric Display: Standard numeric display. (Cells that contain formulas will have their values displayed.)

Tab Mode: The tab mode is inactive: The cursor advances to the next cell in the current cursor direction.

Automatic Cursor Advancing: Auto-advance mode is active. The cursor will advance to the next cell in the current cursor direction after data entry followed by **RETURN**.

Additional Standard Operations: When you execute a /COPY or /REPLICATE command, formulas with references to other cells automatically adjust to their new locations unless you choose an option provided for these commands.

CBASIC

Labels and Identifiers

CBASIC labels and identifiers must have 31 characters or less. If the last character is a %, then an integer numeric value is assumed. If the last character is a $, then a string variable is assumed. Otherwise, a real numeric variable is assumed.

Numbers:

Integer numbers can have values ranging between −32768 and 32767.

Real numbers are represented using standard decimal format; they can have up to 14 digits of precision. Larger numbers are represented using scientific notation. The mantissa has one digit in front of the decimal point. The exponent is a two-digit decimal number.

Line Numbers:

CBASIC statements do not need line numbers. Line numbers, where present, do not have to be in numeric sequence. Line numbers may be integers, real, or scientific-notation numbers. Line numbers with the same numeric value, but a different numeric type are treated as a different line number. For example, the following three line numbers have the same numeric value, but are treated as three different line numbers:

```
100
100.0
1.0E02
```

Expressions:

CBASIC evaluates expressions using the following evaluation hierarchy from highest to lowest:

1. Nested parentheses ( )

2. Exponent ^
3. Multiply *, divide /
4. Add +, subtract −, concatenate +, unary plus +, usary minus −
5. Relational operators LT or <, LE or < =, GT or >, GE or > =, EQ or =, NE or <>
6. NOT
7. AND
8. OR,XOR

Summary of CBASIC Statements

CBASIC statements are summarized below in alphabetic order. Each statement's format is given, along with an example.

CALL—*links to a subroutine*

The CALL statement calls an assembly-language subroutine. Also see SAVE, PEEK, and POKE statements.

Format:

CALL integer expression

The integer expression must evaluate to the absolute memory address of the entry point for the assembly-language subroutine being executed. An assembly-language return (RTN) instruction (executed out of the subroutine) will return execution to the next sequential CBASIC statement.

NOTE

CPU registers may be altered by the assembly-language subroutine.

Examples:

```
CALL 27248
CALL 2CA0H
CALL sub%
```

CHAIN—*transfers control from one program to another*

The CHAIN statement is used to transfer control from one executing CBASIC program to another.

Format:

CHAIN string expression

The string expression identifies the file name (and drive) for the next program to be executed. The selected file must be of type INT and it must exist on the specified drive. When no drive is specified, the active drive is assumed. A CHAIN statement causes the selected file to be loaded and the program held within the file to be executed. In addition, the return stack is reset, open files are closed, and a RESTORE statement is executed. A COMMON statement can be used to exchange data between chained programs.

Examples:

```
CHAIN "MAIN"
CHAIN DRIVE$ + ":"+ PRINT.MSG$
```

Sample Programs:

```
REM TRANSFERS CONTROLTO A PROGRAM OF
REM THE USER'S CHOICE DIRECT COMPILER
REM TO RESERVE EXTRA SPACE FOR
REM PROGRAM'S CONSTANT, CODE, DATA,
REM AND VARIABLE AREAS (32, 1000, 32, AND
REM 32 BYTES RESPECTIVELY) (TO PREVENT
REM OVERWRITING BY THE CHAINED
REM PROGRAM)

%CHAIN 32, 1000, 32, 32

INPUT "WOULD YOU LIKE TO RUN A PROGRAM (Y/N)?";RUN$

        IF RUN$ = "Y" THEN\
          INPUT "WHICH ONE?"; PROG.NAME$:\
CHAIN PROG.NAME$

  STOP

REM PROGRAM CHECKCHAIN—WHEN MAIN
REM PROGRAM CHAINS HERE A MESSAGE IS
REM PRINTED TO SHOW THE CHAIN WAS
REM SUCCESSFUL AND THEN CONTROL IS
REM TRANSFERRED BACK TO THE MAIN
REM PROGRAM

  PRINT "YOU HAVE SUCCESSFULLY CHAINED"
  PRINT "TO PROGRAM CHECKCHAIN"
  CHAIN "MAIN"
  STOP
```

CLOSE—*closes specified files*

The CLOSE statement closes open files.

Format:

CLOSE integer expression {**,integer expression**}

Each integer expression denotes an open file that is to be closed. Reference to a file that has not been opened will cause an error. When a file is closed, the file number is released, and the associated buffer space is returned to the system.

NOTE

CLOSE terminates any IF END statement that references the file being closed.

Examples:

```
CLOSE 1
CLOSE input.file.id%, temp.file.1%
```

Sample Program:

```
REM CREATE TWO NEW FILES, WRITE DATA TO
REM THEM, AND CLOSE THE FILES
  CREATE "NEWFILE.SEQ" AS 1
  CREATE "NEWFILE.RAN" RECL 25 AS 2
  FOR 1% = 1 TO 10
          PRINT #1; "RECORD NUMBER",1%
          PRINT #2,1%; "RECORD NUMBER",1%
  NEXT 1%
  CLOSE 1,2
```

COMMON—*specifies common variables*

The COMMON statement specifies simple and subscripted variables that are retained in a common area of memory and passed between chained programs.

Format:

COMMON variable {, variable}

NOTE

For a subscripted variable, the number of subscripts, not the actual subscript maximum dimensions, follows the parenthesis.

COMMON statements must be the first in a program, preceded only by REM statements or blank lines. Chained programs must begin with COMMON statements having coincident parameter lists. Common variables must be of the same type in all COMMON statements, and they must appear in the same sequence. Variable arrays must have the same number of dimensions, and each dimension must have the same maximum value.

Examples:

```
COMMON DATE$, NAME$,ACCOUNTS$(3)
COMMON SIZE%,ACCOUNT.LIMIT (2), COMPANY$
```

Sample Program:

```
REM TRANSFER CONTROL TO A PROGRAM OF
REM THE USER'S CHOICE

COMMON RET$

REM DIRECT COMPILER TO RESERVE EXTRA
REM SPACE FOR PROGRAM'S CONSTANT,
REM CODE, DATA, AND VARIABLE AREAS (32,
REM 1000, 32, AND 32 BYTES RESPECTIVELY)
REM (TO PREVENT OVERWRITING BY THE
REM CHAINED PROGRAM)

%CHAIN 32, 1000, 32, 32

  INPUT "WOULD YOU LIKE TO RUN A PROGRAM (Y/N)?";RUN$
            IF RUN$ = "Y" THEN \

        INPUT "WHICH ONE?"; PROG.NAME$:\
        INPUT "DO YOU WANT TO RETURN TO THIS PROGRAM?";RET$:\
        CHAIN PROG.NAME$

  STOP

REM PROGRAM CHECK2—WHEN MAIN
REM PROGRAM CHAINS HERE A MESSAGE
REM IS PRINTED TO SHOW THE CHAIN WAS
REM SUCCESSFUL AND THEN CONTROL IS
REM TRANSFERRED BACK TO THE MAIN
REM PROGRAM IF THE USER CHOSE THAT
REM OPTION

COMMON RET$

  PRINT "YOU HAVE SUCCESSFULLY CHAINED"
  PRINT "TO PROGRAM CHECK2"
  IF RET$ = "Y" THEN \
        CHAIN "MAIN2"

STOP
```

CONSOLE—*redirects print to the console*

The CONSOLE statement follows an LPRINTER statement and causes output to be diverted from the printer to the console. This statement can also be used to make console width adjustments.

Format:

CONSOLE

Following execution of a CONSOLE statement, PRINT statement output is directed to the console.

To adjust the console width, use the POKE statement. POKE the required character width to location 272. The new console width becomes effective after the next execution of a console statement. A zero width setting is considered infinite, so that new lines are never automatically started. The default console width is 80 characters.

Example:

CONSOLE

Sample Program:

```
LPRINTER
PRINT "THIS LINE WILL BE OUTPUT TO"
PRINT "THE PRINTER"

CONSOLE

PRINT "THIS LINE SHOULD APPEAR"
PRINT "ON THE CONSOLE"

LPRINTER WIDTH 30
PRINT "THE PRINTER WIDTH WAS SET TO 30"
PRINT "SO THIS SENTENCE";
PRINT "SHOULD BE PRINTED AS 4 LINES BY"
PRINT "THE PRINTER"

POKE 272, 30
CONSOLE

PRINT "THIS SHOULD APPEAR ON THE"
PRINT "CONSOLE SCREEN";
PRINT "THE POKE STATEMENT SET THE LINE"
PRINT "WIDTH TO 30"

STOP
```

CREATE—*creates a new file*

The CREATE statement is comparable to an OPEN statement except that it is used to activate a new file rather than an existing file. Any existing file with the same name is erased so that a new file can be created.

Format:

CREATE string expression
[RECL integer expression]
AS integer expression
[BUFF integer expression]
[RECS integer expression]

The expression following the keyword CREATE is a valid file name that identifies the file to be created. The integer expression following AS designates which of 20 available file numbers is assigned to the new file (see the READ#, PRINT#, IF END, and CLOSE statements). Each active file is assigned an identification number. This number is used in all subsequent references.

The integer expression following RECL specifies record length. When the length is specified, the file will contain fixed-length records that may be accessed randomly or sequentially. If no length is specified, then record length will vary, and the file must be accessed sequentially.

The BUFF and RECS portion of a CREATE statement either appear in conjunction, or are omitted. The integer expression following BUFF indicates how many disk sectors are to be maintained in memory. When using random access, you must specify one disk sector. A value of 1 is automatically assumed in the absence of the BUFF and RECS parameters. RECS identifies the size of a physical sector on the disk. This value is currently ignored and the system assumes a sector size of 128 bytes. (See the CLOSE statement for a CREATE programming example.)

DATA—*lists data constants*

A DATA statement defines constants. These constants are assigned to variables by READ statements.

Format:

DATA constant [,constant]

The DATA statement may be used to list string, integer, or real constants. DATA statements can be located anywhere in a program, except before a COMMON statement. Each DATA statement must occupy one line exclusively and cannot be continued on the next line. No other statement can appear on the same line with a DATA statement.

A list of constants is compiled from all DATA statements in a program. Constants are put into the list sequentially as they appear in the DATA statement(s) parameter list(s), in the same order that the DATA statements themselves appear. READ statements work down the list of constants, assigning the next sequential constant in the DATA list to the next variable in the READ statement parameter list. An attempt to read past the DATA statement list will cause a runtime error.

Example:

110 DATA 1, 2, 2.1, 22.1

Sample Program:

```
REM READ AND PRINT THE DATA LIST
DATA 123.987, 42, "MORE DATA", "EVEN MORE DATA"

READ REAL.NUM, INT,NUM%, STRING1$, STRING2$
PRINT REAL.NUM; STRING1$; INT.NUM%; STRING2$

STOP
```

DEF— *defines a line function*

The DEF statement defines either a single line function or multiple line function. The DEF keyword must appear before any reference to the actual function.

Format:

DEF FN.name [(parm {,parm})] =expression

This format illustrates a single line function. A function name always begins with FN.; the remainder of the function name may consist of any combination of letters, numbers, or decimal points including blanks. The function name must be a valid CBASIC variable name.

The function computes a value, which it assigns to the function name. Subsequently the function name is treated as a CBASIC program variable. The function name determines whether the function is of type real, integer, or string.

In a single line function the expression to the right of the equal sign is evaluated, and the result is returned via the function name.

Parameters ("parm") are listed within the expression. When the function is referenced in the body of the program, actual values must be specified for each parameter. These actual values are used when the expression is evaluated.

Format:

```
DEF FN.name [(parm {,parm})]
```

Multiple line functions begin with a DEF statement and end with a FEND statement. The DEF statement specifies the function name and any function parameters. Any group of statements can occur between the DEF and FEND statements. A RETURN statement must be present to terminate the multiple line function, in a manner analogous to a subroutine return. The value returned by the multiple line function is the last value assigned to the function name before the RETURN statement is executed.

Examples:

```
DEF FN.CIRCLE.AREA (R)=3.142 * (R^2)
THIS.AREA=FN.CIRCLE.AREA (5)
```

Sample Program:

```
DEF FN.CIRCLE.AREA (R)=3.142 * (R * R)
DEF FN.CYLINDER.VOLUME(RADIUS,HEIGHT)
BASE=FN,CIRCLE.AREA (RADIUS)
FN.CYLINDER.VOLUME=BASE * HEIGHT
RETURN
FEND

INPUT "CIRCLE RADIUS?";RAD
THIS.AREA=FN.CIRCLE.AREA (RAD)

INPUT "CYLINDER RADIUS?" RAD
INPUT "CYLINDER HEIGHT?"; HEIGHT
THAT.VOLUME=FN.CYLINDER.VOLUME (RAD,HEIGHT)
PRINT "THE CIRCLE'S AREA IS:", THIS AREA
PRINT "THE CYLINDER'S VOLUME IS:", THAT.VOLUME

STOP
```

DELETE—*erases file entry from the directory*

DELETE removes the indicated files from their respective directories.

Format:

DELETE <expression> {,<expression>}

Each numeric expression indicates the assigned number of an active file. Real values are converted to integer; string values cannot be used. Any IF END statement associated with the file number being deleted will no longer be valid.

Examples:

DELETE 3,2,1
DELETE FILE.NO%,OUTPUT FILE.NO%

DIM—*allocates storage for an array*

The DIM statement allocates storage for an array and defines the upper limit of each subscript; a lower bound limit of zero is assumed. Execution of each DIM statement allocates a new array. If the current array is numeric, it causes the previous array to be deleted, freeing space for the new one. Each element in a string array must be set to null before reexecution of DIM, to regain the maximum amount of storage.

Format:

DIM exp (subscript list) ,exp (subscript list)

Space is dynamically allocated for numeric and string arrays. Elements of string arrays may be any length up to 255 bytes, and change in length as they assume different values. Numeric arrays are initially set to zero while elements of string arrays are null.

The subscript list specifies the number of dimensions and extent of each dimension of the array being declared. The subscript list may not contain a reference to the array being dimensioned.

Examples:

```
DIM A (10)
DIM NAME$ (50), ADDRESS$ (100),
DIM A% (1,2,3,), SALES% (QUOTA%)
DIM X (A%(B%),C%,D%)
```

FEND—*teminates a multiple line-function definition*

CBASIC should never execute a FEND statement; all multiple line functions must end execution with a RETURN statement, otherwise an error will occur. See the DEF statement for more details.

Format:

FEND

FOR—*initiates a FOR/NEXT loop*

The FOR statement establishes a loop index, an initial index value, a termination index value, and an amount by which the index is increased on each iteration through the loop. A NEXT statement is used to terminate the loop.

Format:

FOR index=numeric exp TO numeric exp [STEP exp]

Index must be an unsubscripted variable. The expression following the equal sign is evaluated and then assigned to the index, thus establishing the initial index value for the FOR/NEXT loop.

The expression following TO is the loop termination value. A positive step value causes the loop to execute until the index is greater than the termination value. A negative step value executes the loop until the index is less than the termination value. The type of the termination expression must match the type of the index.

The expression following STEP is the loop increment value. When STEP is omitted, a value of 1 is assumed. The increment is added to the index on each execution of the loop prior to comparing the index with the termination value.

Program speed can be maintained by using an index and expressions of type integer, and by omitting the STEP increment expression when a value of 1 is intended.

NOTE

Statements within a FOR/NEXT loop are always executed at least once.

Examples:

```
FOR 1%=1 TO 1000
FOR J=12.0 TO 123.67 STEP 1.6
```

Sample Program:

```
PRINT "THIS PROGRAM FINDS THE AVERAGE"
PRINT "OF NUMBERS YOU INPUT"
INPUT "HOW MANY NUMBERS?", LAST %

TOTAL=0.0 REM INITIALLY SET TOTAL TO ZERO
FOR INDEX%=1 TO LAST%
PRINT "NUMBER";INDEX%;
INPUT NEXT.NUMBER
TOTAL=TOTAL+NEXT.NUMBER
NEXT INDEX%
AVERAGE=TOTAL/LAST%
PRINT "THE AVERAGE IS";AVERAGE
STOP
```

GOSUB—*causes execution of a subroutine*

The GOSUB statement executes a subroutine. The subroutine is identified via a statement line number. Execution returns to the statement following GOSUB after the subroutine has completed execution.

Format:

GOSUB statement number

The statement number specifies the subroutine entry point.

NOTE

CBASIC can nest subroutines to a depth of 20.

Examples:

GOSUB 200
GOSUB 100.001

Sample Program:

```
REM USE A SUBROUTINE TO MAKE
REM CORRECTIONS

        INPUT "WHAT STRING WILLYOU PRINT?";STRING$
        GOSUB 300       REM MAKE CORRECTIONS
        GOSUB 350       REM PRINT THE STRING

        STOP

300 REM CORRECTIONS SUBROUTINE
        INPUT "WANT TO MAKE CORRECTIONS (Y/N)?";ANS$
        WHILE ANS$="Y"
                INPUT "NEW STRING?";STRING$
                INPUT "IS IT STILLWRONG(Y/N)";ANS$
        WEND
        RETURN

350 REM: PRINT SUBROUTINE
        LPRINTER
        PRINT STRING$
        CONSOLE

        RETURN
```

IF END# — *establishes a conditional file-access branch*

The IF END# statement prepares program logic for conditional execution of a branch during a subsequent file access, providing certain conditions are detected.

Format:

IF END# integer expression THEN statement number

The IF END# statement is unusual in that it prepares program logic for a future conditional branch. Only bookkeeping operations are performed when the IF END# statement is executed. The "integer expression" must be a number between 1 and 20. This is a file number that links the IF END# statement with subsequent file-access statements. The statement number specifies the program line to which the conditional branch will occur if certain conditions are encountered when the subsequent file-access statement is executed. These are the conditions that can cause the IF END# branch:

1. On reading from a file—if the End Of File is detected.
2. On writing to a file—if there is no more disk space.
3. On any file access—if the named file does not exist.

Any number of IF END# statements may appear in a program. The conditional branch specified by the most recent IF END# statement with the same file number gets executed.

IF END# must be the only statement on a line. It cannot be followed by a colon and additional statements, nor can other statements precede it on the same line.

NOTE

When a file is deleted or closed, IF END# statements having the deleted or closed file number are deactivated.

Examples:

```
IF END# 7 THEN 700
REM read a file from #7. On detecting the
REM end of file branch to statement 700
READ #7 ,A,B,C,D
```

Sample Program:

```
REM ADD A NEW RECORD TO NEWFILE.SEQ
REM: OPEN FILE — IF FILE DOES NOT EXIST
REM GO TO 900
        IF END# 10 THEN 900
            OPEN "NEWFILE.SEQ" AS 10

    REM: FIND END OF FILE
    IF END# 10 THEN 200

100 READ # 10; STRING$,NUMBER%
        GOTO 100

200 PRINT # 10;STRING$,NUMBER% +1
        CLOSE 10
        STOP

900 PRINT "ERROR—FILE DOES NOT EXIST"
        STOP
```

IF-THEN-ELSE—*provides a conditional branch*

IF-THEN-ELSE conditionally executes statements depending on the value of an expression. When the conditional expression following IF is true, the group of statements following THEN executes; otherwise the group of statements following ELSE is executed. An expression is "true" if it has a value other than zero.

Formats:

IF integer expression
THEN group of statements
[ELSE group of statements]

The ELSE portion of an IF statement is optional. When ELSE is omitted and the integer expression is false, execution continues with the next sequential statement.

NOTE

A group of statements consists of one or more CBASIC statements separated by colons. The following statements cannot be in this group: DATA, DEF, DIM, IF, and IF END.

The following format is compatible with other BASIC languages:

IF integer expression THEN
statement number

This form of the IF END statement treats the statement number as a GOTO statement. This form does not allow use of the ELSE option.

