commodore plus/4

User's Manual

WORD PROCESSING

SPREAD SHEETS

GRAPHICS

DATA MANAGEMENT

commodore plus/4

USER'S MANUAL STATEMENT

"This equipment generates and uses radio frequency energy. If it is not properly installed and used in strict accordance with the manufacturer's instructions, this equipment may interfere with radio and television reception. This machine has been tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of the FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. If you suspect interference, you can test this equipment by turning it off and on. If you determine that there is interference with radio or television reception, try one or more of the following measures to correct it:

- reorient the receiving antenna
- move the computer away from the receiver
- change the relative positions of the computer equipment and the receiver
- plug the computer into a different outlet so that the computer and the receiver are on different branch circuits

If necessary, consult your Commodore dealer or an experienced radio/television technician for additional suggestions. You may also wish to consult the following booklet, which was prepared by the Federal Communications Commission:

"How to Identify and Resolve Radio-TV Interference Problems." This booklet is available from the U.S. Government Printing Office, Washington, D.C. 20402, Stock No. 004-000-00345-4."

You should use only the cables, accessories, and peripherals recommended by Commodore for your Plus/4. All cables, including the cables for the television hookup, serial port, video port, Datassette™, and joysticks, are specially shielded, in accordance with the regulations of the Federal Communications Commission. Failure to use the appropriate accessories and cables will invalidate the FCC grant of certification, and may cause harmful radio interference.

COMMODORE PLUS/4
USER MANUAL

INTRODUCTION

You've made a wise purchase... the Plus/4 is the first home computer ever designed especially for productivity applications. Of course, it's still able to do all the other things a home computer can be used for. This manual is designed to help you learn those "other things" your Plus/4 can do. You'll learn how to:

- Set up your Plus/4
- Use the different functions of all the keys on the keyboard
- Access different types of Commodore Software
- Use the mathematical, graphics, sound and programming capabilities of your Plus/4

The other manual included with your computer (The Plus/4 Built-in Integrated Software Manual) tells you how to use the wordprocessing, electronic spreadsheet, database and graphics packages. If your main interest is in these productivity applications, and you can't wait to get started using them, we still recommend that you read through at least chapter one of this manual ("Unpacking and Setting Up") before reviewing the Built-in Software Manual.

WHAT'S SPECIAL ABOUT THE PLUS/4

- 64K RAM (60K available for BASIC programming)
- Full Typewriter Style Keyboard
- Optional Built-In Software
- Screen Window Capability
- HELP Key
- 8 Pre-programmed, Reprogrammable Function Keys
- Four Separate Cursor Keys
- Uses Most COMMODORE 64 and VIC-20 Peripherals
- 121 Colors (16 primary colors, 8 luminance levels)
- Over 75 BASIC Commands
- High Resolution Graphics Plotting
- Split-Screen Text With High-Res Graphics
- Graphic Character Set On Keyboard
- Keyboard Color Controls
- 320 × 200 Pixel Screen Resolution
- Reverse and Flashing Characters
- 2 Tone Sound Generators
- Built-In Machine Language Monitor (17 commands)

CREATING A COMPLETE COMPUTER SYSTEM

Computer: Commodore PLUS/4
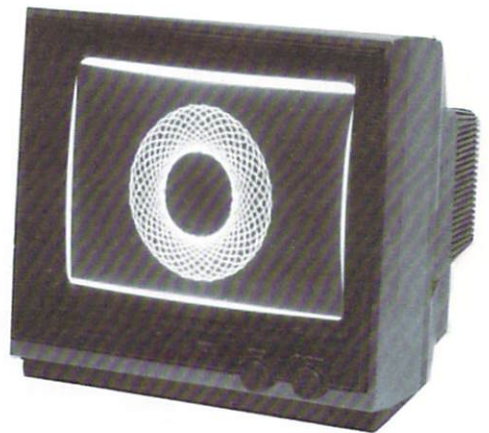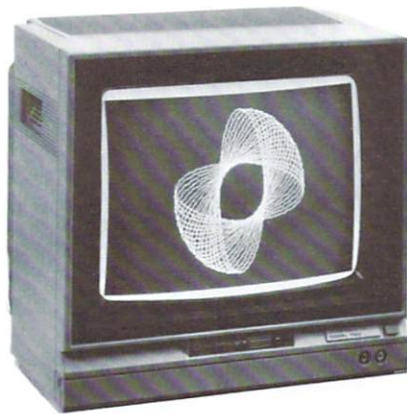
Display: Commodore 1702 or 1802/1803 Color Monitor

Storage: Commodore 1531 Datassette (tape recorder) or Commodore Disk Drive

Printer: Commodore Printer

Modem: Commodore
MODEM/300

Controller: Commodore PLUS/4 Joysticks

WHERE TO FROM HERE? By now you've done enough reading and you want to get started. Here's what we recommend you do now:

- Send in your warranty card
- Subscribe to the Commodore magazines to get the latest information on your computer

Read this manual and try the exercises. Read the built-in software manual and get used to the four integrated packages. Keep checking in with the Commodore dealers in your area for new developments in software, books and peripherals. Learn, program, file, write, calculate, graph, play... enjoy your new Commodore Plus/4!

CHAPTER 1
UNPACKING AND SETTING UP

- Unpacking your Commodore Plus/4

- Getting to know the switches and sockets

- Setting up your Commodore Plus/4

- Troubleshooting chart

- Peripherals

Now that you've opened the box containing your new Plus/4 and found this manual, the first thing that you should do is check to make sure that you have all the items on this list. You should have:

1. Your Commodore Plus/4
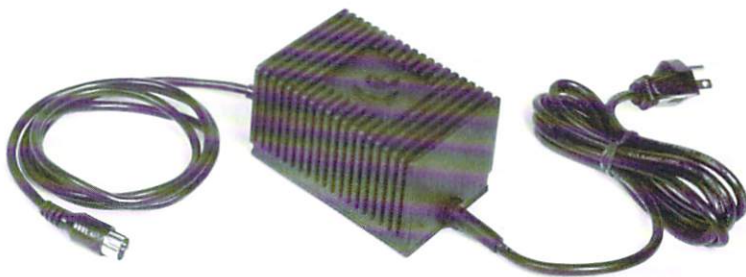
2. The power supply
   One end plugs into a wall outlet, the other plugs into the right side of the computer.
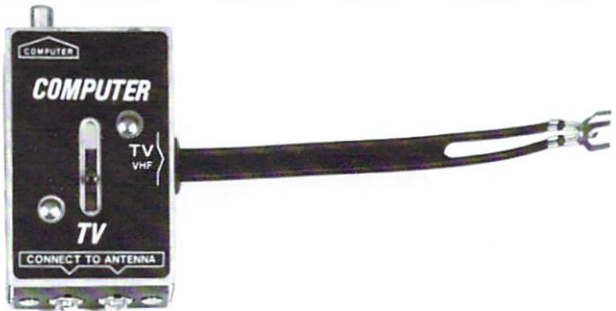
3. The TV switchbox
   This connects to the antenna jack on the back of your TV. You don't need the switchbox if you plan to connect your Plus/4 to a monitor.