Sample Program:

```
REM THIS PROGRAM ASKS FOR USER'S NAME
REM AND CHECKS FOR ERRORS

100 INPUT "WHAT IS YOUR NAME?";LINE NAME$

        PRINT NAME$;
        INPUT "-IS CORRECT (Y/N)?";ANSWERS


200 IF ANSWER$ <> "Y" AND ANSWER$ <> "N"\ INVALID ANSWER
INPUT "TYPE " "Y" " FOR YES OR " "N" " FOR NO"; ANSWERS$:\
GOTO 200

IF ANSWER$ = "N" THEN GOTO 100 \ INCORRECT NAME
ELSE PRINT "THANK YOU"\  CORRECT
STOP
```

INITIALIZE—*Reinitializes the operating system*

The INITIALIZE statement must be executed whenever the diskette in a drive is replaced during program execution. Never change a diskette while any files are open. If you do, data may be lost from open files, and the directory will not be updated.

Example:

INITIALIZE

INPUT—*assigns data to variables*

The INPUT statement receives data from the console input device and assigns it to variables.

Format:

INPUT ["message";] variable {,variable}

The message is an optional string of characters that is printed as a prompt before the computer accepts data from the console; the message must end with a semicolon. In the absence of a prompt, a question mark appears which indicates that input data is expected. The prompt string (or question mark) is followed by a blank character. You must enter data in response to the prompt or question mark.

NOTE

Prompt strings are directed to the console input device even when an LPRINTER statement is in effect.

INPUT statement variables may be simple or subscripted, string or numeric. Data items you enter at the keyboard must be separated by commas. The last data item must be followed by a carriage return. Strings may be enclosed in quotation marks, in which case commas and leading blanks can be part of the string.

There must be a data item for each variable present in the INPUT statement; otherwise CBASIC will request that all data be reentered. The message "IMPROPER INPUT—REENTER" is displayed if too many or too few data items are entered.

You can enter a maximum of 255 characters in response to an INPUT statement. Data will be ignored after this 255-character limit is reached.

If **^Z** is the first character you entered, the program terminates as if a STOP statement had been executed. All CP/M line-editing functions such as **^U** and **^R** are active while data is being input in response to an INPUT statement. (See IF-THEN-ELSE for a sample program.)

Example:

INPUT "Enter three values:" ;A%,B%,C%

INPUT LINE—*assigns console entry to a string variable*

INPUT LINE is a special form of the INPUT statement that reads an entire keyboard entry and assigns it to a single string variable.

Format:

INPUT ["message";] LINE string variable

The prompt is handled as described for the INPUT statement. Only one string variable can follow the LINE keyword.

Entered data is assigned to the string variable. Input terminates with a carriage return. Commas and spaces count as characters to be included in the string.

You can enter a null string by responding to an INPUT LINE statement with a carriage return. Up to 255 characters can be entered; additional characters are ignored.

An INPUT LINE statement does not generate an "IMPROPER INPUT" message. (See IF-THEN-ELSE for a sample program.)

Examples:

```
INPUT "Abort (Y or N)?";LINE ANS$
INPUT "Press RETURN to continue"; LINE DUMMY$
```

LET—*assigns a value to a variable*

The LET statement evaluates an expression and assigns the result to a variable. The LET keyword is optional.

Format:

```
[LET] variable = expression
```

The variable may be simple or subscripted. You must specify a string variable for a string expression. Integer or real variables may be specified for numeric expressions. The type of a numeric expression will convert to agree with the type of the numeric variable.

Examples:

```
LET X=3
AMOUNT=COST*QTY%
NAME$(L%)=FIRST(j%)+LAST$(k%)
```

LPRINTER—*directs print from console to printer*

The LPRINTER statement directs PRINT statement output to the printer (see the CONSOLE statement).

Format:

```
LPRINTER [WIDTH integer expression]
```

The optional WIDTH parameter sets the printer line width. The initial width is set at 132 characters. A carriage return is automatically output following 132 characters or the specified line width. If you set the WIDTH to zero, an infinite width is assumed and no automatic carriage returns are output. (See the CONSOLE statement for a sample program.)

Examples:

```
LPRINTER
LPRINTER PRINTER.WIDTH%
LPRINTER.WIDTH 0
```

NEXT—*terminates a FOR/NEXT loop*

The NEXT statement terminates one or more FOR/NEXT loops. (See the FOR statement.)

Format:

```
NEXT [index {,index}]
```

A NEXT statement without any index parameters terminates the most recently encountered FOR statement loop. The index is optional; if present, it must match the index of the FOR statement for the loop being terminated. If the index does not match, an error will occur. You can terminate more than one FOR statement by listing multiple indexes in the NEXT statement. (See the FOR statement for a sample program.)

ON GOTO—*executes a GOTO statement with a choice of destinations*

The **ON GOTO** statement executes a GOTO, selecting a destination statement number that depends on the value of an expression.

Format:

ON integer expression GOTO statement number list

The statement number list consists of one or more statement line numbers. The integer expression selects one of these statement numbers. If the integer expression is 1, then the first number is selected; if the integer expression is 2, then the second number is selected, and so on.

The integer expression must evaluate to a number ranging between 1 and the number of statement numbers in the statement number list. If the integer expression has any other value, then an error will occur. An error will also be reported if the statement number in the statement number list does not exist and the program logic, therefore, does not know where to branch.

Example:

```
ON I% GOTO 199, 200,300
```

When the statement shown above is executed, if I % is 1, then a branch to line 199 will occur. If I % is 2, a branch to line 200 will occur; and if I % is 3, then a branch to line 300 occurs. In the context of this scenario, line numbers 199, 200, and 300 must exist, and I% must be evaluated as 1, 2, or 3. Here is another example:

On CODE% 3 GOTO 33.1, 33.2, 33.3

Sample Program:

```
REM THIS EITHER CHANGES OR PRINTS THE
REM INPUT DATA STRING UNTILTHE USER
REM TELLS IT TO QUIT

100      INPUT "INPUT DATA STRING:";
         STRING$

125      PRINT "YOUR OPTIONS ARE:"
         PRINT "   1........CHANGE THE STRING"
         PRINT "   2........PRINT THE STRING"
         PRINT "   3........QUIT"
         INPUT "WHICH DO YOU WANT?";NUMBER%
         IF NUMBER%0 AND NUMBER% <4 THEN\
                  ON NUMBER\  GOTO 100, 150, 175\
                  ELSE GOTO 125

150      LPRINTER: PRINT STRING$:CONSOLE

175      STOP
```

ON GOSUB—*executes a GOSUB statement with a choice of subroutines*

The ON GOSUB statement is similar to ON GOTO, described above, except that a subroutine is called at the specified statement.

Format:

ON integer expression
GOSUB statement number list

The statement number list consists of one or more line numbers that are subroutine entry points. The integer expression determines which subroutine gets called. If the integer expression is 1, the first subroutine is called. If the integer expression is 2, the second subroutine is called, and so on. The integer expression must have a value that ranges between 1 and the number of subroutine entry points in the statement number list.

Example:

ON I% GOSUB 199, 200, 300

Sample Program:

```
REM THIS PROGRAM EITHER CHANGES OR
REM PRINTS THE INPUT DATA STRING UNTIL
REM THE USER TELLS IT TO QUIT

  INPUT "INPUT DATA STRING:";STRING$
  MORE% = 1
  WHILE MORE% = 1

    PRINT "YOUR OPTIONS ARE:"
    PRINT " 1.......CHANGE THE STRING"
    PRINT " 2.......PRINT THE STRING"
    PRINT " 3.......QUIT"

    INPUT "WHICH DO YOU WANT?";NUMBER%

    IF NUMBER%>0 AND NUMBER%<4 THEN\
      ON NUMBER%GOSUB 210, 220, 230

  WEND
  STOP

210         INPUT "WHAT STRING WOULD YOU LIKE/";STRING$
            RETURN

220         LPRINTER:  PRINT STRING$:  CONSOLE
            RETURN

230         MORE% =0          REM NO MORE


STOP
```

This is almost identical to the ON GOTO example shown previously. In the ON GOSUB case, 199, 200, and 300 are line numbers within subroutines. After the selected subroutine executes, program logic will return to the statement directly following the ON GOSUB.

OPEN—*activates an existing file*

The OPEN statement is used to activate an existing file. The OPEN statement is similar to the CREATE statement, except that an existing file is activated rather than a new one.

Format:

OPEN string expression
[RECL integer expression]
AS integer expression [BUFF integer exp]
[RECS integer expression]

The string expression following the keyword OPEN is the name of the file to be activated. The integer expression following RECL specifies record length. When you specify the length, the file will contain fixed-length records that you can access randomly or sequentially. If no length is specified, the record length will vary.

NOTE

An IF END statement associated with the assigned file number will not be affected.

The BUFF and RECS portions of an OPEN statement must either be used in conjunction, or be omitted. The integer expression following BUFF indicates how many disk sectors are to be maintained in memory. When using random access you must specify one disk sector. A value of 1 is automatically assumed without the BUFF and RECS option. RECS identifies the size of a physical sector on the disk. This value is currently ignored. The system currently assumes a sector size of 128 bytes.

Examples:

```
OPEN "PAYROLL.DAT" AS 9
OPEN FILE.NAME$ AS FILE.NO%
OPEN WORK.FILE$(1%)\
RECL LENGTH%\
AS FILE.NO%\
BUFF BS% RECS 128
```

Sample Program:

```
REM OPEN NEWFILE.SEQ AND NEWFILE.RAN.
REM READ AND PRINT THE FIRST FILE OF
REM NEWFILE.SEQ AND THE FIFTH FILE OF
REM NEWFILE.RAN

     REM:   OPEN NEWFILE.SEQ—IF IT DOES
            NOT EXIST GO TO 900
            IF END #3 THEN 900
            OPEN "NEWFILE.SEQ" AS 3

     REM:   OPEN NEWFILE.RAN—IF IT DOES
            NOT EXIST GO TO 900
            IF END #4 THEN 900
            OPEN "NEWFILE.RAN" RECL 25 AS 4
            READ #3: STRING$, NUMBER%
       PRINT "FIRST RECORD OF
       NEWFILE.SEQ:",STRING$;NUMBER%
```

```
            READ #4,5;STRING$,NUMBER%
            PRINT "FIFTH RECORD OF
            NEWFILE.RAN:", STRING$ NUMBER%

            CLOSE 3, 4
            STOP
900         PRINT "ERROR DOES NOT EXIST"
            STOP
```

OUT—*outputs an integer to an I/O device*

The OUT statement is used to output an integer to a selected input/output port.

Format:

OUT integer expression, integer expression

The low-order byte of the second expression is sent to the output port addressed by the low-order byte of the first expression.

Examples:

OUT 3, 80H

OUT TAPE.DATA%,NEXTCHAR%

POKE—*stores a byte in a selected location*

The POKE statement stores a byte of data in a memory location identified by its absolute memory address.

Format:

POKE integer expression, integer expression

The low-order byte of the second expression is stored at the memory location addressed by the first expression.

Examples:

POKE 100H, 225
POKE MSG.ID%, END.MARK%

Sample Program:

```
REM POKE OR PEEK AT LOCATIONS CHOSEN
REM BY USER

    MORE% = 1
    WHILE MORE%
        INPUT "TYPE POKE, PEEK, OR QUIT"; STRING$
        IF STRING$ = "PEEK" THEN\
              INPUT "LOCATION?"; LOC%:\
              VAL% = PEEK (LOC%):\
              PRINT VAL%
        IF STRING = "POKE" THEN\
              INPUT "LOCATION?";LOC%:\
              INPUT "VALUE?";VAL%:\
              POKE LOC%, VAL%
        IF STRING$ = "QUIT" THEN\
              MORE% = 0

    WEND
    STOP
```

PRINT—*outputs data to the console or printer*

The PRINT statement outputs data to the console or printer. (See the LPRINTER and CONSOLE statements.)

Format:

PRINT expression {delim expression}
[delim]

Expressions in the PRINT statements parameter list are printed. Numbers are printed in 15-character columns and are left-justified. Scientific notation is used if a number will not fit within the 15-character column width.

Strings are printed without modifications. If the end of a line is reached part way through a numeric field, then the entire number moves to the next line. A string, on the other hand, may be broken by an automatic carriage return. The string is output until the end of the line is reached, and the remainder of the string is output on the next line.

The delimiter (delim) between expressions may be a comma or a semicolon. A comma causes automatic spacing to the next column divisible by 20. A semicolon causes one blank to be output following a numeric value and eliminates spacing following a string.

The delimeter at the end of a PRINT statement is optional; if present, the last delimeter has the same effect as that described earlier. No carriage return is output if the PRINT statement ends in a delimeter. A delimeter at the end of a statement allows the next PRINT statement to continue where the previous one left off. If you use no delimeter, a carriage return and line feed are output at the end of the print operation.

NOTE

A PRINT statement with no parameter list outputs a carriage return and a line feed.

Examples:

```
PRINT VALUE, AMOUNT
PRINT "the amount owed is" ; COST
PRINT A$; B$, C$,
PRINT
```

Sample Program:

```
REM PRINT X AND Y USING BOTH THE , AND ;
REM DELIMETERS
    X = 123.5463
    Y = 98777777765523253647.5890
    PRINT "X =",X
    PRINT "Y =";Y; "X =";X
    STOP
```

PRINT# —*outputs data to a disk file*

The PRINT# statement functions like the PRINT statement described earlier, except that output is directed to a disk file.

Format:

PRINT# integer expression [,integer expression] ; expression {,expression}

The PRINT# statement consists of two parts: the file and record-selector portion, which is terminated with a semicolon, and a list of expressions. The expressions are separated by commas. No comma should appear at the end of the list, however.

The first expression is a number identifying the file to which data will be output; this file number must be active and can range from 1 to 20.

The second integer expression is optional; if present, it identifies the record number of a random-access file where data will be stored. When the second integer expression is present, the file must have been opened with the RECL parameter specifying record length.

Numbers are output to a diskette without any change in format. Strings are enclosed in quotation marks before being output. A string that is output to a diskette file may not contain quotation marks. Blanks are used to pad fixed-length records of a random-access file so that the line feed is the last character in the record. (See the CLOSE statement for a sample program).

Examples:

```
PRINT# 1; A,B,C
PRINT#DATA%,REC.NO%
;NAME$,STREET$.CITY$,STATE$,ZIP
```

PRINT USING—*outputs formatted data*

The PRINT USING statement provides formatted output to the console or printer. You specify the format using a format expression.

Format:

```
PRINT USING string expression ;
        expression list
```

The string expression consists of data fields that describe the format used to output the expression list. The string expression may also contain literal characters that are output as they occur.

The expression list is as described for the PRINT statement. String field specifications may be used in a PRINT USING statement. These PRINT USING field specifiers are described in the "Print Using Table" and below:

Literal Characters:

Characters that are not part of a string or numeric-field format are assumed to be literal characters. Blank spaces, for example, are frequently used to separate fields.

Examples:

"#### ####"	3 blanks separate two numeric fields.
"NO.##########"	The 3 characters NO. precede a numeric field.

\ treats the next character in the format expression as a literal character. A single literal character is defined using a \ character followed by the single literal character. A \ is generally used to print a control character as a literal.

Examples:

"\ #"	Prints #
"\ $\ #\ !"	Prints ##!
"\ \ \ \ "	Prints \ \

Numeric Fields:

# Numeric digit

An integer numeric field is specified by two or more # characters, one per character of the field width. Numbers are right-shifted within the numeric field, with blanks preceding the most significant digit.

Examples:

"###" specifies a 3-digit numeric field.

"######" specifies a 6-digit integer numeric field.

You specify a real numeric with a decimal point by placing a character within the # character string at the location where the decimal is to appear.

Example:

"###.##" specifies a 5-digit numeric field; 3 digits precede the decimal point and 2 follow.

You can format numeric fields with commas by placing one or more characters anywhere between the first and last # characters. The position of the , character in the numeric field specification is irrelevant; the commas are positioned after every third character.

Example:

"##,#####.##" specifies a number with 8 predecimal digit positions and 2 postdecimal digits. A , character prints every third digit to the left of the decimal point.

^ Exponential numeric format:

You specify exponential numeric format by appending one or more ^ characters at the end of the numeric field definition. The decimal-point position affects the exponent value. Four character positions are always added for the exponent.

Example:

"#,####^" specifies a number printed in exponential format with 1 predecimal digit and 4 postdecimal digits.

**** Fills a numeric field with leading asterisks:**

Two asterisks appearing at the beginning of a numeric field fill unused leading characters with asterisks.

Example:

*"**######.##"* specifies a numeric field with 8 predecimal digits and two postdecimal digits. * characters appear in any leading unused position.

$ Monetary format with dollar sign:

A $ sign will appear in front of the first non-blank digit if the numeric field definition begins with two $ characters. The $ character will not be printed for negative numbers and cannot be used with * fill characters.

Example:

"$$######.##" specifies a numeric field with 8 predecimal and 2 postdecimal digits. A $ sign precedes the first nonblank digit.

– Marks position for a number's sign (leading or trailing):

A leading negative sign is defined by a – character in the first-character position of the numeric field definition. A negative sign at the end of a numeric field is specified if the – character appears in the last-character position of the numeric field definition.

Examples:

"–######.##" specifies a numeric field with 6 predecimal digits preceded by a negative sign for negative numbers, and two postdecimal digits.

"######.##–" specifies a numeric field with 6 predecimal digits and two postdecimal digits followed by a negative sign for negative numbers.

String Variables:

& defines a variable-length string field.

A variable-length string field is specified by a single & character.

Examples:

"&" Specifies a string field of any length.

"&&" Specifies two variable-length string fields separated by a single blank character.

To define a fixed-length string field, place a / character in the first and last field positions. Any characters may appear between the first and last / characters.

Example:

"/...../" specifies a 7-character string field.

! specifies a single string character.

A! specifies a single-character string field.

Example:

"!" prints the first character of a string field.

Sample Program:

```
REM USE PRINT USING TO PRINT A PURCHASE
REM RECORD
    COMPANY$ = "SHANNON'S SUPPLIES"
    FIRST.NAMES$ = "LIST"
    MI$ = "G"
    LAST.NAMES$ = "PATRICK"

PARTNO1% = 1306 : PRICE1 = 1304.57 : QUANT1% =4
PARTNO2% = 1296 : PRICE2 = 23.99 : QUANT2 = 6
CHARGE1 = PRICE 1 * QUANT1%
CHARGE2 = PRICE2 * QUANT2%
TOTAL = CHARGE1 + CHARGE2

FOR$= "RECORD OF PURCHASE BY /FFFFF/!. /LLLLLLLLL/"
FROM$ = "FROM &"
ITEMS$ = "PART /##### : ## AT ##,###.## AMOUNT:/ $$##,###.##"

PRINT USING FOR$; FIRST.NAMES$,MI$,LAST.NAMES$
PRINT USING FROM$; COMPANY$
```

```
PRINT
PRINT USING ITEM$;PARTNO1%, QUANT1%,PRICE1,CHARGE1
PRINT USING ITEM$;PARTNO2%, QUANT2%,PRICE2,CHARGE2
PRINT
PRINT USING "!.!.!. OWES A TOTAL OF **###,###.##";\
        FIRST.NAME$,MI$,LAST.NAME$,TOTAL

STOP
```

String-Field Specifications in "PRINT USING" Statements

	FIELD	FORMAT	EXAMPLE	COMPONENT
LITERAL CHARACTER	A single literal character	\x where x can be any character.	"\#" prints # "\#\#\!" prints ##! "\\\\" prints \\	\ is generally used to print a control character as a literal.
	Literal text and field separators	Any character not part of a string or numeric field format.	"#### ####. Three blank spaces separate two numeric fields "NO:###". The three characters NO: precede a numeric field.	Most frequently, literal blank spaces are used to separate fields.
NUMERIC FIELDS	Simple integer numeric	Two or more # characters, one # character per field width.	"###" specifies a 3-digit integer numeric field. "######" specifies a 6-digit integer numeric field.	Numbers are right shifted within the numeric field with blanks preceding the most significant digit.
	Real numeric with decimal point	Place . character within # string at desired decimal point location.	"###.##" specifies a 5-digit numeric field. 3 digits precede the decimal point, 2 follow.	Numbers are rounded.
	Numeric fields with , characters	Place one or more , characters anywhere between the first and last # characters.	"##,,#####,##" specifies a number with 7 predecimal and 2 postdecimal digits. A , occurs before every third digit to the left of the decimal point.	The position of , characters in the field specification is not relevant.
	Numeric field with exponential notation	Append one or more ↑ characters at the end of the field definition.	"#.####↑" specifies a number printed in exponential format with one predecimal digit and four postdecimal digits.	The decimal point position effects the exponent value. Four character positions are always added for the exponent.
	Numeric field with * in leading numeric character position	Add ** to the front of the field specification.	"**######.##" specifies a numeric field with 6 predecimal and 2 postdecimal digits, plus * characters in leading unused character positions.	This is used in financial printouts.
NUMERIC FIELDS	Numeric field with $ character	Add $$ to the front of the field specification.	"$$######.##" specifies a numeric field with 6 predecimal and 2 postdecimal digits, and a $ character preceding the first nonblank digit.	$ is not printed for negative numbers. $$ cannot be used with **.
	Numeric field with - sign in first character position Numeric field with - sign in last character position	Begin field specification with - character. End field specification with - character.	"-######.##" specifies a numeric field with 6 predecimal and 2 postdecimal digits, and a - sign in the first character position for negative numbers. "#####.##-" specifies a - sign in the last character position for the same field.	Normally the - sign precedes the first non-blank character. The leading - cannot be used with ** or $$ options.
	Variable-length string field	A single & character.	"&" prints a string field of any length.	
	Fixed-length string field	/ character in first and last character position.	"/...../" specifies a 7-character string field.	Any characters may appear between the first and last /.
	Single-string field character	A single ! character.	"!" prints the first character of a string field.	

From *CBASIC™ User Guide* by Adam Osborne, Gordon Eubanks, and Martin McNiff. Copyright 1981 McGraw-Hill Inc. Used with permission of OSBORNE/McGraw-Hill.

PRINT USING# *—outputs formatted data to disk file*

The PRINT USING# statement outputs formatted data to disk files. See the PRINT# and PRINT USING statements for details on formatted printing and printing to disk files. (See table on previous page.)

RANDOMIZE*—seeds a random-number generator*

The RANDOMIZE statement is used to seed the random-number generator.

Format:

RANDOMIZE

The time an operator takes to respond to an INPUT statement is used to seed the random-number generator. Thus, an INPUT statement must be executed before the RANDOMIZE statement.

Example:

RANDOMIZE

Sample Program:

```
REM GUESSING GAME
    INPUT "GUESS A NUMBER BETWEEN 0
      AND 100"; GUESS%
    RANDOMIZE
    NUMBER% = 100 * RND
    COUNT% = 1
    WHILE GUESS% <> NUMBER%

    IF GUESS%>NUMBER% THEN PRINT "TOO LARGE"\
    ELSE PRINT "TOO SMALL"

    INPUT "NEXT GUESS?"; GUESS%
    COUNT% = COUNT% + 1

WEND

PRINT "YOU GUESSED IT IN";COUNT%; "TRIES"
INPUT "DO YOU WANT TO TRY AGAIN (Y/N)?"; AGAIN$
IF AGAIN$ = "Y" THEN GOTO 100
STOP
```

READ—*assigns values from DATA statements to variables*

The READ statement assigns values taken from DATA statements to variables in the READ statement parameter list. (See the DATA and RESTORE statements.)

Format:

READ variable {,variable}

The next unused item from the list of constants compiled by concatenating all data-statement parameters is assigned to the next variable in the READ statement parameter list. (See the DATA statement for a sample program.)

Examples:

READ COST1,COST2,COST3
READ TABLE (I%)

READ#—*reads data from disk files*

The READ# statement is used to read data from disk files.

Format:

READ# integer exp [,integer exp];
variable {,variable}

The first expression is the file number for which data will be read; this file number must be active and range from 1 to 20. The second integer expression is optional and identifies the record number of a random-access file from which data will be read. If the second integer expression is present, the file must have been opened with the RECL parameter specifying record length.

When an attempt is made to read past the end of a file, execution continues with the statement number specified by the most recently executed IF END# statement having the same file number. If no IF END statement was executed, an error occurs. (See the OPEN statement for a sample program.)

READ# LINE—*reads a disk file string and assigns it to a variable*

The READ# LINE statement reads a record from a selected diskette file and assigns it to a string variable. The READ# LINE statement acts like the READ# statement.

Format:

READ# integer exp [,integer exp];
LINE string variable

The first integer expression selects the file. The second integer expression, if present, selects the record for a random-access file. These expressions are used as described for the READ# statement.

The string read from the selected record is assigned to the string variable. If the length of the record is greater than 255 bytes, the record is shortened to 255 characters and a warning is printed on the console.

REM—*indicates a remark*

The REM statement is used to document programs. This statement is ignored by the CBASIC compiler.

Format:

REM any character

CBASIC object size is not affected by REM statements, since these statements are ignored by the compiler. REM statements can be split across lines using a \ character. If a REM statement shares a line with other statements, then REM must be the last statement on the line.

RESTORE—*resets the data-list pointer*

The RESTORE statement is used to reset the pointer into the DATA statement's list of constants so that the next value read by a READ statement will be the first item in the first DATA statement.

Format:

RESTORE

Sample Program:

```
REM READ AND PRINT THE DATA LIST
DATA 123.987,42 "MORE DATA", "EVEN MORE DATA"

READ REAL.NUM, INT.NUM%, STRING1$, STRING2$
PRINT REAL.NUM; STRING1$, INT.NUM%; STRING2$

RESTORE
READ REAL.AGAIN, INT.AGAIN%
PRINT REAL.AGAIN, INT.AGAIN%
STOP
```

RETURN—*returns program execution from a subroutine to the calling program*

The RETURN statement causes execution of the program to continue with the statement following the most recently executed GOSUB. The RETURN statement also causes an exit from a user-defined function.

Format:

RETURN

Each GOSUB statement saves the location of the statement following it. Program execution returns to the saved location after the subroutine has completed execution. (See the GOSUB, ON GOSUB, and DEF statements.)

SAVEMEM—*reserves space for a file*

The SAVEMEM statement loads an assembly-language program from diskette into memory.

Format:

SAVEMEM integer constant, string expression

The integer constant specifies the number of bytes of memory to reserve for the file being loaded. Space is reserved at the top of available memory. You can calculate the beginning address of the reserved area by subtracting the integer constant from the largest available address.

The string expression identifies the file to be read. Records are read from the file until an end of file is encountered or until the next record to be read would exceed the specified memory size. If the string expression is null or if the integer constant is less than 128, space is reserved, but no file is loaded.

Only one SAVEMEM statement may appear in a program; if the first program executed contains a SAVEMEM statement, any chained program associated with it must also include a SAVEMEM statement. The associated chained programs must have the same integer constant but a different file may be loaded by each chained program.

Examples:

```
SAVEMEM 1028, "IOPACK.COM"
SAVEMEM 128,""
```

STOP—*stops program execution*

The STOP statement causes program execution to stop. Control is returned to the operating system (CP/M).

Format:

```
STOP
```

NOTE

Any OPEN files will be closed.

WEND—*terminates a WHILE loop*

The WEND statement terminates a WHILE statement loop (see the WHILE statement).

Format:

WEND

WHILE—*initiates the WHILE-WEND loop*

The WHILE statement controls looping between the WHILE statement and its corresponding WEND statement.

Format:

WHILE integer expression

Looping continues until the integer expression is a 0 (false). The loop may contain any number of statements, including additional WHILE statements. If the expression is false initially, then no statements in the loop execute. (See RANDOMIZE for a sample program.)

Examples:

```
WHILE AMOUNT < = MAX
WHILE -1 REM LOOP FOREVER
```

CBASIC Functions

There are three kinds of functions: numeric, string, and disk. The function name further identifies it as an integer, a real, or a string function. If the function requires an expression, then the normal CBASIC expression rules apply. If an integer expression is required, real values are automatically converted to integers. Likewise, integer expressions are converted to real expressions where necessary. Numeric expressions in string functions generate a run-time error; so do string expressions in numeric functions. Avoid conversions by using the proper form for the expression since this improves program execution speed.

ABS—*returns absolute value*

The ABS function returns the absolute value of the argument.

Format:

ABS (numeric expression)

Examples:

```
X=ABS(Y)
DIFF=ABS(COST-PROFIT)
```

ASC—*returns an ASCII value*

The ASC function returns the ASCII integer value for the first character of the argument. Only the first character is considered. If the expression is evaluated as a nul! string or if the argument is numeric, an error will occur.

Format:

ASC (string expression)

Examples:

I% = ASC(STRING$)

FIRST% = ASC(FIRST.NAME$)

ATN—*returns the arctangent of the argument*

The ATN function is used to return the arctangent of an argument. The argument must be expressed in radians. The value returned is a real number. The arctangent returned can be used to compute various inverse trigonometric functions.

Format:

ATN (numeric expression)

Examples:

ANGLE = ATN(X)
ASIN = ATN(X/SQR(1.0 - X*X))

CHR$—*returns the ASCII string equivalent of the argument*

The CHR$ function converts the argument to its one-character ASCII string equivalent. The CHR$ function can be used to send control characters to an output device. If the argument is greater than 255, the high-order byte is ignored.

Format:

CHR$ (numeric expression)

Examples:

```
BELL$ = CHR$(7)
STOP.CHAR$ = CHR$(STOP%)
```

COMMAND$—*returns the command line*

The COMMAND$ function returns the command line used to execute the current program. The name of the program being executed will not be included in the string that is returned. If the TRACE option is used with CRUN, the word TRACE and the associated line numbers will not be included.

Format:

COMMAND$

The COMMAND$ function allows options to be passed to the CBASIC program when it is executed. The COMMAND$ function may be used anywhere and anytime within a program, including programs loaded by a CHAIN statement.

CONCHAR%—*returns the binary equivalent of a character input at the console device*

The CONCHAR% function reads one character from the console input device and returns an integer equal to the binary representation of that character.

Format:

CONCHAR%

The character that is read back is echoed to the console display device by the CONCHAR% function. If no character has been entered at the console, a zero is returned. (See the CONSTAT% function.)

Examples:

ANS% = CONCHAR%
IF CONCHAR% = ESC% THEN DONE% = TRUE%

CONSTAT%—*returns console status*

The CONSTAT% function returns an integer expression to indicate console status. A logical true (−1) signifies that the console is ready. If a logical false (0) is returned, the console device does not have a character ready.

Format:

CONSTAT%

Examples:

IF CONSTAT% THEN ANS% = CONCHAR%
X% = CONSTAT%

COS—*returns the cosine of the argument*

The COS function returns the cosine of the argument. The argument must be expressed in radians, and the value returned is a real number.

Format:

COS (numeric expression)

Example:

```
X = COS(ANGLE)
```

EXP—*returns the exponent of the argument*

The EXP function returns the value of the constant raised to the power of the argument. The value returned is a real number.

Format:

EXP (numeric expression)

Examples:

```
POWER=EXP(X*X-Y*Y)
E=EXP(1)
```

FLOAT—*converts the argument to a real number*

The FLOAT function converts the argument to a real number. If the argument is already a real number, FLOAT converts it to an integer and then back to a real number.

Format:

FLOAT (numeric expression)

Example:

```
X=SIN(FLOAT(I%))
```

FRE—*returns the amount of available data memory*

The FRE function returns the number of available memory bytes in the dynamic or free storage area. The value FRE returns is a real number. Free storage may consist of two or more noncontiguous memory blocks.

Format:

FRE

Example:

SPACE.REMAINING=FRE

INP— *returns a byte from an I/O port*

The INP function returns a byte from a selected input/output port. An integer that is the value read from the port addressed by the argument is returned.

Format:

INP (integer expression)

Examples:

```
DEV.1%=INP(23)
TAPE.STATUS%=INP (TAPE.SP%)
```

INT— *returns the integer portion of the argument*

The INT function truncates the fractional portion of the argument and returns the integer portion. The value returned by INT is a floating-point number; if the argument is an integer, it is converted to a real value.

Format:

INT(numeric expression)

Examples:

```
DOLLARS=INT(TOTAL.DUE)
CENTS=TOTAL-INT(TOTAL.DUE)
```

INT%—*converts the argument to an integer*

The INT% function converts the argument to an integer. The difference between INT and INT% is that the result of INT is a real number, while the result of INT% is an integer. If the argument is an integer, it is converted to a real number and then back to an integer.

Format:

```
INT% (numeric expression)
```

Examples:

```
K%=INT%(SIZE)
LENGTH=DIMENSION (INT%(X))
```

LEFT$—*returns leftmost characters of the argument*

The LEFT$ function returns characters from the left of a string argument.

Format:

```
LEFT$ (string expression, numeric expression)
```

The numeric expression specifies the number of characters returned. If the number of characters that is to be returned is greater than the length of the string, the entire string is returned. If the numeric expression is zero, a null string is returned. If the expression is negative, an error will occur.

Examples:

```
RESPONSE=LEFT$(ANS$,1)
PRINT LEFT$
(NAMES$,MAX.NAME.LENGTH%)
```

LEN—*returns the length of an argument*

The LEN function returns the length of a string argument. The value returned by LEN is an integer, a zero is returned if the value is a null string. An error occurs if the argument is numeric.

Format:

```
LEN (string expression)
```

Examples:

```
LENGTH.NAME%=LEN(NAME$)
IF LEN(TEMP$) >= MAX.L% THEN GOTO 100.00
```

LOG—*returns the natural logarithm of the argument*

The LOG function returns the natural or Naperian logarithm of the argument. The LOG function can be used to calculate logarithms to other bases. The value returned is real. Integer arguments are converted to real numbers before the logarithm is computed. An error occurs if the argument is negative or zero.

Format:

```
LOG (numeric expression)
```

Examples:

```
BASE.TEN.L=LOG(X)/LOG(10)
Z=LOG(W)
```

MATCH—*searchs a string for a match*

The MATCH function returns the position of the first occurrence of a pattern string within a source string, beginning at a specified position in the source string.

Format:

MATCH (string exp, string exp, numeric exp)

The first string expression argument is the pattern, the second expression argument is the source string. The integer argument specifies the character position in the source string where the search is to begin. The search progresses from the starting position to the end of the source string, attempting to match the pattern specified in the first expression. If a match occurs, the position of the first matching character in the source string is returned. If no match occurs, a zero is returned.

The following special pattern-matching characters are provided:

# A pound sign matches any numeric digit (0–9).

! An exclamation mark matches any uppercase or lowercase letter (A–Z and a–z).

? A question mark matches any character.

\ A backslash indicates that the character following the backslash does not have a special meaning. Thus, a backslash before a #,!,?, or another backslash character will override the definition above and result in the character being treated as a normal pattern character.

If either string expression is null, a zero is returned. If the beginning point for the match is greater than the length of the source string, a zero is returned. If the numeric expression is zero or negative, an error will occur.

Examples:

MATCH (“CDE”, “ABCDEFGHI”, 1) returns 3
MATCH (“CDE”, “ABCDEFGHI”, 3) returns 3
MATCH (“CDE”, “ABCDEFGHI”, 4) returns 0
MATCH (“D?F”, “ABCDEFGHI”, 1) returns 1
MATCH (“\ #1 \ \ \ ?”, “1#1\ ?2#”, 1) returns 2

MID$—*returns a portion of a string*

The MID$ function returns a string that may be any portion of another string. MID$ can accomplish the same function as either RIGHT$ or LEFT$, but, in addition, MID$ can return a string from the middle of another string.

Format:

MID$ (string exp, numeric exp, numeric exp)

A portion of the first string expression is returned. The second expression specifies the starting position of the string to be returned. The third expression specifies the length of the string to be returned. If the third expression specifies a character position beyond the end of the string, then characters from the position specified by the second argument, up to the end of the string, are returned. A null string is returned if the starting position specified by the second expression is greater than the length of the string, or if the third expression is zero.

NOTE

An error occurs if either numeric expression is negative, or if the second one is zero.

Examples:

```
MIDDLE$=MID$(NAME$,START.MN%, LENGTH.MN%)
DAY$=MID$("MOTUWETHFRSASU", DAY% 3-2,3)
```

PEEK—*returns the contents of a selected memory location*

The PEEK function returns the contents of the memory location addressed by the function argument.

Format:

PEEK (numeric expression)

The numeric expression is a memory address. The PEEK function returns an integer value equal to the contents of the addressed memory location. The memory location must be within the address space of the computer, or the results will be meaningless. For memory locations greater than 32767, the address must be negative; in this case the address could be expressed in hexadecimal notation for clarity.

Examples:

```
BDOS%=PEEK(6) + PEEK(7) 256
ENTRY%=PEEK(OEOOH)
PARM1%=PEEK(LOC.P1%)
```

POS—*returns the column position of the next print character*

The POS function returns the character number for the next character to be displayed or printed on the current line.

Format:

POS

POS returns the number of characters to the output device. Cursor control characters are also counted even when the cursor has not been advanced. Thus, if the cursor has been positioned by special characters or nonprinting control characters have been output, the POS function does not return the actual cursor position.

Examples:

```
PRINT TAB (POS+3);"#"
LOC.CURSOR%=POS
```

RENAME—*changes the name of a disk file*

The RENAME function changes the name of a disk file.

Format:

**RENAME (string expression,
string expression)**

The first string expression is the new file name. The second string expression is the current file name. The RENAME function returns an integer value of 0 (false) if the RENAME cannot be accomplished and a −1 (true) if the RENAME is successful.

NOTE

The file being renamed must not be open or an error will occur when the file is closed.

Examples:

```
IF RENAME ("MASTER.CUR", "MASTER.TMP")=O THEN GOTO 500
X%=RENAME(NEW.NAME$, OLD.NAME$)
```

RIGHT$—*returns the rightmost character of the argument*

The RIGHT$ function returns the rightmost characters of a string.

Format:

**RIGHT$ (string expression,
numeric expression)**

The numeric expression specifies the number of rightmost characters of the string to be returned. If the number of characters to be returned is greater than the length of the string expression, then the entire first argument is returned. If the numeric expression is 0, a null string is returned. A negative numeric expression causes an error to occur.

Examples:

```
CHECK.DIGIT$=RIGHT$(ACCOUNT.NOS$,1)
LAST.NAME=RIGHT$(NAME$,LEN(NAME$)-LEN(FIRST.NAME$)
```

RND—*returns a random number*

The RND function returns a random number between 0 and 1.

Format:

RND

The RND function generates the next random number in a sequence that is based on the current seed. The value returned is a real number. The RANDOMIZE statement must be executed to generate a seed and a random-number sequence.

Example:

PRINT RND

SADD—*returns the starting memory address of a string*

The SADD function returns the starting memory address of the string currently assigned to the string variable. The first byte of the string is the length of the string. Subsequent bytes hold characters of the string.

Format:

SADD (string variable)

The string variable cannot be a string expression. The value returned by SADD is an integer. A zero is returned for a null string unless the null strings have previously been assigned a test value.

Examples:

```
LOC.PARM%=SADD(NAME$)
PTR%+SADD(A$)
```

SGN—*returns the sign of the argument*

The SGN function returns −1, 0, or 1, depending on whether the argument is negative, zero, or positive respectively.

Format:

SGN (numeric expression)

NOTE

The value returned is an integer.

Examples:

```
IF SGN(TOTAL)=−1 THEN GOTO 200.20
ON SGN(X) + 2 GOTO 10, 20, 30
```

SIN—*returns the sine of the argument*

The SIN function returns the sine of the argument.

Format:

SIN (numeric expression)

The argument must be expressed in radians. The value returned is a real number.

Example:

```
X=SIN(ANGLE)
```

SIZE—*returns the amount of reserved space*

The SIZE function returns the amount of space reserved by a file or a group of files.

Format:

SIZE (string expression)

The string expression may be an ambiguous file name or the name of one file within the directory. Ambiguous file names allow the programmer to determine the space being used by all of the files whose names have common characters. If no files within the directory match the expression, a 0 is returned. The size function returns an integer; the value returned is the number of blocks that the file is using. Each block is 128 bytes.