4. The RF cable
   This connects the TV switchbox to the RF output jack on the left side of the Plus/4. You don't need this cable to connect your Plus/4 to a monitor.

5. The user manual

6. Other assorted literature
   Warranty card
   Commodore Magazines subscription card

7. The Plus/4 Built-in Integrated Software Manual

If you don't find all these items in the box, check with your dealer immediately for replacements.

Before you connect anything, you should look over these drawings of your computer. These drawings identify all the outlets so you can set up your computer system quickly and easily.

**GETTING
TO
KNOW
THE
SWITCHES
AND
SOCKETS**
The Right
Side
of Your
PLUS/4

1 **The On/Off Switch**
Your Plus/4 should be turned OFF when you install or remove car-
tridges or any peripheral device such as a printer or disk drive. There
is a red power light located below the keyboard on the left, so you
can be sure whether power is off.

2 **The Reset Button**
There are two ways to use the RESET button:

1. You can use the RESET button to reset your computer as if you'd
just turned it on. Just press the reset button once. Remember: when
you press the reset button, you lose any BASIC program currently
in memory.*

2. If you want to reset your Plus/4 and keep your BASIC program,
hold down the RUN/STOP key and then press the RESET button. When
you do this, your Plus/4 goes to the built-in machine language monitor.
Type an X and press the RETURN key to get back to BASIC. Your pro-
gram remains intact in the Plus/4 memory. Just type LIST to display the
program on your screen.

*When you press RESET, the Plus/4 automatically issues the NEW
command, which clears the screen. This can be reversed. See the
Plus/4 Programmer's Reference Guide for information on UNNEWing
your program if you've pressed the reset button by accident.

The Left Side Of Your PLUS/4
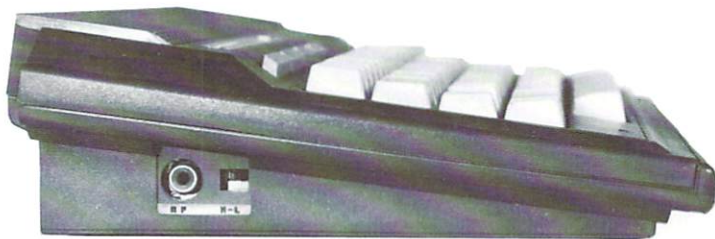
3   4

The socket and the switch on the left side of the Plus/4 are both used for TV connections. Neither is used if you're connecting your Plus/4 to a monitor.

3 The RF Jack
This is where you plug in one end of the RF cable (the thin black cable). You can plug either end into this jack and the other end into the TV switchbox.

4 The High/Low Switch
This switch controls which channel is used for Plus/4 video output. Set the H/L switch to L for output on channel 3. Set the H/L switch to H for video on channel 4.

You can use either channel 3 or 4 on your TV to display the video picture from your computer. If you have a channel 3 TV station in your area, select channel 4, and vice versa. Experiment to see which setting gives you the best picture.

The Back of Your Computer
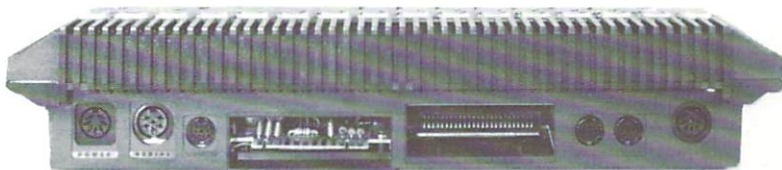
5   6   7   8   9   10   11

The sockets on the back of your computer connect a variety of accessories to your Plus/4. Each connector is different. Be sure you plug each accessory into the correct socket.

5 The Power Socket
The end of the power supply cable fits here. Plug the other end into a standard wall socket for three-prong plugs.

6 The Serial Bus
You can plug a disk drive or a printer into this socket. If you want to plug in both, first plug the disk drive into this opening, then plug the printer cable into the back of the disk drive.

7 The Cassette Port
The Commodore 1531 Datassette tape recorder plugs in here.

8 The RS-232 Port
Accessories such as a modem or an RS-232 adapter plug in here. An RS-232 adapter makes it possible to hook up accessories not accommodated by standard Commodore equipment ports.

9 The Memory Expansion Port
Plus/4 software cartridges and the Plus/4 SFS-481 disk drive plug in here. Before you install or remove cartridges, make sure your Plus/4 is OFF.

10 Joy 1 and Joy 2 : The Game Ports
You can plug joysticks into these sockets. The Plus/4 uses specially designed joysticks available from your Commodore dealer.

11 The Video Socket
This is where you plug in the cable that connects a monitor to your Plus/4. Although this socket is an 8-pin connector, you can use a 5-pin cable in this socket as well. Commodore color monitors come with an 8-pin cable for use with the Plus/4.

SETTING UP YOUR PLUS/4

- To set up your Plus/4, you'll need at least two wall plugs: one for your Plus/4 and one for your TV or monitor.

- If you're installing a disk drive and a printer, you'll need additional wall plugs.

- Your Plus/4 should be placed a comfortable distance from your TV.

- Make sure that your computer is OFF before you start the setup. Check that the POWER LIGHT on the front left is not lit.

If you are connecting the Plus/4 to a television set, you'll need a small screwdriver to attach the TV switchbox. The way you connect the switchbox depends on what type of antenna connection your TV set has.

IMPORTANT: If your antenna is connected to your TV by a single round-ended cable (the 75-ohm co-ax type), you will need either the 300 ohm to 75 ohm adapter, which came with your TV, or you must get a replacement 75 ohm to 75 ohm switchbox. The adapter is a small plastic part with a co-ax connector on one side and two screws on the other. If you do not have one, you can buy one at most electronics stores. Once you attach the adapter to the co-ax connector on your set, you can follow the rest of these instructions. A 75-ohm switchbox allows you to hook the antenna lead into the switchbox, which is connected to the TV, so that you only have to move the switch on the switchbox to watch TV.

You need only connect the switchbox once. When you want to use your computer, just move the switch to the COMPUTER position. When you want to watch TV, move the switch to TV. The switchbox will not interfere with your TV reception.

STEP 1. Disconnect the antenna from your TV: use a screwdriver to loosen the screws on the TV. Remove the two antenna leads.

STEP 2. Connect the TV switchbox to the TV where the antenna leads were: attach the leads on the box to the antenna input on your TV.

STEP 3. Connect the antenna to the switchbox: attach the leads from the antenna to the screws on the side of the switchbox.
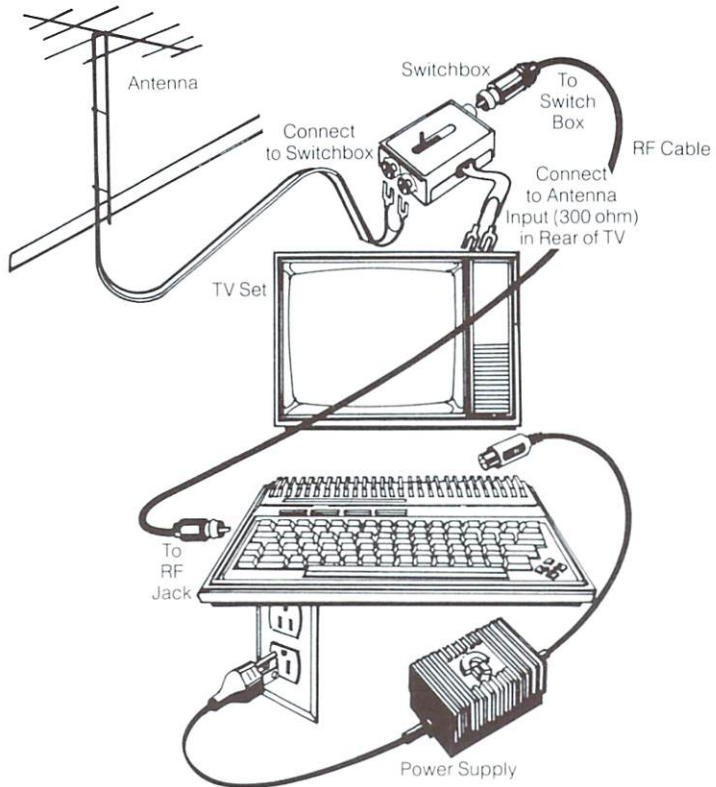
If you have the round coax type antenna connection on your TV:

STEP 1. Disconnect the antenna from your TV: unscrew the antenna wire. You can disconnect it by hand.

STEP 2. Connect the switchbox to your TV: hand fasten it onto the antenna input post on the back of your TV.

STEP 3. (For the 75-ohm switchbox) Connect the antenna to the switchbox, hand-turning the antenna cable into the switchbox.

Back of TV

If an Adapter is Necessary

300 ohm to 75 ohm Adapter

Remove to Connect Antenna Lead to TV

Antenna Lead
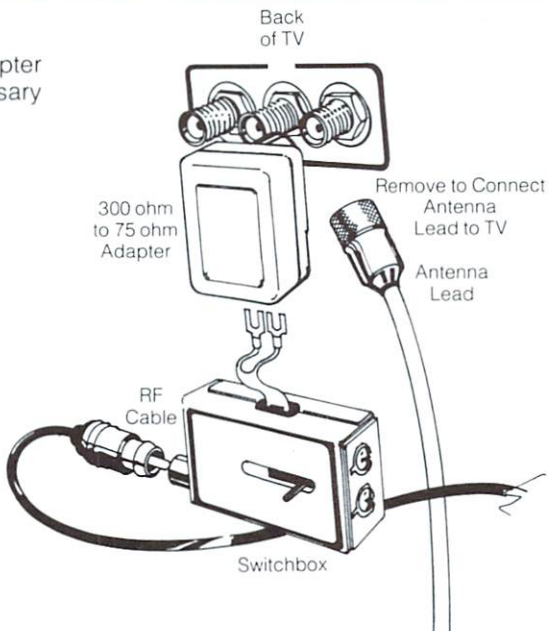
RF Cable

Switchbox

Once the switchbox is in place, get the RF cable (the thin cable with connectors on both ends) that came with your Plus/4. Plug one end into the socket on the side of the switchbox. Connect the other end into the socket marked RF on the left side of your computer.

NOTE: If you're using the 300- to 75-ohm adaptor and you want to watch TV, you must disconnect the switchbox and plug the antenna lead back in. This is easily done by hand, and may be done as often as you like, with no damage to your TV, computer, or antenna.

Selecting A Channel On Your TV

As we explained earlier, your TV should be set on either channel 3 or 4 when you are using your computer. Don't choose a channel that broadcasts in your area. If you use channel 3, set the H/L switch on the side of the computer to L. If you use channel 4, set this switch to H.

Connecting Your Commodore PLUS/4 to a Monitor

If you're connecting your computer to a monitor instead of a TV, follow the instructions in the manual that is included with the monitor. Hooking up a monitor, like the Commodore 1702 Color Monitor, is simple. It requires only one cable that connects directly from your monitor to the VIDEO socket in the back of your computer.

Final Steps

1. Attach the power supply cable from the power box to your Plus/4. Plug the round end of the cable into the POWER socket on the back of the computer; plug the power supply into the wall socket.

2. If you are using a TV, make sure that the setting on the H/L modulator and the channel on your TV are in agreement. (If your computer is set at L, the TV must be on channel 3; the TV should be tuned to channel 4 with your computer at H.) Make sure that the switchbox is set to the COMPUTER setting.

   If you are using a Commodore color monitor, use the rear jacks, and check that the back/front switch is set to back.

3. Turn on your computer. (The switch is on the right side as you face the Plus/4.)

If all is well, this message appears on your screen:

COMMODORE BASIC 3.5 60671 BYTES FREE

READY.

■

The flashing cursor under the READY message tells you that the Plus/4 is waiting for you to start typing. The background color is white, while the letters are printed in black, with a light purple border around the screen.