Examples:

```
AMOUNT$+SIZE(WORKFILE)
I%=SIZE("*.BAK")
FREE.SPACE%=DISK.CAPACITY%- SIZE("*.*")
```

SQR—*returns the square root of the argument*

The SQR function returns the square root of the argument.

Format:

SQR (real expression)

The SQR function returns a real number. If the argument is negative, a warning appears on the console, and the absolute value of the argument is used in calculating the square root.

Examples:

```
HYP=SQR(X*X + Y*Y)
PRINT SQR(X)
```

STR$—*returns a character string*

The STR$ function returns a text character that is the ASCII equivalent of the argument.

Format:

STR$ (real expression)

The STR$ function converts a real number into its string equivalent. For example, the real number 1.234 would convert to the string "1.234".

Examples:

```
A$=STR$(X)
PRINT STR$ (ZIP)
```

TAB—*positions the cursor*

The TAB function is used to position the cursor at the character position the argument specifies.

Format:

TAB (integer expression)

The TAB function can be used only in PRINT statements. The cursor or printer, depending on whether LPRINTER is in effect, moves to the right of the selected column. The TAB does not move backwards. If the expression is less than the current cursor position, a new line starts and the TAB then occurs.

If the argument exceeds the line width of the device receiving output, an error occurs. A semicolon should be used following a TAB; a comma could cause additional spacing.

Examples:

```
PRINT TAB(START.COL%); NAME$
PRINT X;TAB(30);Y
```

NOTE

The TAB function counts characters output since the last carriage return and line feed. This could differ from the current cursor line position. Differences result if the program has output non-printing control characters or characters that move the cursor.

UCASE$—*converts a string to uppercase characters*

The UCASE$ function is used to convert a string to its uppercase equivalent.

Format:

UCASE$ (string expression)

The value UCASE$ returns is a string. All lowercase alphabetic characters in the argument are converted to uppercase characters. Other characters remain unaltered.

Examples:

```
U.CITY$=UCASE$(CITY$)
X$=UCASE$(X$)
```

VAL—*converts a string to a real number*

The VAL function converts a string to a real number.

Format:

```
VAL (string expression)
```

The VAL function converts a string to its real numeric equivalent. The string must be a valid text representation of a real number. For example, the string "1.234" is converted into the real number 1.234. The first character of the string must be a number or a plus or minus sign. An error is generated if the string includes any character that is not part of a valid real number. Zero is returned for a null string.

Examples:

```
X=VAL(A$)
PI=VAL("3.1416")
```

Compiler Directives

Compiler directives control the compiler (CBAS2.COM). Except for the END statement, all directives are preceded by a percent sign in the first column. The compiler ignores any nondirective characters on the same line.

%LIST Begins listing a source file when it is encountered during compilation.

%NOLIST Terminates listing of the source file initiated by %LIST.

%PAGE <constant> Sets page length output to the printer. The default page length is 64.

%EJECT Positions listing sent to the printer and disk at the top of the following page.

%INCLUDE <file name> Causes named file to be compiled into the source file. %INCLUDE may be preceded by a drive identifier if needed. The file referenced by %INCLUDE must be of type .BAS. Each statement number in the referenced file is indicated by an equal sign after its assigned statement number. %INCLUDE(s) may be nested up to six levels.

%CHAIN <constant>, <constant>, <constant> Sets the size of the main programs—constant, code, data, and variable area—to prevent overwrites. The first constant is the area used for real constants, the second is the size of the code area, the third is used to store value from DATA statements, and the fourth is the size of the area used to store variables.

[statement number >] END The END statement is optionally used to terminate reading of the source file. This statement must be the first on a line to cause all statements following it to be ignored.

Compiler Toggles

You set compiler toggle switches by following the program file name with a blank, a dollar sign, and the desired letters as follows:

B Suppresses program listing to the console during compilation. (Default is OFF.)

C Suppresses generation of an .INT file. Saves time when compiling for errors. (Default is OFF.)

D. Suppresses translation of lowercase to uppercase letters. (Default is OFF.)

E Lists line numbers where errors have occurred. Toggle E must be ON when you use TRACE. (DEFAULT IS OFF.)

F Sends the compiled output listing to the list device as well as the console. (Default is OFF.)

G Sends a compiled output listing with the same name as the source, but of type .LST, to a disk file. A drive identifier may be appended in parentheses after the toggle code. (Default is OFF.)

Examples:

```
B:CBAS2 A:ACCOUNTS $BGF
CBAS2 PAYROLL $(G)EC
```

Cross-Reference Lister (XREF.COM)

The utility program XREF.COM creates a disk file that lists alphabetically every identifier used in a CBASIC program. The list indicates the usage of the identifiers (function, parameter, or global) and lists each line on which the identifier was used. Functions appear first in the list, with associated parameters and local variables immediately following.

Format:

```
XREF < file name > [disk ref] [$<toggles>] ['<title>']
```

The named source file must be of type .BAS. By default, the listing is sent to the disk on which the source file is located. A drive identifier after the file name sends the cross-reference file to the drive specified.

A file created by XREF has the same name as the source, but adds the extension .XRF. The standard output is 132 columns wide. You can use toggles in the XREF command to direct output of the cross-reference listing. A blank space, a dollar sign, and the following toggle letter codes direct output of the XREF file:

A Outputs the cross-reference file to the list device and disk file.

B Suppresses output to the disk. No output is produced if you specify only B.

C Suppresses output to the disk but permits output to the list device. This toggle has the same effect as specifying A and B.

D Produces 80-column output instead of 132.

E Produces output of identifiers and their usage without line numbers.

F(*n*) Lets you change the default page length (60 lines) to *n*.

G Suppresses form-feeds and printing of heading lines.

H Suppresses translation of lowercase to uppercase letters.

The optional 'TITLE' field must be the last on the command line. The characters indicated between the apostrophes are printed as the heading for each page. You can specify up to 30 characters when the output width is set to 132, and 20 if the column width is set to 80.

TRACE

The TRACE option allows run-time debugging of a program by printing the compiler-assigned line number of each statement as it is executed.

Format:

```
CRUN2 <filename> [TRACE] <Ln1>[,<Ln2>]]]
```

You must have used the E toggle in the source program in order for TRACE to work. The first number specifies the line number where tracing should begin and the second number specifies where tracing should end. You can use the first number alone to begin tracing at any desired line. If you don't specify a line number, the entire program is traced.

Example:

```
CRUN2 EXAMPLE TRACE 1,3
```

would cause the following output:

```
AT LINE 0001
AT LINE 0002
AT LINE 0003
```

MBASIC

Special Control Characters

^A	enters EDIT mode on current line or previously typed line.
^C	interrupt program execution; returns to BASIC command level.
^G	rings bell.
^H	deletes last character entered.
^I	advances to next tab stop (every 8 columns).
^J	divides logical line into physical lines.
^O	halts/resumes program output.
^R	retypes current line.
^S	suspends program execution.
^Q	resumes execution following **^S**.
^U	deletes current line.
^X	deletes current line.
<RETURN>	ends every program line.
<ESC>	leaves EDIT mode subcommands.
.	defines current line for EDIT, RENUM, DELETE, LIST, LLIST.
&O or **&**	serves as a prefix for octal constant.
&H	serves as a prefix for hexadecimal constant.
:	separates statements entered on the same line.
?	serves as a PRINT statement equivalent.

MBASIC Commands

AUTO — *automatically generates line numbers*

The AUTO command sequentially numbers program lines following each carriage return.

Format:

AUTO [line number [, increment]]

Line numbering begins with the number following AUTO and is incremented as specified by the next number; the default value for both is 10. The increment value specified by the most recent AUTO command is adopted when you use a comma but no increment value.

An asterisk appears when AUTO generates an existing line number indicating that the contents of the existing line will be replaced. A carriage return typed after the asterisk skips the existing line and generates the next line number. **^C** terminates AUTO without saving the current line and returns to the command level.

Examples:

AUTO 15,5	Generates line numbers 15,20,25,30,35. . .
AUTO	Generates line numbers 10,20,30,40,50. . .

CLEAR—*reallocates memory space*

The CLEAR command is used to set numeric variables to zero, string variables to null, and, optionally, to set the end of memory and amount of stack space.

Format:

CLEAR [,[<expression>] [, <expression>]]

The first expression, if present, sets the highest memory location available for use. The second expression reserves stack space. The default stack space setting is either 256 bytes or an eighth of the available memory, depending on which is smaller.

Examples:

CLEAR ,32554	Sets highest memory to 32554
CLEAR ,,2999	Sets stack size to 2999
CLEAR ,32554,2999	Sets highest memory to 32554 and stack size to 2999

CONT — *resumes program execution*

The CONT command causes execution of the current program to continue following **^C**, STOP, or END statement.

Format:

CONT

Execution continues where it left off when a CONT command is issued. If execution halted following a prompt for an INPUT statement, the prompt is redisplayed. CONT is usually used with STOP to examine and change values or to resume execution following an error.

Example:

(See STOP statement.)

DELETE — *deletes program lines*

The DELETE command erases program lines between the first and second line numbers listed.

Format:

DELETE [<first line number>]
[<second line number>]

You can delete program lines from the beginning of the file to any desired line number by following DELETE with a hyphen and the last line to be eliminated. The command level always returns following execution of a DELETE command. If a nonexistent line number is referenced, an "Illegal function call" error occurs.

Examples:

```
DELETE 10
DELETE 10–50
DELETE –100
```

EDIT—*enters edit mode*

The EDIT command lets you edit a specified program line. Once in the edit mode you use subcommands to move the cursor, insert, delete, replace, or search for data.

Format:

EDIT <line number>

When you issue the EDIT command, the specified line number is listed followed by a space. Alternatively, **^A** can be used to enter the edit mode for the line currently being written. When **^A** is issued, the edit mode is indicated by a carriage return and an exclamation mark. If a syntax error occurs during program execution, the edit mode is automatically entered.

A number may precede many of these subcommands, indicating that the command should execute that many times; (*n*) represents any number. The editing subcommands are summarized below:

(*n*) SPACE moves the cursor right (*n*) number of characters.

(*n*) ← or **^H** moves cursor left (*n*) number of characters (destructive in insert mode).

(*n*) D deletes (*n*) number of characters to the right of the cursor.

I (string) inserts the specified (string) at the cursor position.

H (string) deletes all characters right of the cursor and inserts the specified (string).

X (string) moves the cursor to the end of the line and inserts the specified (string).

(*n*) S (target) finds the (*n*)th occurrence of the specified (target) character.

(*n*) K (target) deletes all characters to the (*n*)th occurrence of the specified (target) character.

(*n*) C (list) changes the next (*n*) characters to the specified (list) of characters.

A restores original line for reediting.

L lists the remainder of the line and repositions the cursor at the beginning.

Q abandons the edit mode and returns to the command level without saving changes.

RETURN saves the newly edited line and exits from the edit mode.

E saves the newly edited line without displaying the remainder.

NOTE

If you enter an unrecognized edit subcommand or parameter, the bell will ring.

FILES—*lists files on the currently active drive*

The FILES command is used like the CP/M DIR command to view a directory of the files residing on the current disk.

Format:

```
FILES [<filename>]
```

You can list all files or a specific file on the active drive, or you can view a category of files. The standard CP/M wildcard conventions can be used.

Examples:

```
FILES
Files *.BAS
Files MBASIC.BA?
```

LIST—*lists program lines*

The LIST command causes specified program lines to be listed on the screen.

Format:

```
LIST [<line number>[ -[<line number>]]]
```

Typing the LIST command while you are in the command mode provides a listing of all program lines in memory. A single line number or a range of lines between a first and second line number may be listed. A hyphen used in place of either the first or the second line number lists all program lines beginning or ending at the indicated line number.

Examples:

```
LIST
LIST 10
LIST 10–100
LIST –50
LIST 50–
```

LLIST—*prints program lines*

The LLIST command causes specified program lines to be listed at the printer instead of at the console.

Format:

```
LLIST [<line number>[-[<line number>]]]
```

The LLIST command functions exactly as the previously described LIST command, except that the printer serves as the output device.

LOAD—*loads a program file*

The LOAD command causes a named disk file to be loaded in memory.

Format:

```
LOAD <file name> [,R]
```

The file name is the name under which the file was saved and whose default extension, .BAS, is supplied by CP/M. When LOAD is issued, all files are closed and memory is cleared before the specified file is loaded. The R option leaves data files open and automatically runs the newly loaded program. You use the R option to chain programs that share common data.

Example:

LOAD "NEWFYL", R Assumes file type of .BAS

MERGE—*combines two programs*

The MERGE command references and combines a disk file with the program in memory.

Format:

MERGE <filename>

The file name is the name you assigned to the file when you saved it. Only files saved in ASCII format may be merged, or else a "Bad file mode" error will occur. CP/M automatically furnishes a .BAS-type extension. Lines in the file being merged replace those in the calling file if line numbers are duplicated. The command mode returns following a MERGE command.

Example:

MERGE "DATAFYL"

NAME—*renames a file*

The NAME command renames a disk file.

Format:

```
NAME <old filename> AS <new filename>
```

You list the name of the file to be renamed first, followed by AS and the new name that will identify the file. If the old file name does not exist, or if the new file name is already being used, an error occurs.

Example:

```
NAME "OLDFYL" AS "NEWFYL"
```

NEW—*deletes current program from memory*

The NEW command deletes all variables and clears the current program from memory.

Format:

```
NEW
```

Issue NEW to clear memory and prepare for a new program to be loaded. This command is usually used on unwanted files or ones that have already been saved. The command mode returns following the NEW command.

Example:

```
NEW
```

NULL—*sets the number of nulls to follow each program line*

The NULL command establishes the number of nulls that are to follow each line in the program.

Format:

NULL <integer expression>

The integer expression signifies the number of nulls output after each line. The default value is zero. You should use 0 or 1 for most 1200-baud or parallel printers, 2 or 3 for 30-cps hard copy printers, and 3 or larger for 10-character-per-second tape punches or typewriter printers.

Examples:

NULL 2

RENUM—*renumbers program lines*

The RENUM command sequentially renumbers program lines.

Format:

RENUM [[<new number>] [,[<old number>] [,<increment.]]]

The new line number replaces the old line number, which is incremented as specified. By default, line numbering begins with the first line number of the program, starting with line 10 and incremented by 10. The newly specified line numbering is also supplied for all existing statements that reference lines by their old numbers. Line numbers cannot be renumbered out of sequence or extend beyond 65529 or an "Illegal function call" will occur.

Examples:

```
RENUM
RENUM 300,,50
RENUM 1000,900, 20
```

RESET—*closes files and updates the directory*

The RESET command closes all open disk files and updates the directory information.

Format:

RESET

You should normally issue RESET before removing a diskette to ensure that all files have been closed and to maintain an accurate directory.

RUN—*executes a program*

The RUN command executes either the program currently in memory or a program stored in a disk file.

Format:

RUN <line number>

The above format executes the program in memory. Optionally, the program may be executed from a specific line number specified after the command.

Format:

```
RUN <file name> [,R]
```

This form of the RUN command executes a particular program stored in a disk file. When the file is loaded, all other files are closed and everything residing in memory clears. The R option can be used to leave data files in memory in an open condition.

Examples:

```
RUN 100
RUN"ANYFILE",R
```

SAVE—*saves named file*

The SAVE command saves the contents of memory to a named disk file.

Format:

```
SAVE"<file name>"[,A] [,P]
```

You can save the contents of memory under any valid file name. CP/M automatically supplies a default-type extension of .BAS if you don't supply it. If the name you assign to the file already exists, the contents of the existing file are written over.

Files are saved in binary format; however, you can save the file in ASCII format by placing a comma and the letter A after the file name. You can protect the file from unauthorized access by appending P to the end of the statement. The P option prevents lines in the program from being subsequently listed or edited.

Examples:

SAVE "COMFYL", A	Saves ASCII
SAVE "PROGRAM", P	Saves protected
SAVE "BINARY"	Saves binary

SYSTEM—*closes files, then relinquishes control to CP/M*

The SYSTEM command causes all open files to be closed, updates the directory, and returns to the CP/M command level. Note: **^C** does not exit from MBASIC-80 to CP/M.

Format:

SYSTEM

TRON/TROFF—*enables/disables tracing*

The TRON command is used to turn tracing ON. When tracing is ON, the line number currently being executed appears within brackets. Execution of a TROFF or NEW command turns tracing OFF.

Formats:

TRON

TROFF

Example:

```
TRON
AUTO 10,10
10 A=5
20 PRINT A
30 END
40 ^C
RUN
[10] [20] 5
[30]
TROFF
```

WIDTH—*sets line width*

The WIDTH command establishes the length of lines displayed on the screen or optionally output to the printer.

Format:

WIDTH [LPRINT] <integer expression>

This command affects the width of lines displayed at the console. The optional LPRINT clause sets the width of lines output to the printer. The integer expression specifies the number of characters, ranging between 15 and 255, that occupy a line; the default line width is 72. A line width of 255 is considered infinite, so carriage returns do not occur automatically at the end of lines. When you position the printhead or cursor using the POS or LPOS functions, it returns to zero after reaching position 255.

Examples:

```
WIDTH 52
WIDTH LPRINT 255
```

MBASIC Statements

CALL—*links to an assembly-language subroutine*

The CALL statement transfers program flow to an assembly-language subroutine.

Format:

CALL <variable name> [(<argument list>)]

The variable name contains the starting memory address for the subroutine, and it cannot be an array variable name. The argument list holds arguments passed to the subroutine. The argument list cannot contain literals.

Example:

```
10 ROUTINE=&HD000
20 CALL ROUTINE (I,J,K)
```

CHAIN—*passes variables between programs*

A CHAIN statement transfers control and then passes variables to a referenced program.

Format:

CHAIN [MERGE] <filename> [,[<line number>] [,ALL]
[,DELETE <range>]]

The file name identifies the program being referenced. The line number or expression evaluates to the line where execution of the called program begins. If you omit the line number or expression, execution begins at the first line.

If you use the ALL option, all variables are passed to the called program. You have to use a COMMON statement in conjunction with CHAIN when passing partial variables.

The MERGE option lets you bring in a subroutine as an overlay. A MERGE operation is performed with the current and the called programs. The called program must be an ASCII file if it is to be merged.

To remove an overlay so another overlay can be brought in, use the DELETE option. The line numbers specified in the range are modified accordingly by RENUM.

NOTE

The MERGE option leaves files open and preserves the current OPTION BASE setting. If MERGE is omitted, CHAIN does not preserve variable types or user-defined functions.

Examples:

```
CHAIN "PROG"
CHAIN "PROG", 30, ALL
CHAIN MERGE "PROG", 30
CHAIN MERGE "PROG",30, DELETE, 30–110
```

CLOSE—*closes disk files*

The CLOSE statement concludes input and output to disk files.

Format:

CLOSE [[#] <file number> [,[#] <filenumber...>]]

The same number under which a file was opened is used to close the file. The closed file's number is released and may subsequently be used to open another file. If you don't specify a file number, all files are closed.

A CLOSE statement issued for a sequential output file writes the final buffer of output.

NOTE

Files automatically close when the computer encounters an END statement or a NEW command.

Examples:

CLOSE #1

CLOSE

CLOSE FILE.1,FILE.2 Where FILE.1 and FILE.2 are valid variables that contain the appropriate file numbers.

COMMON—*specifies common variables*

The COMMON statement specifies simple and subscripted variables that are retained in a common area of memory and are passed between chained programs.

Format:

COMMON <list of variables>

Common statements should appear at the beginning of the program. You cannot use the same variable in more than one COMMON statement. To specify an array variable, append opened and closed parentheses "()" to the variable name.

NOTE

If you use the BASIC compiler, you must declare common arrays in preceding DIM statements.

Example:

```
10 COMMON A,B,C,D(),G$
20 CHAIN "PROGRAM.TYP", 10
```

DATA—*holds a data list within a program*

DATA statements define string and numeric constants, which are stored until you assign them to variables through READ statements.