4. Check the troubleshooting chart if you have problems. You may need to adjust your TV set to get a sharper picture.

Symptom	Cause	Remedy
Indicator light not 'ON'	Computer not turned ON	Make sure power switch is in ON position
	Power cable not plugged in	Check power socket for loose or disconnected power cable
	Power supply not plugged in	Check connection with wall outlet
	Bad fuse in computer	Take system to authorized dealer for replacement of fuse
No picture	TV on wrong channel	Check other channel for picture (3 or 4)
	Incorrect hookup	Computer hooks up to VHF antenna terminals
	Video cable not plugged in	Check TV output cable connection
	Computer set for wrong channel	Set computer for same channel as TV
	Switchbox not set to computer	Check that switchbox is in 'computer' position
	TV not on	Turn TV on
Random pattern on TV with cartridge in place	Cartridge not properly inserted	Reinsert cartridge after turning OFF power
Picture without color	Poorly tuned TV	Retune TV
	TV not connected	Check connections properly

Picture OK, but no sound	TV volume too low	Adjust volume of TV
	Poorly tuned TV	Retune TV
	Auxiliary output not properly connected	Check connection as shown on diagram on page 14-15

IMPORTANT: Some TV sets cannot display the entire Plus/4 screen. Instead, their picture cuts off the far left and far right column of the Plus/4 screen display. We recommend using a different TV set or a monitor such as the Commodore 1702 or 1802/1803 color monitor.

If this is not possible, you can deal with the problem by pressing the ESCape key, followed by the 'R' key. This reduces the computer screen display size so the entire picture can fit onscreen. You must repeat this each time you power up or reset your Plus/4.

PERIPHERALS Peripherals are the accessories that you can add to your Plus/4 system. These accessories are available at your Commodore dealer, and allow you to use the Plus/4 to the fullest. Peripherals give your Plus/4 system the capability to store and save data, print hard copy (in black and white or color), use disk and cassette-based software, and access the information and services available through telecommunications.

To save or recall programs, you'll need a device that stores data. Data can be stored on and retrieved from both diskettes and cassette tapes. To use diskettes, you'll need a DISK DRIVE. Disk drives are typically fast and efficient to use. Disk drives that are compatible with your Plus/4 are the Commodore models 1541 and 1551. For cassette-based storage and retrieval, the Commodore 1531 DATASSETTE tape recorder fills the bill.
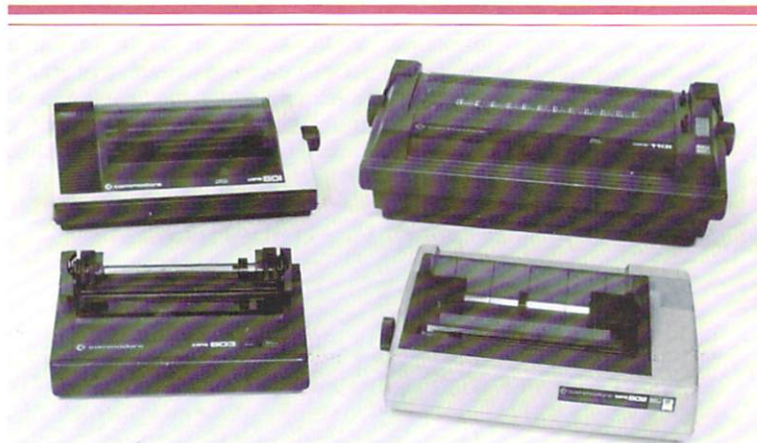
When using a wordprocessing program or a graphics package on the Plus/4, a printer will reproduce what is on the screen on paper. There are several models of Commodore printers available that work with the Plus/4. These include the MCS-801, MPS-802, MPS-803 (with tractor-feed) and DPS-1101 (letter quality). Different printers specialize in different types of print-outs. Ask your dealer which best suits your needs.
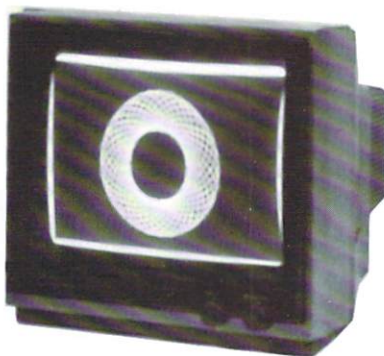
Your television may not give you as clear a picture as you'd like for your computer. Commodore color monitors are specially designed to give you the sharpest, brightest picture for your Plus/4 screen output. There are several Commodore monitor models available, including the 1702, and 1802/1803.

There are networks that you can reach over the phone whose purpose is to provide information, programs, news, stock market reports, entertainment, and almost anything else you could think of to computerists by using the phone lines. To gain access to the tremendous range of services, software, and information available, you must be equipped with a MODEM.

The Commodore Plus/4MODEM connects your Plus/4 to these information services over telephone lines. With one of these modems, you can have access to computer services such as CompuServe and The Source. Commodore supports its own service called the Commodore Information Network, which is available through CompuServe. The Commodore Information Network (CIN) specializes in information for the Commodore owner, including current dealer lists, Commodore hardware and software tips, and a direct line to Commodore Customer Support. A wide assortment of programs is available on the Commodore User Database segment of CIN, including graphics, music, educational programs, utilities and games.

CHAPTER 2

USING THE KEYBOARD AND THE SCREEN

- A tour of the keyboard

- Special keys

- Graphic keys

- Programmable function keys

- The HELP key

A
TOUR
OF
THE
KEYBOARD

Most of the keys on the Plus/4 keyboard are identical to the keys on a typewriter, but each key can do more than a typewriter key. In this section, you'll learn how to use special keys like the ⊂ C= ⊃ key and the cursor arrow keys. This section will show you the extra features of every key, including how to use the graphic symbols pictured on the fronts of many of the keys.

While we guide you on the tour of the Plus/4 keyboard, you should find the keys and practice using them.

RETURN

You have to press the **RETURN** key at the end of each line of instructions you enter on your Commodore Plus/4 keyboard. You might think of this key as an ENTER key because **RETURN** actually enters information and instructions into the computer.

SHIFT

This key works like the shift key on a regular typewriter. Your Plus/4 has two **SHIFT** keys and a **SHIFT LOCK**, which works like the shift lock on a typewriter.

By pressing the **SHIFT** key, you can get the graphic symbol on the right side on each graphics key when you are in upper-case/ graphics mode.

Your Plus/4 is automatically in upper-case/graphics mode when you turn it on. In upper-case/graphics mode, all the letters appear upper-case without the **SHIFT** key.

The **SHIFT** key pressed with a letter key gets upper-case (capital) letters when you are in upper/lower-case text mode (the same as the **SHIFT** key on a typewriter). When in this mode, the letters you type are in lower-case except when you use the **SHIFT** key.

NOTE: You can go back and forth between upper-case/graphics and upper/lower-case text modes by pressing the **SHIFT** and **C⁼** key at the same time.

RUN/STOP

Press this key to break into a running program to STOP what your Plus/4 is doing. When the Plus/4 is running a program, pressing this key returns control back to you and the keyboard.

When you hold down the **SHIFT** and **RUN/STOP** keys simultaneously, the Plus/4 loads and runs the first program on a disk in the disk drive.

The Cursor Keys

It's easy to move the cursor quickly around the screen in any direction. Just press the cursor arrow key that points in the direction you want to go. Like all keys on the Plus/4 keyboard, each cursor key can repeat indefinitely while the key is held down. This automatic repeat function keeps the cursor moving until you release the key.

NOTE: You can move the cursor over letters and numbers on the screen without affecting those characters.

INST/DEL

You can INSERT and DELETE letters and numbers from the line you are typing by pressing this key. When you press this key by itself, that character immediately to the left of the cursor disappears, and the cursor moves over to where the missing character was. You can use the cursor keys to go back to the middle of a line and then use **DEL** to DELete a letter. When you do this, the letter to the left is deleted, and the rest of the letters on the line move over one space to the left to close the gap.

You can open up space to insert letters and numbers by using the **SHIFT** and **INST** keys. Space opens to the right of the cursor; the cursor itself does not move. When you insert space in the middle of a line of letters, the rest of the line moves to the right.

The **INST/DEL** key saves a lot of time when you want to edit or change what you've typed.

CLR/HOME

This key serves three functions: HOME, CLEAR, and CLEAR WINDOWS. When you press this key, the cursor immediately moves to the top left corner of the screen. This is called the HOME position. The rest of your screen stays the same. If you hold down the **SHIFT** key and press **CLR/HOME** , not only does the cursor move to HOME, but the screen clears. All that remains on the screen is the blinking cursor at the top left corner of the screen. If you press this key twice,

any screen windows that you have set up are erased. Screen windows are work areas that you designate on part of the screen; there'll be more about them later.

CONTROL

This key always works with another key. The **CONTROL** key works like the **SHIFT** key: you must hold it down while you press the other key.

1. As the COLOR KEYS section explains, pressing **CONTROL** and a color key allows you to choose the color of the text printed on the screen.

2. You can pause a program that is PRINTing or LISTing on the screen by pressing **CONTROL** and the S key (press any key to resume program output).

3. **CONTROL** is also used with the **REVERSE ON/OFF** and **FLASH ON/OFF** keys.

In addition, some software programs that you buy make use of the **CONTROL** key for their own special functions.

C=

Like the **CONTROL** key, the Commodore key works with other keys. It has four functions:
1. When used with the **SHIFT** key, the **C=** key lets you switch between upper-case/graphics mode and upper/lower-case text modes.
2. When you're in either mode, the **C=** key acts as a shift to let you type the graphics symbol pictured on the LEFT front of each key. Just hold down **C=** and press the graphic key you want.

3. When you want to change the color you are typing in to one of the 8 colors listed on the BOTTOM row of the face of the color keys, press **C=** and the color key you want.

4. When you want to slow down a scrolling program display, hold down the **C=** key. The display scrolling speed slows down considerably. When you release the key, the screen scrolling resumes normal speed. (Hey, it can do something by itself!)

The Color Keys

You can change the colors of the letters, numbers, and graphics symbols on the screen to any one of the 16 colors available on your Plus/4. It's simple to do:

- If you want one of the 8 colors listed on the TOP row on the front of the color keys (like BLK for black), just hold down the **CONTROL** key and press the key with the color you want at the same time.
- If you want one of the 8 colors listed on the BOTTOM row on the front of the color keys (orange, for example), just hold down the **C=** key and then press the color key with the color you want.

Practice changing colors to make sure you understand how to do this. You'll notice that after you change the color, every letter and number typed AFTERWARDS is in the color you last chose.

REVERSE ON

REVERSE OFF

Your Plus/4 lets you print the reverse image of letters and numbers. In other words, if you are using black letters on a yellow background, you can use the reverse image keys to print yellow letters on a black background.

Here's all you do to get reversed images: Press the **CONTROL** key and the **RVS ON** key. Now everything you type is displayed in reverse until you press the **CONTROL** and **RVS OFF**, the **RETURN** key, or the **ESCape** key and 0. This returns you to typing normal (non-reversed) characters.

FLASH ON

FLASH OFF

You can make the characters on your screen flash continuously. Just press **CONTROL** and the **FLASH ON** key to make whatever you type flash. Typing **CONTROL** and **FLASH OFF**, **RETURN**, or **ESCape** lets you type normal (non-flashing) characters again.

GRAPHIC KEYS

As we mentioned before, when you turn on the Plus/4, it is in upper-case/graphics text mode. When you're in this mode, you can type the full set of more than 60 graphics you see on the fronts of many of the keys, as well as all upper-case letters without using the **SHIFT** key. The **SHIFT** key lets you type graphics in this mode, instead of upper-case letters.

There are two graphic symbols on each graphics key:

- To print the graphic symbol on the right, hold down the **SHIFT** key while you press the appropriate key.
- To print the graphic symbol on the left, hold down the **C=** key while you press the selected key.

You can create pictures, charts, and designs by printing graphics side-by-side or on top of each other, like building blocks. Try printing some of the graphics keys to see how they work. Chapter 7 explains more about graphics.

You can switch between upper-case/graphics mode and upper/lower-case mode by pressing the **SHIFT** and **C=** keys at the same time. In either mode, type BASIC commands without holding down the **SHIFT** key.

In this mode, you can type upper- and lower-case letters, just like a regular typewriter. (You will have to shift for upper-case letters.) You also can use the graphic characters on the left front of the keys, which print as in upper-case/graphics mode; hold down **C=** and press the graphic key. The left side graphics are ideal for creating charts, graphs, and business forms.