Format:

DATA <list of constants>

You can use DATA statements anywhere in a program to list string, integer, or real constants. Any number of DATA statements are permissible, each occupying one line exclusively. You can list as many constants, delimited by commas, that can fit on a line. The constants from all of the DATA statements are stored in memory in the same sequential order in which they appear in the program. These constants are then assigned to variables within the READ statements being executed.

Constants in DATA statements and variables in READ statements that reference them, must be of the same type. The list of constants can contain any combination of numeric formats except numeric expressions. String constants that contain commas, colons or leading/trailing blanks must be surrounded by quotation marks. An attempt to read past the available data constants will cause an error.

NOTE

The RESTORE statement can be used to reset the data pointer, causing the data to be reread.

Example:

```
110 DATA 1, 2, 2.1, 22.1
111 DATA "Sample, with comma", Sample
without comma
```

DEF FN—*defines user-written function*

The DEF FN statement is used to define and name a function that the user writes.

Format:

```
DEF FN <name> [(<parameter list>)] =<function definition>
```

The function name must be a valid variable name preceded by the letters FN. The parameter list contains variable names that correspond to those used by an expression in the function definition. When the named function is referenced, the values in the parameter list are substituted for the variable names in the function definition. Each variable name in the parameter list must be set off from the next by a comma.

The function definition consists of an expression, that when referenced, performs the required operation. Variables in the expression are interpreted only in the context of the function. If the variable is not explicitly stated in the parameter list, the current value of the variable is assumed.

User-defined functions can be either string or numeric. When a type is specified in the function name, the value of the expression is forced to match it before returning to the calling file. If the type of the function name and the argument are different, then a "TYPE MISMATCH" error will occur. An error also occurs if the DEF FN is not executed prior to the function it defines.

Example:

```
100 DEF FNEF (X,Y)=X^2+Y^2
110 A=FNEF (3,4)     (A would equal 25 in
                     this example.)
```

DEF INT/SNG/DBL/STR—*defines variable types*

The DEF statement identifies variable names that begin with a predetermined letter as being either integer, single- or double-precision, or string-type variables.

Format:

DEF <type> <range(s) of letters>

The type is defined as INT (integer), SNG (single-precision), DBL (double-precision), or STR (string). The range of letters listed after the type declaration identify variable names that begin with the indicated letters as being a specific type of variable. However, type-declaration characters have precedence over these DEF statements in the typing of variables. Undefined variables without declaration characters are assumed to be single-precision variables.

Example:

```
100 DEFSTR A–F
110 DEFDBL G
120 DEFINT H–M, N–S
```

DEF USR—*defines assembly subroutine entry point*

The DEF USR statement defines the starting address of an assembly-language subroutine.

Format:

```
DEF USR [<digit>] =<starting address>
```

A digit from 0 to 9 corresponds to the number of the USR routine whose starting address follows; USR 0 is assumed if no digit is specified. You can use any number of DEF USR statements in a program to redefine subroutine starting addresses so as many subroutines as necessary can be accessed.

Example:

```
100 DEF USR9=2200
110 A=USR9 (X^2/2.14)
```

DIM—*allocates storage for an array*

The DIM statement allocates storage for an array and defines the upper limit of each subscript; a lower-bound limit of zero is assumed.

Format:

```
DIM <list of subscripts>
```

The subscript list indicates the number and extent of dimensions for the array being declared. The minimum value of a subscript is considered to be 0 unless an OPTION BASE statement designates otherwise. Subscripted variables that are not dimensioned have an upper-bound limit of 10. If a subscript larger than the value specified in the DIM statement is encountered, a "Subscript out of range" error occurs.

Example:

```
10 DIM X(20)
20 FORY=0 to 20
30 READ X (Y)
40 NEXTY
```

END— *terminates program execution*

The END statement halts program execution, closes all files, and returns to the command level.

Format:

```
END
```

You may place END anywhere in the program to terminate program execution. An END statement is not required at the end of a file.

Example:

```
10 IF A=20 THEN END ELSE GOTO 20
```

ERASE—*eliminates specified arrays*

The ERASE statement erases previously defined arrays. Following their erasure, the space occupied by the arrays is released, and you can use it to redimension the arrays.

Format:

ERASE <array variable list>

If you make an attempt to redimension an array without first erasing it, a "Redimensioned array" error will occur.

Example:

```
10 ERASE X,Y
20 DIM Y(50)
```

ERROR—*allows user-defined error codes or simulates error*

The ERROR statement allows error codes to be defined, or may be used to simulate the occurrence of a specified error.

Format:

ERROR <integer expression>

The value of the integer expression must be between 0 and 255. To define an error code, specify a value greater than any existing MBASIC error codes. An error-trapping routine can then handle the user-defined error code. Errors are simulated, with an integer expression corresponding to the desired MBASIC error code, which causes the associated error message to be printed. If the ERROR statement references a code for which no error message has been designed, an "Unprintable error" message appears. If no error-trap routine is associated with the error, then an error message appears and execution is terminated.

Examples:

```
10 ON ERROR GOTO 100
20 INPUT "TYPE A NUMBER FROM 1 TO 10"; N
30 IF N<1 or N>10 THEN ERROR 200
```

ERR and ERL—*serve as variables in error routine*

The ERR and ERL variables direct program flow in an error-handling subroutine. The ERR variable contains the error code, and ERL lists the line number where the error was detected.

Format:

```
IF ERR = error code THEN ...
IF ERL = line number THEN...
```

Since ERR and ERL are reserved variables, they cannot be placed to the left of the equal sign in a LET statement. The applicable error codes are listed in the section on MBASIC error messages.

FIELD—*defines a field in a random file buffer*

The field statement allocates space for variables in a random file buffer. A FIELD statement must execute before you can use a GET or PUT statement to extract or insert data in the random file buffer.

Format:

```
FIELD[#]<filenumber>,<field width>AS <string variable>
```

The same number used to open the file identifies the file whose variable fields are to be defined. The number of character spaces to be reserved in the buffer for the string variable is determined by the field width. The total number of bytes allocated by the field width cannot be larger than the record length specified when the file was opened. As many field statements as necessary can execute for the same file, with all being in effect simultaneously.

NOTE

Variables whose fields have been allocated should not be used in INPUT or LET statements. Once a variable field is defined, any subsequent input will move the pointer in the random file buffer.

Example:

```
FIELD #1,5 AS FIRST$, 10 AS SECOND$
```

FOR . . . NEXT—*establishes loop parameters*

The FOR statement defines an index, establishes an initial and terminal index value, and initiates a loop. The NEXT statement diverts execution back to the corresponding FOR statement.

Format:

```
FOR<index>=<initial>TO<terminal>[ STEP< degree>]

.
.
.
NEXT [<index>] [,< index>...]
```

The index is used as the counter and must be an unsubscripted variable. The initial value of the counter is established by the first numeric expression, with the terminal value being specified by the second expression. The counter is either increased or decreased each time the loop is completed and incremented by one unless modified by STEP.

You can use the STEP option to increment the index by degree. A positive STEP value causes the loop to be executed until the index value exceeds the terminal value. STEP can be negative, however, in which case the index is decremented until the counter value is less than the initial value. You can nest FOR/NEXT loops, as long as each has its own unique index variable.

The NEXT statement causes the associated FOR statement to execute until the loop is terminated. If you use an index variable, it must match the index variable in the corresponding FOR statement. More than one FOR/NEXT loop may be terminated by a single NEXT statement listing all the index variables. If no index variable is listed, the most recently encountered FOR statement is terminated. When you nest FOR/NEXT loops, the NEXT statement for the inside loop must appear before that for the outside loop.

Example:

```
10 A=1
20 FOR A=1 to 1000 STEP 2           <---.
30 PRINT A                              : Loop
40 NEXT A                           ----
```

GET—*reads record from random disk file*

The GET statement reads a random-disk file record into a random buffer.

Format:

GET [#] <file number>[,<record number>]

The file number is the number you assign to the file when you open it. The specified record number is read into the buffer, where an INPUT# or LINE INPUT# statement can subsequently access it. If no record number is indicated, the next sequential record following that most recently read is assumed. The highest record number that can be read is 32767.

Example:

```
GET #1, 17*I+1
```

GOSUB . . . RETURN—*executes a subroutine*

The GOSUB statement directs program execution to the beginning line number of a subroutine. The RETURN statement causes execution to continue with the statement after the most recent GOSUB.

Format:

```
GOSUB <line number>
.
.
.
RETURN
```

A subroutine can be referenced many times, and, if necessary, from within another subroutine. Nesting of subroutines is limited only by available memory. More than one RETURN statement may be located in a subroutine, each executing a return where needed. Subroutines should be preceded by a STOP, END, or GOTO statement to prevent unintentional execution.

Example:

```
10 GOSUB 40
20 PRINT "Executing main program again"
30 END
40 PRINT "This is the SUBROUTINE" <--------.
50 RETURN                <-------- Subroutine
```

GOTO—*branches as specified*

The GOTO statement causes program execution to continue, beginning at a specified line.

Format:

```
GOTO <line number>
```

Program execution is diverted to the indicated line number and continues with the first executable statement.

Example:

```
10 READ A                              <---------
20 PRINT "A=";A,                                :
30 B=3.14 * A^2                                 :
40 PRINT "AREA=" ;B                             :
50 GOTO 10              ------------------------:
60 DATA 1, 2, 3, 4, 5
```

Note: Endless loop caused by GOTO 10

IF(GOTO)-THEN-ELSE—*directs conditional branch*

An IF-THEN-ELSE statement either branches to a particular line number or executes a group of statements, depending on the value of an expression. An IF-GOTO-ELSE statement functions in the same manner but is strictly for branching to a specific line number.

Formats:

```
IF <expression> THEN [<statements>] or
            <line number>
 [ELSE] [<statements>] or <line number>

   IF <expression> GOTO <line number>
  [ELSE] <statements> or <line number>
```

When the conditional expression following IF is true (not zero), execution of the line number or group of statements following THEN (or line number after GOTO) takes place. When the expression is false (zero), the next executable statement is processed. The ELSE option may execute a group of statements or branch to a specific line.

Nesting of IF (THEN, GOTO) ELSE statements is limited only by line length. When you use multiple ELSE clauses, the closest THEN or GOTO statement is matched.

Examples:

```
IF STATE > 49 THEN HAWAII = NAMEN
IF STATE = 48 THEN ALASKA = NAMEN
  ELSE GOTO 410
IF (HOT=TRUE) GOTO 7734
```

INPUT—*assigns data to variables*

The INPUT statement prompts for data on the screen to be entered and, subsequently, assigned to variables.

Format:

```
INPUT [;] [<"prompt string">;]
        <list of variables>
```

A semicolon following INPUT prevents a carriage return entered via the keyboard from being echoed as a line feed. The prompt string is an optional message (in quotes) that requests the appropriate data during program execution. In the absence of a prompt, a question mark is displayed indicating that data is expected. The semicolon after the prompt string is necessary, but you can use a comma in its place to suppress the prompt question mark.

Variables to be defined are listed at the end of the INPUT statement. The data items supplied for a given variable are separated by commas. Data entered in response to the prompt is assigned sequentially to the variable list. The number of data items must correspond to the listed number of variables. The type of data item must agree with the type of variable. If the data entered is the wrong type or does not match the number of listed variables, the message Redo from start" appears.

Example:

```
10 INPUT "Pick a number and get one free"; A
20 PRINT A "PLUS 1 IS"; A+1
30 END
```

INPUT# *—assigns stored data to variables*

The INPUT# statement reads data from a disk file and assigns it to variables.

Format:

INPUT# <file number>, <variable list>

The file number identifies which disk file contains the data to be assigned to the variable list. The data items should be stored in the file as if entered at the keyboard. Numeric values are delimited by a space, carriage return, linefeed, or comma. String items are delimited in the same way but may be enclosed in quotes if one of these delimiters is intended for the string. A data item cannot exceed 255 characters. If end-of-file is reached while inputting data, remaining item(s) are ignored.

Example:

```
10 OPEN "I", #1, "DATA"
20 INPUT#1,N$, D#, H$
30 IF RIGHT$ (H$,2)= "78" THEN PRINT N$
40 GOTO 20
```

INSTR—*searches for a given string*

INSTR looks for the first occurrence of a string in another string and returns its position. The string to be searched is listed first, followed by the string portion being searched for. An optional offset can precede both of these strings to define the starting position of the search. INSTR returns a 0 if starting position is null or greater than the string being searched, or if the string being searched for cannot be found. If the string being searched for is null, a 1 is returned.

Format:

```
INSTR ([position],<search string>,
      <string being searched>)
```

Example:

```
10 A$ = "SEARCH STRING"
20 B$ = "S"
30 PRINT INSTR (4,A$,B$)
RUN
8
```

INT—*returns the integer portion of the argument*

The INT function eliminates the fractional portion of the argument and returns the integer portion. If the argument is a real number, it is first converted to an integer and then converted back to a real number.

Format:

INT (<numeric expression>)

Example:

PRINT INT (9.8)

KILL—*deletes a disk file*

The KILL statement deletes a given file from the current diskette.

Format:

KILL <filename>

Only closed files may be eliminated with the KILL. If KILL is specified for an open file, a "file already open" error will occur. You can expunge any type of file with the KILL statement.

Example:

```
100 KILL "DATA.FYL"
110 KILL "data.fyl" <------ Useful for
                            eliminating
                            lowercase
                            file names
```

LET—*assigns value to variable*

The LET statement assigns the value of an expression to a variable.

Format:

[LET] <variable>=<expression>

The word LET is optional and may be omitted; the equal sign itself denotes an assignment.

Example:

```
10 LET A=12
20 B=A+1
```

LINE INPUT—*assigns data to a string variable*

The LINE INPUT statement asks for a line of data that, when entered via the keyboard, is assigned to a string variable.

Format:

```
LINE INPUT [;] [,"prompt string"]
      ,<string variable>
```

LINE INPUT may be followed by a semicolon, in which case the carriage return used to enter the line is not echoed. The prompt string is a literal that appears on the screen asking for the line to be input. The line entered at the keyboard can be up to 254 characters long. The input line is entered by following it with a carriage return. You can use **^C** to escape from a LINE INPUT statement and return to the command level, if necessary; CONT resumes execution at the LINE INPUT.

Example:

```
10 LINE INPUT, A$
20 LINE INPUT; "NAME";N$
```

LINE INPUT#—*reads line from disk file*

The LINE INPUT# statement reads a line up to 254 characters long from a sequential disk file and assigns this data to a string variable.

Format:

LINE INPUT# <file number>, <string variable>

The file number identifies the file containing the data to be assigned to the string variable. The data for one LINE INPUT# statement is delimited from the next by a carriage-return linefeed. You usually use LINE INPUT# when you've broken lines of a data file into fields or when a program is reading an ASCII-saved program file into another program.

Example:

```
10 LINE INPUT#2, B$
```

LPRINT—*outputs data to the line printer*

The LPRINT and LPRINT USING statements perform the same action as the PRINT [USING] statements, except that data is output to the printer instead of on the screen.

Format:

LPRINT [<list of expressions>]

Lines printed through the LPRINT statement assume a 132-character-wide line.

LSET—*stores left-justified data in random file buffer*

The LSET statement transfers data from memory to a random file buffer, or you can use it with a nonfielded string variable to left-justify a string in a given field.

Format:

LSET <string variable> = <string expression>

LSET left-justifies a string expression in a corresponding string variable. If the string exceeds the field length, the material on the right is truncated. If numeric values are to be affected, you must first convert them to a string, using an MK (I,S,D)$ function.

Example:

```
10 LSET A$= MKS$(MAX)
20 LSET B$= "JOHN DOE"
```

MID$—*replaces portion of a string*

The MID$ statement substitutes characters in an existing string with replacement characters.

Format:

MID$ (<string exp1>,n [,m])=<string exp2>

The characters in the first string expression, beginning at the nth character position, will be replaced with the characters in the second expression. You can use the m parameter to signify the number of characters involved in the replacement. A MID$-initiated replacement may never exceed the character length of the original expression.

Example:

```
10 A$="1981"
20 MID$(A$,4)="2" or
MID$(A$,3,2) = "82"
30 PRINT A$
```

ON ERROR GOTO—*enables error-trap routine*

The ON ERROR GOTO statement executes an error-handling subroutine when an error is encountered.

Format:

ON ERROR GOTO <line number>

ON ERROR GOTO enables error trapping. When an error is encountered following this statement, the error-handling subroutine beginning at the specified line number executes. To disable error trapping so subsequent errors halt execution and display the appropriate error message, execute an ON ERROR GOTO O statement. You can disable error trapping within an error-handling subroutine when you expect errors for which you haven't formulated a recovery action.

Example:

```
10 ON ERROR GOTO 500
```

ON GOTO (GOSUB)—*conditional branch*

The ON GOTO statement directs program execution to one of several line numbers, depending on the value of an expression. The ON GOSUB statement conditionally executes a subroutine, beginning at one of several line numbers, depending on the value of an expression.

Formats:

ON <expression> GOTO <list of line numbers>

ON <expression> GOSUB <list of line numbers>

The expression value determines which of the line numbers will execute: fractions are rounded. If the expression is zero, or greater than the number of items in the list (without exceeding 255), the next valid statement is executed. If the value of the expression is negative or greater than 255, an "illegal function call" error occurs. Use a RETURN statement to terminate each subroutine.

Example:

```
10 ON A% GOTO 100,200,300,
20 ON B% GOSUB 400, 100, 200
```

OPEN—*activates an existing file*

The OPEN statement allows I/O operations to be performed on a named file. When this statement is executed, buffer space is allocated for the indicated file, and the mode of access to be used with the buffer is defined.

Format:

```
OPEN <mode>, [#] <file number>, <file name>, [<reclen>]
```

The first character of a string expression indicates the desired mode of access. The letter O indicates a sequential output mode, I indicates a sequential input mode, and R indicates either a random input or output mode of access. The file number may be 1, 2 or 3. To open files numbered from 4 to 15, reinitialize with A>MBASIC /F : (4–15). In all cases the file must be currently unoccupied. You will subsequently use the assigned file number to access the file. The file name is the name you gave to the file when you saved it. The default record length is 128 bytes, but you can alter it by specifying a length at the end of the statement. A file you open for sequential input or random access can occupy more than one file number; however, a file opened for output can be associated with only one file number.

OPTION BASE—*declares subscript array value*

The OPTION BASE statement declares the minimum value for an array subscript. The default base value is 0, and the highest base value possible is 1.

Format:

OPTION BASE (1 or 0)

Example:

OPTION BASE 1

OUT—*sends byte to port*

The OUT statement outputs a specific byte of data to a hardware output port.

Format:

OUT <port>, <byte>

The integer expression, representing the port and byte of data to be output, must range between 0 and 255.

NOTE

The Osborne 1 uses memory map I/O; the OUT instruction is used strictly for switching banks. (See the chapter on system specifications in the User Guide.)

Example:

```
10 OUT 20, 100
```

POKE — *writes byte in memory*

The POKE statement writes a specific byte of data to memory address.

Format:

POKE <address>, <byte>

The memory address to be referenced precedes the byte of data to be written. The address must be in the range of 0 to 65536, and the specified byte must range between 0 and 255. POKE is usually used in association with the PEEK function.

Example:

```
10 POKE &H5A00, &HFF
```

PRINT— *outputs data to the console*

The value of each expression listed in a PRINT statement is displayed on the screen.

Format:

PRINT [<list of expressions>]

Numeric and string expressions listed in the PRINT statement are evaluated and then printed on the screen. String expressions must be surrounded by quotation marks. If no expression is listed, a blank line is output.

You dictate the format of printed values through the use of punctuation characters. Each line is divided into print fields of 14 spaces each. A carriage return is output at the end of each PRINT statement line unless you suppress it with a comma or a semicolon. A comma after an expression indicates that the next value printed should start in the next field. Spaces or a semicolon after an expression mean that the next value to be printed should continue immediately after the value before it. When a comma or semicolon terminates the statement, the values printed for one PRINT statement may continue with those from another.

Positive numbers are printed with a leading space, while negative numbers are preceded by a minus sign; all numbers are followed by a space. Single-precision numbers that can be represented with less than 6 digits or double-precision numbers that can be represented with less than 16 digits are so printed.