ESCAPE

The **ESCape** key lets you perform many special screen edit-ing functions, including functions utilizing the windowing capability of the Plus/4. Windows are areas of the screen (defined by you) that may be used as work space without affecting the rest of the screen. The **ESCape** key can perform several window editing functions, as well as many other regular uses, such as inserting, deleting, and scrolling.

The **ESCape** key is typically used with standard alphabet keys. To activate a function, press the **ESCape** key followed by one of the keys listed below:

A   Automatic insert mode
B   Set the bottom right corner of the screen window
     (at the current cursor location)
C   Cancel automatic insert mode
D   Delete current line
I   Insert a line
J   Move to the beginning of the current line
K   Move to the end of the current line
L   Turn on scrolling
M   Turn off scrolling
N   Return to normal screen display size
O   Cancel insert, quote, reverse, and flash modes
P   Erase everything up to the cursor position on the current line
Q   Erase everything up to the end of the current line from
     cursor position
R   Reduce screen display
T   Set the top left corner of the screen window
V   Scroll screen up
W   Scroll screen down
X   Cancel the escape function

Special Symbols

The Plus/4 keyboard also contains special symbols not found
on many typewriters, or even on most computers. These special
symbols include the English pound sign (£), pi (π), greater and less
than signs (< >), brackets ([ ]), and arrows (↑). These special sym-
bols keys are used in programming your Plus/4.

| F1/F4 | F2/F5 | F3/F6 | HELP/F7 |
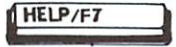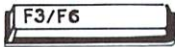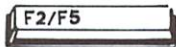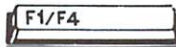
The four keys at the top of your keyboard are special function keys that let you save time by performing repetitive tasks with the stroke of just one key.

You can display what each key does by typing KEY and pressing RETURN.

The screen displays:

```
KEY
KEY 1,"GRAPHIC"
KEY 2,"DLOAD"+CHR$(34)
KEY 3,"DIRECTORY"+CHR$(13)
KEY 4,"SCNCLR"+CHR$(13)
KEY 5,"DSAVE"+CHR$(34)
KEY 6,"RUN"+CHR$(13)
KEY 7,"LIST"+CHR$(13)
KEY 8,"HELP"+CHR$(13)
```

Here's what each key does:

KEY 1   enters one of the GRAPHICS modes when you supply the number of the graphics area (e.g., GRAPHICS 2, which is split screen, high resolution mode) and a **RETURN** . On computers with built-in software, KEY 1 is redefined so that pressing it activates the software package.

KEY 2   prints DLOAD " on the screen. All you do is enter the program name to load a program from disk and hit **RETURN** instead of typing out DLOAD yourself.

KEY 3   lists a DIRECTORY of files on the disk in the disk drive.

KEY 4   clears the screen (even in one of the graphics modes.)

KEY 5   prints DSAVE " on the screen. All you do is enter the program name to save the current program on disk and press **RETURN**

KEY 6   RUNs the current program.

KEY 7   displays a LISTing of the current program.

KEY 8   (the **HELP** key) highlights errors in program statements in flashing print.

To use one of these functions, just press the appropriate function key. You need to use the **SHIFT** key to get FUNCTIONS 4, 5, 6, and 7.

You can redefine any of these keys to perform a function that suits your needs. Redefining is easy, using the KEY command. You can redefine the keys from BASIC programs, or change them at any time in direct mode. (The new definitions are erased when you turn off your Plus/4.) You can redefine as many keys as you want and as many times as you want.

| HELP |
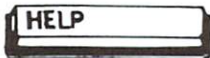
When you make an error in a program, the Plus/4 displays an error message to tell you what you did wrong. These error messages are further explained in Section 4 of the Plus/4 Encyclopedia in the second half of this manual.

You can get more assistance with errors by using the **HELP** key. After an error message, press **HELP** to locate your error. When you press **HELP**, the line with the error is displayed on the screen with the error flashing on and off. For example:

?SYNTAX ERROR IN ◄─── | Plus/4 displays this |
LINE 10

HELP ◄─── | You press HELP |

10 PRONT "COMMODORE ◄─── | Plus/4 displays this |
COMPUTERS" | with your error |
| flashing |
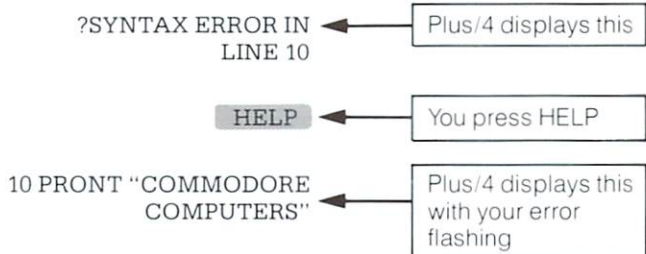
CHAPTER 3
USING
SOFTWARE

- Introduction

- Built-in software

- Cartridges

- Cassettes

- Diskettes

INTRODUCTION

The family of software available for your Plus/4 is growing quickly. Your dealer can keep you up-to-date on new products and inform you about the features of software that's currently available.

Your Commodore Plus/4 can use software on CARTRIDGE, CASSETTE TAPE and DISKETTE form, available from your Commodore dealer. All you do is load them into your Plus/4. You can also create and store your own programs on cassette tapes or floppy disks.

BUILT-IN SOFTWARE

Your Plus/4 is equipped with built-in software packages. These are programs built into the Plus/4, turned on by pressing the appropriate FUNCTION key. Your Plus/4 built-in software makes your computer a word processor, database, spreadsheet and graphics machine. A built-in package is ready to use whenever you power up your computer. When you turn on your Plus/4, the screen message tells you which packages are available and what function key to use to activate them. Also. you can use the KEY command to see the function key definitions. If there is function key software built into your Plus/4, the definition for KEY 1 will be: SYS XXXXX:package name. Just press FUNCTION KEY 1 and press **RETURN** to activate the program.

CARTRIDGES
Loading Cartridges

Commodore produces a full assortment of cartridge software for your Plus/4. There is a variety of personal, education, and business programs, as well as exciting games available on cartridge for your Plus/4. Follow these steps to use cartridges:

STEP 1   Turn OFF your Plus/4.

> **IMPORTANT:** YOU MUST **TURN OFF** YOUR COMPUTER BEFORE YOU **INSERT OR REMOVE** CARTRIDGES. IF YOU DON'T, YOU MAY **DAMAGE** THE **CARTRIDGE** AND THE **COMPUTER**.

STEP 2   Hold the cartridge with the label facing UP, and insert the cartridge firmly in the cartridge slot into the back of your computer.
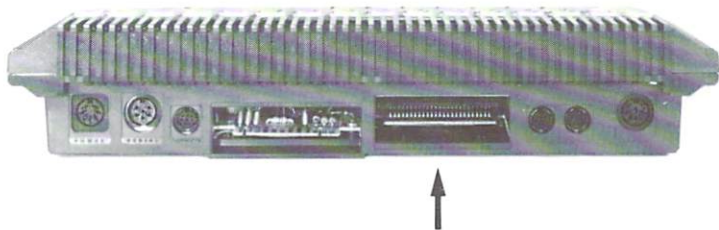
STEP 3   Turn ON your Plus/4.

STEP 4   Begin the game or program according to the instructions that come with the software. A cartridge program starts immediately, while function software starts after pressing the function key.

CASSETTES
Loading Cassette Tapes

A variety of software products for the Plus/4 is available on cassette tape. These cassette tapes are similar to the music cassettes that you play on your tape deck or stereo. Computer tapes run in the Datassette tape recorder, available from your Commodore dealer.

You can also use cassette tapes and the Datassette to store programs you write yourself. The next section explains how to save programs on tape.

The steps for loading tape are the same whether you are using prerecorded software or programs you saved yourself.

STEP 1   Insert the cassette into your Datassette and close the door.

STEP 2   Rewind the tape to the beginning by pressing the REWIND button on the Datassette.

STEP 3   When the tape is rewound to the beginning, type LOAD and press the **RETURN** key. The computer responds with the following message:

PRESS PLAY ON TAPE

STEP 4   Press the PLAY button on the Datassette. The screen goes blank as the Datassette starts. When a program is found, the screen displays this message:

FOUND "program name"

STEP 5   Press the Commodore key to load the program that was FOUND. If there is more than one program on the tape, and the program the Plus/4 found isn't the one you want, press the space bar to keep searching.

When the program is loaded, the word READY appears. If you want to stop the loading before it's complete, press the **RUN/STOP** key. After the software is loaded, type RUN to start the program. You can also LIST the program or change it, if it is a BASIC program.

> NOTE: To LOAD a specific program on the tape, use the LOAD "program name" form of the LOAD command.

Saving Programs On Cassette Tape

When you write a program and want to save it on cassette tape, follow these steps:

STEP 1   Type:

SAVE "program name"

The program name you use can be anything you want, but can be no more than 16 letters and/or numbers long.

STEP 2   Press the **RETURN** key. The computer displays this message:

PRESS RECORD AND PLAY ON TAPE

STEP 3   Press the RECORD and PLAY buttons on your Datassette. The screen goes blank. When your program is saved, the word READY appears on the screen.

Examples of SAVE Commands for Cassette Tape:

SAVE "MYJOB" ◄———   This name is the specific name of the program being saved

SAVE "3TEST" ◄———

> NOTE: When saving a program onto a cassette tape, always be aware of where the tape is positioned. In particular, be careful not to save a program at the absolute beginning of a tape, since many tapes have magnetic leaders, which will not record information. Thus, part of the program would not be saved.

When LOADing or SAVEing a program, if you decide to stop before it's finished, you must press the RUN/STOP key first. After pressing RUN/STOP on the keyboard, then press the stop button on the Datassette.

Disks are fast and easy to use. Be sure to handle your disks and your disk drive carefully. Disks may be referred to as diskettes, floppy disks, or floppies interchangeably; they are all the same thing.
The steps are the same for loading all disks:

STEP 1   Make sure that your disk drive is ON.

STEP 2   Insert the disk into the disk drive. The label side of the disk must face up. Insert the disk into the opening so that the labeled end goes in last. Look for a little notch on the side of disk (it might be covered with a sticker). This notch should be the left side as you put in the disk, assuming that you're facing your disk drive. Be sure the disk is in all the way.

STEP 3   Close the protective door on the disk drive after you insert the disk.

STEP 4   Type:

DLOAD "program name"   ◄——— Specific name of the program to be LOADed

To save time, you could press FUNCTION KEY 2 and type in the program name and the second quote marks.

STEP 5   Press the **RETURN** key. The disk spins and your screen says:

SEARCHING FOR PROGRAM NAME

LOADING

READY

■

STEP 6   Your software is ready to use. Now type RUN and press the
           RETURN key to start the program.

If the red light on the disk drive blinks after the DLOAD is finished,
something went wrong. Type:

?DS$ (and hit **RETURN**)

to find out what went wrong.

Examples of DLOAD commands:

DLOAD "*"	LOADs the 1st program on the disk.
DLOAD "MYFILE"	LOADs a disk program called MYFILE.
DLOAD "SET*"	LOADs the first program on the disk that begins with the letters SET.
DLOAD "$"	LOADs the directory, a listing of all the programs on the disk in the drive.

Headering A Diskette   Headering prepares a new BLANK disk for use. Any blank disk must be formatted before it may be used, by using the HEADER command.

IMPORTANT: DO NOT HEADER A DISK THAT HAS ANYTHING
ON IT UNLESS YOU WANT TO ERASE THE ENTIRE DISK.
HEADERING ERASES EVERYTHING ALREADY ON A DISK.

The format for the HEADER command is:

HEADER "disk name", Udevice#, Ii.d.#,Ddrive#

- The name you use is the name of the entire disk. Give the disk any name up to 16 characters.

- Device # specifies which device for your computer (disk drive as opposed to Datassette), and is usually number 8.

- The i.d. is the letter I and any two letters and/or numbers, like I21, IR5, etc. Give the disk any i.d. you want, but you should give every disk a different i.d. to avoid confusion.

- If you have a dual drive, add D0 or D1 to identify the drive number. If you have a single drive, you must use D0.

ARE YOU SURE?

As soon as you press RETURN after typing the HEADER command, the Plus/4 asks ARE YOU SURE? This is to give you a last chance to change your mind.

To header the disk, type YES or Y and press **RETURN** . If you decide not to header the disk, type NO or N and press **RETURN** .