NOTE

A question mark may be substituted for the word PRINT.

Example:

```
10 A=10
20 PRINT "TEN PLUS ONE EQUALS";
   A+1
(or ? "TEN PLUS ONE EQUALS"; A+1)
   30 END
```

PRINT USING—*prints formatted data*

The PRINT USING statement outputs formatted data on the screen.

Format:

```
PRINT USING <string expression>;
        <expression list>
```

The string expression consists of data-field specifiers and possible literals enclosed in quotation marks. These field specifiers define the format to be used when values for the expression list are output. Each expression in the list is delimited by a semicolon and will be printed in the specified format. String-field specifications used to define formatting are described below:

Literal Characters:

Characters that are not part of a string- or numeric-field format are assumed to be literal characters. Blank spaces, for example, are frequently used to separate fields. You can explicitly identify a character as a literal by preceding it with a slash character (\).

Examples:

```
PRINT USING "### ###";A$
PRINT USING "NO: ###";B$
PRINT USING "\ $"; C
```

Numeric Fields:

Numeric integer fields are indicated by one # character per digit. Specify a real numeric with a decimal point by placing the period where you need it. You can format numeric fields with commas by positioning them anywhere in the field. The position of the comma is irrelevant, since it will appear after every third character.

Examples:

```
PRINT USING "###.##"; A
PRINT USING "####,####"; B
```

You can specify exponential numeric format by appending one or more carets (^) to the end of the numeric-field definition. The decimal-point position affects the exponent's value, since four character positions are always added for the exponent.

Example:

```
        PRINT USING "#.####^"; A
```

You can pad a numeric field with leading asterisks. A dollar sign will appear in front of the first nonblank digit if the numeric-field definition begins with two dollar signs ($$). You can place a leading or trailing minus sign wherever you want it.

Examples:

```
PRINT USING "**#####.##";A
PRINT USING "$$####.##"; B
PRINT USING "-#####"; C
```

String Variables:

Specify variable-length fields with ampersands. A fixed-length field is defined by a slash (/) before and after it. An exclamation indicates a single-character string field.

PRINT USING# — *outputs data to disk*

The PRINT# and PRINT USING# statements function like the PRINT and PRINT USING statements, except that the data is written to a sequential disk file.

Format:

```
PRINT# <filenumber>,
[USING <string exp>;] <list of exps>
```

The file number is the number you assigned to the file when you opened it. The string expression consists of the field specifiers you need for formatting. The numeric or string expressions in the list will be written to the specified disk file. An exact image of the data is output, so pay careful attention to the use of formatting specifiers.

PUT — *writes data from buffer to disk*

The PUT statement writes a record from a random buffer to a random disk file.

Format:

PUT [#] <file number> [,<recordnumber>]

The file number assigned to the disk file when you opened it references the file in this statement. The file number is followed by the number of the record to be written. The record number can range between 1 and 32767. You usually use a PUT statement to recover data stored in the buffer by a PRINT#, PRINT USING#, or WRITE# statement.

Example:

```
PUT #3,4
```

RANDOMIZE—*seeds random-number generator*

The RANDOMIZE statement is used to seed the random-number generator.

Format:

RANDOMIZE [<expression>]

The expression indicates the number to be seeded and may range from −32768 to 32767. If you omit it, program execution stops, the message "Random Number Seed (−32768 to 32767)?" is displayed; execution resumes when you supply this number. If not reseeded, the RND function will return the same sequence of random numbers.

Example:

```
LIST
5 PRINT"START?";
10 DUMMY$=INKEY$
15 S=S+1
```

```
20 IF S=32769! THEN S=-32768!
25 IF DUMMY$<>"Y" THEN 10
30 RANDOMIZE S
35 FOR A=1 TO 50 STEP 2
40 PRINT INT(RND*100);
45 NEXT A
```

READ—*assigns values to variables*

READ statements extract values from DATA statements and assign them sequentially to variables.

Format:

READ <list of variables>

The numeric and string values in DATA statements are assigned sequentially to READ statement variables in the same order in which they appear in the program. The type of listed variables must match the data values used. One READ statement may read more than one DATA statement, and a single data statement may supply variables for multiple READ statements. If there are more READ variables than DATA items, an "Out of data" error occurs. If there are fewer READ variables than DATA items, variables are read from the next READ statement, if present, or are ignored.

NOTE

Use RESTORE statement for reading data from the start of a program.

Example:

```
10 FOR A=1 TO 10
20 READ B (A)
30 NEXT A
40 DATA 1, 2, 3, 4, 5, 6, 7
50 DATA 8, 9, 10
```

REM—*indicates a remark*

All characters to the right of a REM statement are considered remarks and are ignored by program execution.

Format:

REM <any characters>

You can continue a REM statement on the following line by using a backslash. You can use these statements as a target when branching execution through the use of a GOTO or GOSUB statement. Indicate remarks in midline with an apostrophe (') instead of the statement word REM.

Example:

```
REM COMPUTING THE MEAN
REM EQUIVALENT
REM ' MEANING THE COMPUTER
REM EQUIVALENT
```

RESTORE—*restores data-list pointer*

The RESTORE statement provides a method by which DATA statements can be read, beginning at any location in the program.

Format:

RESTORE [<line number>]

Execution of the RESTORE statement causes subsequent READ statements to access data beginning with the first DATA statement or a specified line number.

Examples:

RESTORE
RESTORE 20

RESUME—*resumes execution after an error routine*

A RESUME statement is located in an error-trap routine to cause the resumption of execution following an error-recovery procedure.

Format:

RESUME [0] [NEXT] [<line number>]

Both RESUME and RESUME 0 cause execution to resume with the statement where the error occurred. RESUME NEXT resumes execution with the statement following the one that caused the error. You can specify a line number where execution should resume.

RSET —*stores right-justified data in a random file buffer*

The RSET statement transfers data from memory to a random file buffer, or you can use it with a nonfielded string variable to right-justify a string in a given field.

Format:

RSET <string variable> = <string expression>

RSET right-justifies a string expression in a corresponding string variable. If the string exceeds the field length, characters are dropped from the right. Numeric values must be converted to string in order to be affected.

Example:

```
90 A$=SPACES (10)
100 RSET A$=B$
```

STOP—*terminates program execution*

When the STOP statement is encountered, program execution is halted and control returns to the command level.

Format:

STOP

A STOP statement may appear anywhere in a program to terminate execution. When execution halts with STOP, the message "Break in line ______" appears, indicating the number of the last line executed. Execution may be continued with a CONT statement.

Example:

```
STOP
```

SWAP—*exchanges variable values*

A SWAP statement is used to exchange the value of one variable for that of another.

Format:

SWAP <variable>, <variable>

Single-precision, double-precision, string, or integer variables may be exchanged. The swapped variables must, however, be of the same type or a "Type mismatch" error will occur.

Example:

SWAP A$,B$

WAIT—*suspends execution and monitors port status*

The WAIT statement temporarily halts program execution while the status of a hardware port is evaluated. Execution resumes when a particular bit pattern is achieved.

Format:

WAIT <port number>, I[,J]

The port number is exclusive OR'ed with the expression J, and then AND'ed with I. If J is omitted, its value is assumed to be 1. The port data is read repeatedly until the value is other than 0, in which case the next executable statement is processed. (This statement has little utility on the OSBORNE 1.)

NOTE

If an infinite loop occurs, the machine must be reset.

Examples:

```
WAIT 32, 3
WAIT 32, 3,2
```

WHILE/WEND—*performs a conditional loop*

The WHILE portion of the statement initiates the loop, which is repeated until the value of the expression is false (0). The WEND portion of the statement terminates the loop when the expression is deemed true (not zero).

Format:

```
WHILE <expression>
.
.
.
WEND
```

Looping continues until the value of the expression is zero. The loop may contain any number of statements, including other WHILE/WEND statements. When WHILE/WEND statements are nested, each WEND matches the most recent WHILE. When the expression is evaluated as other than zero, execution continues with the statement following WEND.

Example:

```
WHILE AMOUNT <=MAX     <---------
WEND                   ---------:  Loop
```

WRITE—*outputs data at console*

The WRITE statement sends specified data to the console.

Format:

WRITE [<list of expressions>]

Numeric and/or string expressions in the list are evaluated and displayed on the screen. Each expression in the list must be separated from the next with a comma. If the list of expressions is omitted, a blank line is output. A carriage return is supplied following the last item in the list.

Example:

```
10 A=10: B=20: C$= "THE END"
20 WRITE A,B,C$
```

WRITE#—*writes data to disk*

The WRITE# statement outputs specified data to a named sequential disk file.

Format:

WRITE# <filenumber>, <list of expressions>

You must open a file in the output mode before you can write data to it. The file to be written to is subsequently referenced by the same number you assigned the file when it was opened. Commas in the list delimit string and numeric expressions.

The WRITE# statement performs essentially the same function as the PRINT# statement, except that delimiters are supplied when the data is written to disk.

Example:

```
WRITE #1,A$,B$
```

MBASIC Functions

ABS—*returns absolute value*

The ABS function returns the absolute value of a numeric expression.

Format:

```
ABS (<numeric expression>)
```

Example:

```
COST-ABS (5*40 (-5))
PRINT ABS(COST-PROFIT)
```

ASC—*returns an ASCII value*

The ASC function returns the ASCII integer value for the first character of the argument. Only the first character is considered. If the expression is evaluated as a null string, or if the argument is numeric, an error will occur.

Format:

```
ASC (<string expression>)
```

Example:

```
I%=ASC (STRING$)
FIRST% = ASC (FIRST.AME$)
```

ATN—*returns arctangent of the argument*

The ATN function is used to return the arctangent of an argument. The argument must be expressed in radians. The value returned is a real number. The arctangent returned can be used to compute various inverse trigonometric functions.

Format:

```
ATN (numeric expression)
```

Examples:

```
ANGLE = ATN (X)
ASIN = ATN (X/SQR(1.0-X*X))
```

CDBL—*converts argument to double precision*

The CBDL function converts a numeric expression to a double-precision argument.

Format:

```
CDBL (<numeric expression>)
```

Example:

```
10 A = 126.67
20 PRINT A; CDBL (A)
```

CHR$—*returns ASCII string equivalent of the argument*

The CHR$ function converts the argument to a single-character string. The argument is assumed to be an ASCII code. The function returns the ASCII character represented by the argument. If the argument is greater than 255, the high-order byte is ignored.

Format:

```
CHR$ (<numeric expression>)
```

Example:

```
BELL$ = CHR$ (7)
```

CINT—*converts the argument to an integer*

The CINT function converts a numeric expression to an integer by rounding the fractional portion. An overflow error occurs if the numeric expression is less than −32768 or larger than 32767.

Format:

```
CINT (<numeric expression>)
```

Example:

```
PRINT CINT (9.6)
```

COS—*returns the cosine of the argument*

The COS function returns the cosine of the argument. The argument must be expressed in radians. The value returned is a real number.

Format:

COS (numeric expression)

Example:

```
10 A=3*COS (.5)
20 PRINT A
```

CSNG—*converts the argument to single precision*

The CSNG function converts a numeric expression to a single-precision number.

Format:

CSNG (<numeric expression>)

Example:

```
10 A# = 123.345
20 PRINT A#; CSNG (A#)
```

CVI, CVS, CVD—*convert string value to numeric value*

The CV(I,S,D) functions convert string to numeric values. CVI converts a two-byte string value to an integer, CVS converts a four-byte string to a single-precision number, and CVD converts an eight-byte string to a double-precision number.

Formats:

CVI (2-byte string)
CVS (4-byte string)
CVD (8-byte string)

Example:

```
10 FIELD #1, 4 AS N$, 12 AS B$,...
20 GET #1
30 Y=CVS (N$)
```

EOF—*checks for end of file*

The EOF function returns −1 (true) if the end of a sequential file has been reached. This function is used to test for the end of file before inputting from it, to avoid "Input past end" errors.

Format:

```
EOF (<file number>)
```

Example:

```
10 OPEN "I" ,2, "DATA"
20 B=0
30 IF EOF (2) THEN 100
40 INPUT #2, M(B)
50 B=B+1:GOTO 30
```

EXP—*returns the exponent of the argument*

The EXP function returns the value of the constant raised to the power of the argument. The value returned is a real number, even if the value is an integer.

Format:

```
EXP (<numeric expression>)
```

Example:

```
10 X = 2
20 PRINT EXP (X-1)
```

FIX—*returns integer portion of the argument*

The FIX function returns the truncated integer portion of a numeric expression. FIX performs the same function as the formula SGN (X) * INT(ABS (X)), except that the next lower number for negative X is not returned.

Format:

FIX (numeric expression)

Example:

PRINT FIX (49.75)

FRE—*shows amount of data memory area remaining*

The FRE function returns the number of available memory bytes in the dynamic, or free, storage area. The value FRE returns is a real number. Free storage may consist of two or more noncontiguous memory blocks. Using FRE(" ") forces a "garbage collection" before returning the number of free bytes (may take 1 to 1½ minutes).

Format:

FRE (0 or string expression)

Example:

PRINT FRE (0)

HEX$—*returns the hex value of the argument*

The HEX$ function returns a string representing the hexadecimal value of the decimal argu-

Format:

HEX$ (<numeric expression>)

Example:

```
10 INPUT X
20 A$ = HEX$ (X)
30 PRINT X "DECIMAL IS" A$
  "HEXADECIMAL"
```

INKEY$—*returns character string from console*

The INKEY$ function returns either a one-character string which is read from the console or a null string if no character is pending. All characters are passed to the program, except for ^C, which causes the program to be terminated.

Format:

INKEY$

Example:

```
1000 'TIMED INPUT SUBROUTINE
1010 RESPONSE$=" "
1020 FOR I%=1 TO TIMELIMIT%
1025 A$ = " "
1030 A$=INKEY$ : IF LEN (A$) = 0
  THEN 1060
1040 IF ASC (A$) = 13 THEN TIMEOUT%
  = 0 : RETURN
1050 RESPONSE$ = RESPONSE$ + A$
1060 NEXT I%
1070 TIMEOUT% = 1 : RETURN
```

INP—*returns a byte from an I/O port*

The INP function returns a byte from a selected input/output port. INP returns an integer that is the value read from the port addressed by the integer expression. This function has little utility on the Osborne 1.

Format:

```
INP (integer expression (0–255))
```

Example:

```
100 A=INP (255)
```

INPUT$—*returns string read from console or disk*

The INPUT$ function returns a string of characters read from the terminal or, optionally, from a disk file. If the terminal is the source of the input, no characters are echoed, and all control characters with the exception of **^C** are passed through to the program.

Format:

```
INPUT$ (numeric expression) [ ,#filenumber]
```

Example:

```
10 PRINT "Enter P to continue or S to stop"
20 X$=INPUT$ (1)
30 IF X$ = "P" THEN 500
40 IF X$ = "S" THEN GOTO 100
```

LEFT$—*returns leftmost characters*

The LEFT$ function returns the leftmost characters of the argument. The numeric expression specifies the number of characters to be returned. If the number of characters to be returned is greater than the length of the string expression, the entire first argument is returned. If the numeric expression is zero, a null string is returned; if it is negative, an error will occur.

Format:

```
LEFT$(<string expression>,
   <integer expression>)
```

Example:

```
10 A$ = "LEFTMOST"
20 B$ = LEFT$ (A$,4)
30 PRINT B$
RUN
LEFT
OK
```

LEN—*returns the number of characters in a string*

The LEN function returns the number of characters in a string expression. It returns 0 for a null string.

Format:

```
LEN (<string expression>)
```

Example:

```
10 X$ = "NUMBER OF CHARACTERS"
20 PRINT LEN (X$)
```

LOC—*shows next random record or sequential sector*

The LOC function, when invoked for a random file, returns the next default record number to be accessed. When you use LOC for a sequential file, the number of sectors read from or written to the file since its return is shown.

Format:

```
LOC (<file number>)
```

Example:

```
10 IF LOC (1) > 50 THEN STOP
```

LOG—*returns the natural logarithm of the argument*

The LOG function returns the the natural logarithm of the specified argument. The argument must be greater than zero, or an error will occur. You can use this function to calculate logarithms to other bases.

Format:

```
LOG (<numeric expression>)
```

Example:

```
PRINT LOG (45/7)
```

LPOS—*returns print strike position*

The LPOS function returns the assumed current position of the line printer hammer within the line printer buffer. LPOS does not necessarily give the physical position of the printhead. A numeric expression is used as a dummy argument.

Format:

LPOS (<numeric expression>)

Example:

```
10 IF LPOS (X) > 60 THEN LPRINT CHR$ (13)
```

MID$—*returns a portion of a string*

The MID$ function returns a string that may be any portion of another string. MID$ can accomplish the same function as either RIGHT$ or LEFT$, but, in addition, MID$ can return a string from the middle of another string.

Format:

MID$ (<string expression>,
<integer expression>,[integer])

A portion of the string expression is returned, beginning at the position defined by the integer expression that follows it. An optional integer expression specifies the length of the string to be returned. If the last expression specifies a character position beyond the end of the string, then characters from the position specified by the first integer expression up to the end of the string are returned. A null string is returned if the starting position specified by the second expression is greater than the length of the string, or if the third expression is zero.

Example:

```
10 A$ = "EXTRACT"
20 B$ = "SEARCH STRING EXPRESSION"
30 PRINT A$; MID$ (B$,8,6)
```

MKI$, MKS$, MKD$—*converts numeric value to string*

The MK(I,S,D)$ functions convert numeric values to string values. MKI$ converts an integer to a two-byte string, MKS$ converts a single-precision number to a four-byte string, and MKD$ converts a double-precision number to an eight-byte string. These functions must be used on numeric values that are placed in a random file buffer with an LSET or RSET statement.

Formats:

MKI$ (<integer expression>)
MKS$ (<single precision expression>)
MKD$ (<double precision expression>)

Example:

```
10 AMT = (K+T)
20 FIELD #1, 8 AS D$, 20 AS N$
30 LSET D$ = MKS$(AMT)
40 LSET N$ = A$
50 PUT #1
```

OCT$—*returns octal value of the argument*

The OCT function returns a string that represents the octal value of the decimal argument. The argument is rounded to an integer before being evaluated.

Format:

OCT$ (<numeric expression>)

Example:

```
PRINT OCT$ (24)
30
```

PEEK—*returns contents of memory location*

The PEEK function returns the contents of a selected memory location. The numeric expression represents the memory location to be examined. PEEK returns an integer value equal to the contents of the specified memory location. For memory locations greater than 32767 the argument must be negative, in which case you should express the argument in hexadecimal notation for clarity.

Format:

PEEK <numeric expression>

Example:

```
BDOS% = PEEK (6)+PEEK (7)*256
```

POS—*returns position of the next print character*

The POS function returns the current cursor position. The leftmost column is position 1.

Format:

POS (<numeric expression>)

Example:

```
IF POS (X)>60 THEN PRINT CHR$ (13)
```

RIGHT$—*returns rightmost characters of a string*

The RIGHT$ function returns the rightmost characters of the argument. The numeric expression specifies the number of characters to be returned. If the number of characters to be returned is greater than the length of the string expression, the entire first argument is returned. If the numeric expression is zero, a null string is returned; a negative numeric expression causes an error to occur.

Format:

```
RIGHT$ (<string expression>,
    <integer expression>)
```

Example:

```
10 A$ = "STRING EXPRESSION"
20 B$ = RIGHT$ (A$,4)
30 PRINT B$
RUN
SION
```

RND—*returns a random number*

The RND function returns a random number between 0 and 1. The RND function generates the next random number in a sequence based on the current seed. The value returned is a real number. The RANDOMIZE statement must be executed to generate a seed and a random number sequence. An optional numeric expression may direct the sequence. If the expression is less than 0, then the sequence repeats; if it is greater than 0 or omitted, the next random number in the sequence is generated. If the expression equals 0, then the same sequence repeats.

Format:

```
RND ([numeric expression])
```

Example:

```
10 FOR I = 1 to 5
20 PRINT INT(RND*100);
30 NEXT
RUN
24  30  31  51  5
Ok
```

SGN—*returns the sign of the argument*

The SGN function returns −1, 0, or 1, depending on whether the argument is negative, zero, or positive, respectively.

Format:

```
SGN (<numeric expression>)
```

Example:

```
IF SGN(TOTAL) = −1 THEN GOSUB 200
ON SGN(X) + 2 GOTO 10, 20, 30
```

SIN—*returns the sine of the argument*

The SIN function returns the sine of a numeric expression in radians. The expression is calculated in single precision.

Format:

```
SIN (<numeric expression>)
```

Example:

```
PRINT SIN (2.7)
```

SPACE$—*returns a specified number of spaces*

The SPACE$ function returns a string of spaces of a specified length. The argument representing the length must range between 0 and 255 and will be rounded to an integer.

Format:

SPACE$(<numeric expression>)

Example:

```
10 FOR I = 1 TO 5
20 X$ = SPACE$(I)
30 PRINT X$;I
40 NEXT I
```

SPC—*outputs blanks*

The SPC function outputs a specified number of blanks in conjunction with PRINT or LPRINT statements. The number specified can range between 0 and 255.

Format:

SPC (<integer expression>)

Example:

```
"SPACED" SPC (20) "OUT"
```

SQR—*returns the square root*

The SQR function returns the square root of an argument which must be larger than 0.

Format:

SQR (<numeric expression>)

Example:

```
10 FOR X = 10 TO 25 STEP 5
20 PRINT X, SQR (X)
30 NEXT
```

STR$—*returns string representation*

The STR$ function returns the string representation of the argument.

Format:

STR$ (<numeric expression>)

Example:

```
PRINT STR$ (35)
```

STRING$—*returns a string*

The STRING$ function returns a string whose length is specified by the first integer expression. The characters in the returned string have the same ASCII code as specified by the second integer expression or begin with the first character of the specified string expression.

Format:

STRING$ <integer exp>,
<integer or string exp>

Example:

```
A$ = STRING$(75, "A")
B$ = STRING$ (75, 25)
```

TAB—*positions cursor*

The TAB function positions the cursor at the column specified by the argument. If the cursor is already positioned beyond the specified column, the cursor moves to that position on the next line. TAB may be used only in PRINT statements and must range between 1 and 255.

Format:

TAB (<integer expression>)

Example:

```
PRINT TAB (20),A$
```

TAN—*returns the tangent in radians*

The TAN function returns the tangent of a numeric expression in radians, which calculates as a single-precision expression. If an overflow occurs, the "OVERFLOW" message is displayed, the machine infinity is supplied, and execution continues.

Format:

TAN (<numeric expression>)

Example:

```
A= TAN (3.14)
```

USR—*calls a user routine*

The USR function calls an assembly-language subroutine identified by the specified argument. The user number, which may range from

0 to 9, corresponds to the user number the DEF USR statement assigns. User number 0 is assumed if you don't specify one.

Format:

USR [user number] (<numeric expression>)

Example:

X = USR9 (Y)

VAL—*converts a string to a real number*

The VAL function converts a string into a real number. The real number VAL returns equals the number the string expression represents. This conversion is equivalent to numeric keyboard input in response to an INPUT statement.

Format:

VAL (string expression)

Example:

PRINT VAL ("3.1")

VARPTR—*returns starting address of the argument*

The VARPTR function returns the beginning address of a variable name or a disk I/O buffer assigned to a file number. Any type of variable name (numeric, string, array) may be examined, with the address returned being in the range 32767 to −32768. When a negative address is returned, add it to 65536 to obtain the actual address. This function is usually for finding the address of a variable or array and then for passing it to an assembly-language subroutine.

Format:

VARPTR (<variable name>) or
(#<file number>)

Example:

A= VARPTR (X)
B= VARPTR #(9)

Initialization Options

/F:# if you want to open more than 3 MBASIC files at one time. The maximum numer is 15.

/M:# (decimal or hex) sets the highest memory location to be used by MBASIC; when you want to leave some empty RAM space for other purposes.

/S:### must be used to access random files larger than 128 bytes per record.

Software Error Messages

This section provides a complete listing and explanation of CP/M, Wordstar, SuperCalc, CBASIC, and MBASIC error messages.

CP/M Error Messages

BOOT ERROR

This message, repeated endlessly (or until the RESET button is pressed) indicates that drive A contains no diskette, an unformatted diskette, a diskette without a CP/M operating system, or simply the wrong kind of diskette; or that the diskette door is improperly latched.

Several CP/M error messages print due to error conditions encountered by the 8080 assembler:

NO FILE or **NO SOURCE FILE PRESENT**

The specified file was not found.

NO SPACE or **NO DIRECTORY SPACE**

The diskette directory is full.

SOURCE FILE NAME ERROR

An improper file name was found.

SOURCE FILE READ ERROR

The assembler could not read the source code correctly.

OUTPUT FILE WRITE ERROR

The assembler could not read the source code correctly.

Assembly Language

The error-flag codes and their line numbers are echoed at the console when errors occur within an assembly-language program:

D

Data error. Data statement element cannot be placed in the specified area.

E

Expression error. Expression is improperly formed and cannot be computed.

L

Label error. Illegal label context.

N

Not implemented. The mnemonic used refers to MAC assembler mnemonics, not ASM.

O

Overflow error. Expression is too complicated.

P

Phase error. Inconsistent label value encountered.

R

Register error. Value specified as register not compatible with the operation code.

V

Value error. Improperly formed operand in the expression.

ED

The Digital Research editor (ED) uses the following error codes:

?

Unrecognized command.

>

Memory buffer full.

#

Cannot apply command number of times specified.

O

Cannot open LIB file in R command.

The following BDOS errors may be encountered:

BDOS ERR ON X: BAD SECTOR

CP/M could not read from or write to the diskette in question.

BDOS ERR ON X: SELECT ERROR

CP/M does not recognize the drive selected.

BDOS ERR ON X: R/O

The diskette you attempted to write information onto is write-protected.

WordStar Error Messages

@ @ @ @

File WSMSGS.OVR not found. Menus & messages will

display as @ @ @ @ only

Indicates that the message file cannot be located and most messages will appear as @@@@; under this condition the help level is automatically set to 0. Under some circumstances, you may not be able to continue editing.

EDIT FUNCTION ERROR MESSAGES:

```
* * * INTERRUPTED * * * Press ESCAPE key [ ]
```

This message occurs when a function is in progress and the interrupt command **^U** is typed.

```
* * * NOT FOUND: string * * * Press ESCAPE Key [ ]
```

This message indicates that a search initiated by a FIND (^QF), REPLACE (^QA), or FIND, REPLACE AGAIN (^L) command could not locate the specified string.

```
* * * ERROR E5: THAT MARKER NOT SET * * *
Press ESCAPE Key [ ]
```

Reference to a marker which you did not set during the current editing session.

```
* * * ERROR E6: BLOCK BEGINNING NOT MARKED
(OR MARKER IS UNDISPLAYED) * * *
Press ESCAPE Key [ ]
```

Reference to a beginning block marker that has either not been set or is currently hidden.

```
* * * ERROR E7: BLOCK END NOT MARKED
(OR MARKER IS UNDISPLAYED) * * *
Press ESCAPE Key [ ]
```

Reference to an end block marker that has either not been set or is currently hidden.

```
* * * ERROR E8: BLOCK END MARKER BEFORE BLOCK
BEGINNING MARKER * * * Press ESCAPE Key [ ]
```

The block-end marker has been set before the block-beginning marker.

```
* * * ERROR E9: BLOCK TOO LONG—MOVE OR COPY IN
TWO SMALLER BLOCKS * * * Press ESCAPE Key [ ]
```

The size of the text in the currently marked block exceeds the amount that WordStar can handle. Divide the block and move it in portions. There is no limit to the size of blocks that can be written.

```
* * * ERROR E10: CURSOR NOT IN RANGE FOR
COLUMN MOVE/COPY * * * Press ESCAPE Key [ ]
```

A column block move or copy cannot be accomplished because the cursor lies in a negative print position or past column 240.

```
* * * ERROR E11: THAT FILE EXISTS ON DESTINATION
DISK, DELETE EXISTING FILE FIRST, OR USE A DIFFERENT
DISKETTE. * * * Press ESCAPE Key [ ]
```

This message is caused by an attempted file transfer to a diskette that already contains a file of the same name. If such a transfer were successful, the original file would be replaced.

```
* * * ERROR E12: DISK FULL * * *
Press ESCAPE Key [ ]
```

Indicates that the capacity of the diskette to which data is being sent has been reached. You should take preventive measures to ensure that this catastrophic error does not occur. If this error occurs while you are moving the cursor toward the beginning of a large file, try moving the cursor to the end of the file and then saving with ^KS. If the error occurred while you were saving a file, press ESCAPE and delete some files with ^KJ, or try writing a portion of the file to the A drive.

```
* * * ERROR E13: COLUMN READ/WRITE NOT
ALLOWED * * * Press ESCAPE Key [ ]
```

This message indicates that reading and writing of blocked columns between files is not currently implemented. You can accomplish this maneuver by reading or writing a regular block containing the column and then copying or moving the column and erasing the remainder.

NOTE

You should note internal errors 115, 116, 117, 118, 119, and 136, and if you can reproduce them, you should report them to Osborne Computer Corporation and MicroPro International.

FILE NAME ERRORS:

```
filename.typ NOT FOUND
```

The file name you supplied for a FILE NAME? prompt cannot be located.

```
INVALID FILE NAME: string entered
```

The string you entered in response to a FILE NAME? prompt is invalid.

```
Can't edit a file type.BAK or. $ $ $ – rEname or cOpy
the file before editing
```

This message occurs when you make an attempt to edit (D or N) a WordStar backup or temporary file. The file can be renamed or read into another file if your intent is to edit the file.

```
File WS.COM Not Found - Can't Run program unless
WS.COM is available
```

The message means the main WordStar program (WS.COM or other if changed through installation) cannot be found.

```
File x:filename.typ ALREADY EXISTS
```

The name being assigned to a file through the RENAME command E already exists; choose a different name or rename the original.

```
FILE x:filename.typ NOT ON SAME DRIVE
```

This message tells you that a file cannot be renamed from one drive to another.

```
FILE x:filename.typ EXISTS-OVERWRITE? (Y/N): [  ]
```

This message appears when you try to copy a file to an already existing file name. If you want to copy over the original file, simply type Y for yes.

MailMerge Error and Warning Messages

MailMerge has several warning and error messages. An error message always appears when an invalid DOT command is encountered. Error messages also occur when a referenced file cannot be found, or when the contents of a data file do not correspond to the keywords listed by .RV in the document file.

MailMerge displays a warning message on encountering certain special conditions. Processing continues even though the warning messages are displayed. Errors indicated on the screen remain in the file being processed.

Some messages inform you of conditions that might not have adverse consequences but that you should nevertheless consider. Review all messages that accumulate on the screen and note any conditions that require action. Edit the file to correct any mistakes that you might have made while creating the file.

The following warning and error messages appear when problems are detected:

```
* * * Invalid DOT command ignored
```

A DOT command not used in its proper form or context will cause this error message. The DOT command in question is displayed on the following line. A more specific error message may accompany this message in some cases.

```
* * * Insert diskette with file (:) filename.typ
then press RETURN
```

Requests insertion of the diskette containing the named file into the indicated drive (shown after the message). This message results when you process a .DF or .FI command with the word CHANGE after the file name.

```
* * * Cannot change disk in drive (:), request ignored
```

Displayed when you have specified a diskette change for the drive holding the program files. To avoid this type of error, limit a diskette change to the B disk drive. An attempt to access the specified file will take place, just in case the file is on the specified drive (or the logged drive if you did not specify one). If the file cannot be found, then the following message is displayed:

* * * file (:) filename.typ not found

Indicates that the file name called for could not be found in the specified drive (or the logged drive if no drive was specified). Both drives are searched in order to locate the specified file. If the file cannot be found, processing will continue without it.

* * * but found, and will use, (:) filename.typ

When a file cannot be located on the specified drive, a search of the other drive occurs. The message above is displayed if a file with the specified name is found. This message tells you where the file was located, allowing you to determine if this is the file you intended to reference.

* * * No .DF before .RV

No data file with the name you specified in .DF could be found, or the .DF command itself could not be found. If a "file not found" message does not accompany this message, then check the file to make sure that a .DF was used and that it was placed before .RV. Printing of the document file will continue without data for the keywords.

* * * WARNING: Overlong data value truncated

Displayed when data items contain more than the maximum 200 allowable characters. Only the first 200 characters are used, the excess may be omitted or may be used as data for the next variable read by .RV. This message might indicate an error in the format of the data file.

* * * Invalid variable name in .RV command ignored

Means that one or more of the keywords identified in the .RV command were not in their valid form. Ampersands should not be used in the keywords listed by .RV.

```
* * * WARNING: data exhausted, null value(s) used
```

Displayed when data from the data file is exhausted before all the keywords listed in .RV have been read. The variable identifiers for which there is no data will be assigned a null value consisting of no characters. Printing will usually stop before the next copy is printed after this message is displayed.

This type of error usually occurs when the last record of a data file is being processed. The error condition might be present anywhere in the data file, even though the error is not detected until processing of the file is almost complete. This type of error is typically caused by a lack of proper commas and carriage returns. Sometimes the error is caused by the presence of the wrong data file.

NOTE

When a document file containing .DF or .RP is processed, MailMerge searches ahead after each printed document looking for data to be used in the next document. If no data item is encountered, then processing will terminate without the above message being printed.

MISCELLANEOUS ERRORS:

```
* * * ERROR E38 (-42): BAD OVERLAY FILE, OR
WRONG VERSION OVERLAY FILE * * *
Press ESCAPE Key [ ]
```

```
* * * ERROR E43 (44): WRONG VERSION OVERLAY
FILE * * *
```

Occurs when the wrong version of an overlay file (.OVR) is being used; sometimes caused by a damaged diskette.

```
E46: Overlay file WSOVLY.OVR Not found * * *

Press ESCAPE Key [ ]
```

The file named WSOVLY.OVR is missing from the current version of WordStar.

```
* * * ERROR E47: FILE MERGEPRN.OVR NOT FOUND

(The separately supplied file MERGEPRIN.OVR is required

for use of MailMerge) * * * Press ESCAPE Key [ ]
```

This message appears when a MailMerge operation is attempted and the MailMerge program file (MERGEPRIN.OVR) cannot be located on the currently logged drive.

```
* * * ERROR 52: PROGRAM IS AN EMPTY FILE!? * * *

Press ESCAPE Key [ ]
```

This message appears when an invalid program is referenced through the RUN-A-PROGRAM command, R.

```
* * * ERROR E53: PROGRAM TOO BIG FOR

MEMORY AVAILABLE UNDER WordStar * * *

Press ESCAPE Key [ ]
```

The program trying to be run is too large to be run through WordStar.

WARNINGS:

```
* * * WARNING: WORD TOO LONG TO FIT MARGINS
```

This warning appears when too many characters are strung together on a line.

CAN'T DISPLAY PAGE BREAKS IN A NON-DOCUMENT FILE

The Page-Break display command, ^OP, was issued and is not valid in the nondocument mode.

PUT AT FILE BEGINNING FOR CORRECT PAGE-BREAK DISPLAY

Indicates that the DOT commands being used should be located at the beginning of the file so that the page breaks can be determined accurately.

?

A lingering question mark appears in the rightmost flag column when you specify an erroneous or incomplete DOT command, a missing numeric argument, an unrecognizable code, or an excessive number.

*** * * WARNING: WRONG VERSION OF WSMSGS.OVR— SOME MESSAGES MAY BE INCORRECT * * ***

This warning is telling you that the version of the message file (WSMSGS.OVR) you are using is incompatible with the version of WS.COM being used.

*** * * WARNING: DISK FULL, DELETING OLD .BAK FILE TO MAKE SPACE (NORMALLY, THE PREVIOUS BACKUP FILE IS DELETED ONLY AFTER EDIT IS SUCCESSFULLY COMPLETED).**

This warning message informs you that the diskette being written to is becoming full. You should take action immediately to avoid serious complications.

```
WARNING: You are editing the same file as you are printing.
WordStar will not allow you to save the edited version until
the print has completed or has been abandoned.
```

Indicates that the file being printed cannot be simultaneously edited.

NOTE

Occasionally the Osborne 1 beeps and the screen fills with lines of exclamation points. This condition occurs when you are issuing more commands than the computer can handle.

PRINTING MESSAGES:

```
filename.typ NOT FOUND
```

The file named for printing could not be located on the logged or indicated drive.

```
INVALID FILE NAME: string
```

The string you entered in response to a print prompt is invalid.

```
WARNING: You are printing the same file as you are editing.
The last saved version will be printed, not reflecting unsaved
changes. Furthermore, WordStar will not allow you to save
the edited version while the print is in progress.
```

Indicates that the file referenced for printing is currently being edited and the backup version will be printed if it is available.

```
* * * PRINT OUTPUT DISK FULL. PRINT PAUSED. * * *
```

Occurs when the diskette onto which the print-output file is being written becomes full. Delete unneeded files when applicable.

INFORMATION MESSAGES:

```
FINISHING PRINT BEFORE EXIT (type ^ U to cancel exit
command) . . .
```

Occurs when you try to exit to CP/M while a print operation is in progress.

```
FINISHING PRINT OF SAME FILE BEFORE SAVING
(type ^ U to cancel Save command) . . .
```

Occurs when you try to save a file that is currently being printed.

```
FINISHING PRINT OF .BAK FILE BEFORE SAVING
(type ^ U to cancel Save command) . . .
```

Occurs when you try to save a file you are editing while its backup version is printing.

NOTE

If a FATAL error occurs, call your authorized Osborne dealer.

SuperCalc Error Messages— Causes and Cures

The following material provides you with a detailed description of the error messages that you may receive while using the SuperCalc program. They are discussed in alphabetical order. For each error message, we have included a brief explanation of its cause and a procedure for correcting the situation that resulted in the message.

Here is a list of errors considered:

`Column ERROR`

Incorrect specification of a column. Correct specification is a letter from A to Z or two letters from AA to BK. To correct: use the in-line editor to correct the entry and reenter the command, or cancel the command with ^Z.

`Disk FULL`

The disk designated to receive the file does not have enough space. The SuperCalc program will ask if you want to redo the operation (Y) or not (N). If you want to redo it, remove the disk and insert another one that has enough space; then press **Y**. If you press **N**, the operation is aborted, and you return to the SuperCalc program.

`Drive not ready`

This is a system error message from the BIOS portion of your CP/M operating system. It is possible that the drive will become ready and that retrying will work. Check to make sure that the disk drive is closed.

`File NOT on disk`

This occurs with the load command. The file name given is not found on the disk drive specified or implied in the entry. Check your command entry.

1. Check the drive designation. If you did not specify one, the SuperCalc program assumes you mean the current default drive.
2. Check the spelling of the file name.
3. Check to see that the correct diskette is in the drive.

In cases 1 or 2, use the in-line editor to correct the drive designation or the file name and reenter the command.

In case 3, either place the correct disk in the drive or, if this is not feasible, cancel the commmand with ^Z.

Formula ERROR

There are two possible causes.

1. You entered text without a leading ". SuperCalc assumes that you intended to enter a formula, and it cannot make sense out of the entry as a formula.
2. There is some error in the way you specified a formula. Check it for correct specification of function name, correct use of expressions, balanced parentheses, valid cell names, etc.

To correct: use the in-line editor to correct your entry and reenter, or cancel the entry with ^Z.

Memory FULL

Too much content in the worksheet. (This is a different case from Worksheet Full, described below, in which there are too many cell stubs on the worksheet.) To correct: blank any contents that you can spare. If you can, move material to the upper left of the worksheet, trying to preserve a roughly rectangular shape. Save the worksheet, ZAP the screen, and reload the worksheet.

If this does not free enough space, then you must break the worksheet into convenient portions for future work. To do this, ZAP the screen and reload selected portions of the saved worksheet. Build two or more worksheets out of these portions, saving them as separate worksheets.

Overlay ERROR

This is a serious error that prevents the SuperCalc program from being used. There are two possible causes:

> The SuperCalc program has (1) not been "installed" or (2) has been "installed" incorrectly. Installing the SuperCalc program means customizing it for your computer system. This customizing involves specifying the terminal that you are using, the disk drives available, the memory space available, and the version of the CP/M operating system that you use. SuperCalc comes correctly installed for the Osborne 1. See the Appendix for information on SuperCalc Installation procedures.

To correct:

1. If you are installing the SuperCalc program yourself, reinstall it, checking the installation documentation carefully as you proceed.
2. Consult your authorized Osborne 1 dealer for information and assistance.

Protected Entry

This message can appear as the result of an error, or it may appear as an informational note. If the message is the result of an error, it will appear during data entry or the edit commmand. You are attempting to enter data into an active cell that is protected. You must either remove the data from the entry line or cancel the edit command.

This message may appear as an informational note during a blank, copy, load, or replicate command. If there are protected cells in the area being blanked or in the destination area of the copy, load, or replicate command, the protected cells in the area remain unchanged; the other cells in the area have been changed. If you meant to leave the protected cells unchanged, all is well. If not, you may wish to unprotect them and redo the command.

Range ERROR

Incorrect specification of a range. A range may be a single cell, a partial column, or a partial row. To correct: use the in-line editor to correct the entry and reenter the command, or cancel the command with ^Z.

Row ERROR

Incorrect specification for a row. Correct specification is a number from 1 to 254. Use the in-line editor to correct the entry and reenter the command, or cancel the command with ^Z.

Replicate Definition ERROR

The destination may be specified incorrectly, or the destination area may be too small.

1. Specification error for the destination.

a. If the source is a single cell, the destination should be specified as a partial column or partial row.

b. If the source is a partial column, the destination should be specified as cells on the upper row of the destination. This will look like a partial row.

c. If the source is a partial row, the destination should be specified as cells in the column on the left of the destination. This will look like a partial column.

2. Destination area is too small (will not fit).

Given the size of the source and the location of the destination, the result will not fit within the worksheet boundaries. Correct the specification using the in-line editor and reenter the command, or cancel the command with ^Z. (Note: SuperCalc caught the error before attempting to execute the command.)

Window Parameter ERROR

This occurs during the window command when you attempt to split the screen when the active cell is at the left or right edge or the top or bottom row of the display screen. Because of the way that the command works, the split cannot take place at the edges of the screen.

Either move the active cell away from the edge of the display window or scroll the screen to provide an additional column or row between the edge and the location you desire for the split.

Worksheet FULL

The worksheet is too large in size; there are too many cell stubs. (This is different from the case described above in Memory Full, where the worksheet has too much content.)

If you can, blank any unnecessary contents and move the other contents to the upper left, trying to preserve a roughly rectangular shape. Then save the worksheet, ZAP the screen, and reload.

"Memory Use—Hints and Concepts" in the SuperCalc reference section explains how it is possible to unintentionally create many more cell stubs than necessary. You may get a Worksheet Full message even though you have few contents and they are at the upper left. In such a case, saving the worksheet, ZAPping the screen, and reloading the worksheet will get rid of unnecessary cell stubs.

CBASIC Error Messages

CBASIC COMPILER ERRORS:

The following compiler error messages can appear during compilation of a source file:

NO SOURCE FILE: < filename > .BAS

The source file could not be found on the indicated drive.

OUT OF DISK SPACE

The compiler encountered insufficient disk space while writing the .INT or .LST file.

OUT OF DIRECTORY SPACE

The compiler ran out of directory entries while attempting to create or extend an .INT or .LST file.

BDOS ERROR ON (A,B)

This CP/M error message indicates that an error occurred while the computer was reading from or writing to a disk file.

PROGRAM CONTAINS *n* UNMATCHED FOR STATEMENT(S)

n FOR statements have no associated NEXT statements.

PROGRAM CONTAINS *n* UNMATCHED WHILE STATEMENT(S)

n WHILE statements have no associated WEND statements.

PROGRAM CONTAINS 1 UNMATCHED DEF STATEMENT

A multiple line function was not terminated with a FEND statement, possibly causing further errors.

WARNING INVALID CHARACTER IGNORED

An invalid character was detected in the last line, then was replaced by a question mark and ignored.

INCLUDE NESTING TOO DEEP NEAR LINE *n*

An INCLUDE statement exceeded the maximum nesting level near line *n*.

COMPILER ERROR CODES:

The following two-letter error codes display with the line number and position of the error:

BF

Invalid branch into a multiple line from outside of the function.

BN

Invalid numeric constant was encountered.

CI

Improper filename used in an %INCLUDE directive.

CS

The COMMON statement was not the first program statement preceded only by a directive, remark, or blank line.

CV

A subscripted variable in a COMMON statement was not properly defined.

DL

Duplication of the same line number, an undefined function, or a DIM statement that does not precede all referenced arrays, was detected.

DP

A DIM variable was previously defined in another DIM statement or was used as a simple variable.

FA

A function name not used in the function was encountered to the left of the equal sign in an assignment statement.

FD

A function name is the same in two DEF statements.

FE

An incorrect mixed-mode expression exists in a FOR statement, usually the expression following TO is involved.

FI

The FOR loop index is not an unsubscripted-numeric variable expression.

FN

An incorrect number of parameters are used in the function reference.

FP

The function-reference parameter type does not match that in the DEF statement.

FU

An undefined function has been referenced.

IE

The IF statement expression is erroneously evaluated as type string.

IF

The FILE statement variable is type numeric instead of type string.

IP

An input prompt string was not enclosed in quotes.

IS

A subscripted variable was not dimensioned before it was referenced.

IT

Indicates that an invalid compiler directive was issued.

IU

A DEF-statement defined array was not subscripted.

MC

A variable was defined more than once in a COMMON statement.

MF

The expression is evaluated as type string instead of type numeric.

MM

An invalid mixed mode was encountered, usually caused by a mixture of string and numeric types in an expression.

MS

A numeric instead of a string expression was used.

ND

A DEF statement could not be found for a corresponding FEND statement.

NI

A NEXT variable reference did not match that referenced by the associated FOR statement.

NU

A NEXT statement occurred without an associated FOR statement.

OF

An illegal branch from within a line function was attempted.

OO

The ON statement limit of 40 was exceeded.

PM

A DEF statement was encountered within a multiple line function. Functions cannot be nested.

SE

A syntax error occurred in the source line, usually as the result of an improperly formed statement or misspelled keyword.

SF

A numeric instead of a string expression was used in a SAVEMEM statement. Check for quotes around string constants.

SN

An incorrect number of subscripts were found in a subscripted variable, or a DIM variable was previously used with a different number of dimensions.

SO

A statement that is too complex should be simplified in order to be compiled.

TO

Indicates a symbol table overflow, meaning the program is too large for the current Osborne 1 memory configuration.

UL

You have referenced a nonexistent line number.

US

A string was terminated with a carriage return, rather than quotes.

VO

Variable names are too long for one statement.

WE

The expression following the WHILE statement is not numeric.

WN

The nesting level of WHILE statements (12) has been exceeded.

WU

A WEND without an associated WHILE statement was encountered.

RUN-TIME ERRORS:

The following run-time error messages are displayed below the most recent screen line, to indicate conditions which usually terminate program execution.

NO INTERMEDIATE FILE

A file name of type .INT could not be located on the specified drive.

IMPROPER INPUT-REENTER

The fields you entered at the keyboard do not match those specified in the INPUT statement.

WARNING CODES:

Two-letter codes preceded by the word WARNING indicate errors that do not prevent execution of a program but should be attended to. These codes are:

DZ

A number divided by zero resulted in the largest CBASIC number.

FL

A field length greater than 255 bytes was encountered during a READ LINE; the remainder is ignored.

LN

A LOG function argument was zero or negative; the value of the argument is returned.

NE

A negative number before the raise to a power operator (^) was encountered, resulting in the absolute value of the parameter being calculated.

OF

A real-variable calculation produced an overflow. The result is set to the largest valid CBASIC real number. Overflow is not detected with integer arithmetic.

SQ

A negative number was specified in the SQR function. The absolute value is used.

RUN-TIME ERROR CODES:

The following two-letter codes are preceded by the word ERROR and cause execution to terminate:

AC

An ASC function string argument was evaluated as a null string.

BN

The BUFF value in either the OPEN or CREATE statement is less than 1 or greater than 52.

CC

The CHAINed program code area is greater than the calling program's code area. Use %CHAIN for adjustment.

CD

The CHAINed program data area is greater than the calling program's data area. Use %CHAIN for adjustment.

CE

The file being closed could not be found in the directory.

CF

The CHAINed program constant area is greater than the calling program's constant area. Use %CHAIN for adjustment.

CP

The CHAINed program variable storage area is greater than the calling program's variable storage area. Use %CHAIN.

CS

The CHAINed program reserved a different amount of memory with a SAVEMEM statement than the calling program.

CU

An inactive file number was specified in the CLOSE statement.

DF

An already active file number was specified in an OPEN or CREATE statement.

DU

An inactive file number was specified by a DELETE statement.

DW

Indicates a write to a file for which no IF END statement has been executed; may occur if the disk directory is full.

EF

Indicates a read past an end of file for which no IF END statement has been executed.

ER

A write to a record whose length exceeds the maximum record length specified by an OPEN, CREATE, or FILE statement was attempted.

FR

The renamed file name already exists.

FU

A read or write operation to an inactive file was attempted.

IF

An invalid file name was specified.

IR

A record number of zero was specified.

IV

Execution of an INT file created by a version 1 compiler was attempted. Recompile using version 2 compiler.

IX

A FEND statement was encountered before execution of a RETURN statement.

ME

A full directory resulted in an error while you were creating or extending a file.

MP

A third MATCH function parameter was zero or negative.

NF

A file number less than 1 or greater than 20 was specified, or a file statement was executed when 20 files were already active.

NM

Not enough memory was available to load the program.

NN

A PRINT-USING statement could not print a number because no numeric data field could be found in the USING string.

OD

A READ statement was executed with no corresponding data.

OE

Invalid execution of an OPEN statement for a nonexistent file when no prior IF END statement had executed.

OI

An ON GOSUB expression, or an ON GOTO statement was evaluated as a number less than 1, or greater than the number of line numbers in the statement.

OM

Current program exceeded available memory. Close unneeded opened files, nullify unused strings, and read data from a disk file.

QE

A PRINT string contained a quotation mark and could not be written to the specified file.

RB

Attempted random access to a file activated with BUFF where more than one buffer was specified.

RE

Attempted read past the end of a record in a fixed file.

RG

A RETURN was issued for which there was no associated GOSUB statement.

RU

A random read or print to a file that was not fixed was attempted.

SB

An ARRAY subscript exceeded the defined boundaries.

SL

A string longer than 255 bytes resulted from a concatenation operation.

SO

The file specified in SAVEMEM was not on the indicated disk.

SS

The second parameter in the MID$ function, or the last parameter in the LEFT$ or RIGHT$ was negative or zero.

TL

The TAB statement parameter was less than 1 or greater than the current line width.

UN

A PRINT USING statement contained a null edit string, or an escape character (\) was the last in an edit string.

WR

An attempt was made to write to a file after it had been read but before it had been read to the end of the file.

MBASIC Error Messages

1:NF

NEXT without FOR

A NEXT statement variable was encountered without a corresponding FOR statement variable.

2:SN

Syntax Error

An incorrect sequence of characters was encountered.

3:RG

Return without GOSUB

A RETURN statement was encountered for which no previously executed GOSUB could be matched.

4:OD

Out of data

Indicates that a READ statement was executed for which no unread DATA statements could be found.

5:FC

Illegal function call

possibly as the result of:

a) Too large or negative subscript
b) Negative or zero LOG argument
c) Negative argument to SQR

d) Negative mantissa with noninteger exponent
e) Illegal call to USR function with no defined starting address
f) Improper argument to MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, TAB, SPC, STRING$, SPACE$, INSTR, ON . . . GOTO

6:OV

Overflow

Result of calculation exceeds upper limit of MBASIC 80 number format and thus cannot be displayed.

7:OM

Out of memory

Indicates that the program is too long, or has too many loops, variables, or expressions.

8:UL

Undefined line

A nonexistent line number was referenced in a GOTO, GOSUB, IF . . . THEN . . . ELSE, or DELETE statement.

9:BS

Subscript out of range

The array element being referenced is outside the array dimensions or has the wrong number of subscripts.

10:DD

Redimensioned array

Indicates that more than one DIM statement was given for the same array. May also be caused when a DIM statement is given for an array after the default dimension of 10 has already been established.

11:/O

Division by zero

Division by zero was encountered in an expression, or the operation of involution resulted in zero being raised to a negative power.

12:ID

Illegal direct

Execution of a statement that is illegal in direct mode was attempted.

13:TM

Type mismatch

A variable name of the wrong type was assigned, or a function was given the wrong type argument.

14:OS

Out of string space

The remaining free memory has been exceeded because of string variables.

15:LS

String too long

An attempt to create a string longer than the maximum length of 255 characters was made.

16:ST

String formula too complex

An expression that is too long or too complicated to process was encountered.

17:CN

Cannot continue

Indicates an unsuccessful attempt to continue a program that was halted because of an error or was modified during a break in execution, or a program that does not even exist.

18:UF

Undefined user function

An undefined USR function is being referenced without being defined by a DEF statement.

19

No RESUME

No RESUME statement has been specified for the error-trapping routine in progress.

20

RESUME without error

A RESUME statement has been encountered before an error-trapping routine has been entered.

21

Unprintable error

An error for which no message has been defined has been encountered.

22

Missing operand

An expression containing an operator was encountered that had no operand following it.

23

Line buffer overflow

Indicates that the line being input contains too many characters.

26

FOR without NEXT

A FOR statement for which no corresponding NEXT statement has been specified has been encountered.

29

WHILE without WEND

A WHILE statement for which no corresponding WEND statement has been specified has been encountered.

30

WEND without WHILE

A WEND statement for which no corresponding WHILE statement has been specified has been encountered.

50

Field overflow

The record length allocated by a FIELD statement for a random file is being exceeded.

51

Internal error

An internal malfunction has occurred; report this error to your dealer.

52

Bad file number

The file number being used to reference a file is either out of the range specified at initialization or is a file number that has not been OPENed.

53

File not found

The file being referenced by a LOAD, KILL, or OPEN statement does not exist on the currently logged drive.

54

Bad file mode

An illegal attempt has been made to LOAD a sequential file using PUT, GET, or LOF. Can also be caused by execution of an OPEN statement for which a file mode other than I, O, or R has been specified.

55

File already open

A sequential output mode OPEN statement is issued for an already open file. Can also be caused by a KILL statement issued for an open file.

57

Disk I/O error

A fatal error occurred during an input or output operation.

58

File already exists

The file name supplied in the NAME statement already exists on the specified drive.

61

Disk full

The message shows that there is no space left on the currently logged drive.

62

Input past end

Indicates that an INPUT statement was issued after all the data in the file had been used, or for a null file.

NOTE

Use the EOF function to detect the end of file.

63

Bad record number

The record number specified in the PUT or GET statement is either greater than the maximum allowed (32767) or equal to zero.

64

Bad file name

An invalid file name was used in a LOAD, SAVE, KILL, or OPEN statement.

66

Direct statement in file

A direct statement used to load an ASCII-format file caused execution of the LOAD statement to be terminated.

67

Too many files

Attempt to create a new file using SAVE or OPEN cannot be accomplished because all 255 directory entries are full.

APPENDIX 4

Single- & Double-Density Differences

Technical Specifications

The Osborne 1 double-density can read data from and write data to the following diskette formats:

- Osborne single-density
- Osborne double-density
- Xerox 820 single-density
- Cromemco single-density
- IBM Personal Computer using CP/M-86
- DEC VT-180

Exact technical specifications for these formats are as follows:

	OSBORNE		ALTERNATE		
	Single-Density	**Double-Density**	**Xerox & Cromemco**	**IBM CP/M-86**	**DEC**
tracks per diskette	40	40	40	40	40
physical sectors per track	10	5	18	8	9
logical sectors per track	20	40	18	32	36
bytes per sector	256	1024	128	512	512
total diskette capacity	100K	200K	90K	160K	180K
reserved tracks	3	3	3	1	2
directory track	4	4	3	1	2
data storage capacity	92K	185K	74K	156K	171K
CCP starting location	0CF00h	0CB00h			
BIOS starting location	0E500h	0E100h			
sector skew factor	2	1	5	1	2

Using Different Diskette Formats

The double-density Osborne, when started (e.g., "booted") using double-density Osborne diskettes, recognizes, reads data from, and writes data to any of the diskette formats listed in the chart above, if those diskettes are placed in the B drive. Some programs on the alternate format or Osborne single-density diskettes are also usable.

The Osborne can only write to the format of the alternate format diskette that is actually present in the computer; it will not create an alternate diskette format on its own (i.e., you cannot use the Osborne FORMAT program to create alternate format diskettes).

If an Osborne **single**-density diskette is used to start a double-density Osborne, the computer will then act like a single-density Osborne; in this case, the computer can only recognize single-density Osborne diskettes, just as if it were a single-density Osborne.

There is a way, however, to start the computer using single-density Osborne diskettes or the alternate format diskettes. This is done by copying double-density system tracks onto the alternate format or Osborne single-density format diskettes, by using the CP/M SYSGEN utility as follows:

> Put your (double-density) CP/M diskette into drive A and boot. Leave HELP by pressing **ESC**. Type **SYSGEN** and press **RETURN**.
>
> Put your Osborne single-density or alternate format diskettes in drive B.
>
> Respond to SOURCE? by typing the letter **A**. Respond to DESTINATION? by typing the letter **B**.

The alternate diskette can now be used as the boot diskette in the same manner as a double-density diskette. This operation does not alter the data on the diskette. It will, however, prevent the Osborne single-density or alternate format diskettes from being used in their original machines. The converted Osborne single-density diskettes can be converted back by using the SYSGEN utility to copy the system tracks from a single-density diskette. Alternate format diskettes might possibly be likewise reconverted by using the system generation facilities of their original machines; see their reference manuals for this information.

Changing Single-Density Diskettes to Double-Density

Diskettes included with the Osborne 1 double-density computer are all double-density format and come with double-density SYSGEN. Osborne-approved software, however, is sold in single-density format and with no operating system, so that it can be used by either single- or double-density Osbornes. (The Extended Utilities diskette provided with the double-density upgrade is also single-density format; however it will boot double-density as if it were a double-density diskette.)

You can use single-density working copies of such software in a double-density Osborne by putting double-density SYSGEN on them. Or you can convert them to double-density in this manner:

> Boot up the double-density CP/M diskette in drive A, or your Extended Utilities diskette, in the case of a double-density upgrade. Press **ESC** to leave HELP.
>
> Place a blank diskette in drive B. Format this diskette in DOUBLE-DENSITY as described in chapter 3.
>
> Type the word **PIP** and press **RETURN**. The PIP program loads and an asterisk (*) appears on the screen.
>
> Remove the CP/M diskette from the A drive. Replace it with the single-density program diskette you want to copy.
>
> Now type the following, **exactly** as shown:

```
B:=A:*.*
```

> and press **RETURN**. Make sure you have typed it exactly as shown.

The screen displays the files as they are transferred from the A disk to the B disk. When the transfer is completed, the asterisk reappears on the screen. The double-density diskette in drive B is now a *double-density copy* of the single-density diskette in drive A.

Now place your double-density CP/M (or Extended Utilities) diskette back in drive A, press **RESET**, and then press **RETURN**. Then follow the SYSGEN procedure described earlier to copy system tracks onto your double-density diskette.

You may use your double-density CP/M (or Extended Utilities) diskette to make a COPY of this diskette, and the copy will also be double-density.

Use this same PIP procedure to consolidate several programs onto one double-density diskette. Check that the recipient diskette has room for the programs. You may also PIP selected files, as described in chapter 3 of the User's Guide or in the CP/M section of the Reference Guide. PIP can also transfer the other way, from double- to single-density. In each case, double-density must be "booted" (used to start the computer).