Here are some examples of HEADER commands:

HEADER "LETTERS", U8,107,D0
HEADER "FINANCES", U8,IS3,D0

Now that you know how to HEADER a disk, you're ready to use disks to write and save programs on your Plus/4. The first section of the Plus/4 Encyclopedia has more information about the HEADER command.

Saving Programs On Diskette
When you want to reuse a program you've written, be sure to SAVE it before you LOAD another program or turn off the Plus/4. If you don't, you'll lose the program.

When you change a SAVEd program, you have to save it again if you want to keep the new version.

When you reSAVE a program, you are replacing the old version with

the new one. If you want to keep both the old and the changed versions, you have to give the new one a different name when you SAVE it.

Follow these steps to save a program on disk:

STEP 1 Type **DSAVE** "program name".

STEP 2 Press **RETURN** . The computer displays this message when the program is saved:

SAVING "program name"
OK
READY.

■

Example:

DSAVE "MYPROG5" ◄───── The program name can be up to 16 characters long.

If the red light on the disk drive blinks after the DSAVE is finished, something went wrong. Type:

?DS$ (and hit **RETURN**)

to learn what went wrong.

THE DIRECTORY COMMAND

When you SAVE programs on disk, the computer keeps a listing of all the files saved on that disk. You can display the listing as a table of contents to see what's on a disk by using the directory command:

Type: **DIRECTORY** then press **RETURN**
(or press FUNCTION KEY 3)
As soon as you press **RETURN** , your Plus/4 displays everything on your disk.
You can also display just part of the table of contents:

DIRECTORY "MY*" **RETURN** Lists every file on the disk that starts with the letters MY.

CHAPTER 4

GETTING STARTED

- Keyboard colors

- Color and reverse printing

- Some simple programs

- Correcting typing mistakes

- Introduction to the Plus/4 screen

- More about PRINTing on the screen

- Screen windows

INTRODUCTION The purpose of this chapter is to begin to acquaint you with some of the characteristics and capabilities of the Plus/4, and how to take the first steps toward programming with your computer.

KEYBOARD COLORS

You can change the color of the characters on the screen to improve readability or to find a color combination you like. To see how the different color characters look on your screen, try this:

STEP 1   Hold down the **CONTROL** key.

STEP 2   Press the 6 key while you're holding the **CONTROL** key down. The cursor turns green.

STEP 3   Let go of **CONTROL** and type some letters. Everything you type appears in green now.

HOLD **CONTROL** WITH COLOR KEY	COLOR RESULT
1	BLACK
2	WHITE
3	RED
4	CYAN
5	PURPLE
6	GREEN
7	BLUE
8	YELLOW

Using the **CONTROL** key with the number keys 1 through 8 allows you to choose the colors shown on the top of each color key.

Now hold down the **C=** key. By typing on the keys between 1 and 8, the cursor changes one of the 8 colors printed on the bottom of each color key. All 16 colors can appear on the screen at the same time.

HOLD **C=** WITH COLOR KEY	COLOR RESULT
1	ORANGE
2	BROWN
3	YELLOW-GREEN
4	PINK
5	BLUE-GREEN
6	LIGHT BLUE
7	DARK BLUE
8	LIGHT GREEN

COLOR AND REVERSE PRINTING

Your Plus/4 can display numbers, letters and graphic symbols in the 16 different colors. You can also display these characters in reverse, with the primary (cursor) and background colors reversed.

STEP 1   Clear your screen by pressing **SHIFT** and **CLR/HOME**

STEP 2   Hold down the **CONTROL** key and press the **RVS ON** key:

STEP 3   Release the keys and hold down the space bar (the long bar at the bottom of the keyboard).

STEP 4   Hold down the space bar as long as you want. While you hold down the space bar, a line the same color as the letters on your screen should get longer. If the line gets to the end of the row, it continues on the next row.

STEP 5   Release the space bar (but don't press the **RETURN** key).

STEP 6   Hold down the **CONTROL** key and press one of the color keys (not a color that's already on your screen). As soon as you do this, the cursor will be the color of the key you pressed.

STEP 7   Hold the space bar down again. Now your Plus/4 will draw a line in the new color. Continue changing colors with the **CONTROL** or **C=** keys and the color keys. Then hold down the space bar to make different colored lines.

STEP 8   Turn off reverse print by holding down **CONTROL** and pressing the **RVS OFF** key. Pressing the **RETURN** key also turns off reverse printing.

Try typing some letter in reverse. Just hold down CONTROL and RVS ON to turn on reverse, and then type whatever you want. Reverse letters make excellent headlines. You can also use them to highlight

special words and numbers. Try this:

PRINT" R COMMODORE ▬ PLUS/4"

Press **CONTROL**   Press **CONTROL**
and **RVS ON**   and **RVS OFF**

Now try the same line, replacing reverse on and off with FLASH ON
and OFF:

PRINT" ▌ COMMODORE ▬ PLUS/4"

Press **CONTROL**   Press **CONTROL**
and **FLASH ON**   and **FLASH OFF**

Both functions may also be used on the Plus/4 as part of a
program statement.

Type this program exactly as it appears here. Don't leave out the numbers at the beginning of the line, since they are the line numbers that tell the order that the lines of the program are implemented by your computer. Be sure to press the **RETURN** key at the end of each line you type.

10 PRINT"PLUS/4" ◄———

> This line tells your computer to PRINT Plus/4 on the screen.

20 GOTO 10 ◄———

> This line tells your computer to go back to line 10 and print Plus/4 again.

RUN ◄———

> This commands your computer to carry out what the 2 lines tell it to do.

Press the **RUN/STOP** key to stop the program.
Why did your Plus/4 print its name so many times? GOTO tells your computer to go back to line 10 and PRINT Plus/4 again and again. This repetition is called a loop.

Now type this:

NEW **RETURN**

> Here you tell the computer to forget the last program and get ready for a new one.

The computer responds:

READY. ◄———

> You DON'T type this; your Plus/4 does to tell you it's READY for a new program.

10 PRINT"PLUS/4" ◄──────

> Same line 10 as last time. PRINT tells your Plus/4 to display everything between the quotes.

20 COLOR 0,12 ◄──────

> This command tells the computer to change the color of the screen.

RUN

This time there's no GOTO loop in the program, so your orders are carried out just once.

CORRECTING TYPING MISTAKES

If you make a mistake when you're typing, there are several ways to make changes.

1. YOU CAN RETYPE A LINE anytime, even after you've RUN the program. The Plus/4 automatically replaces the old line with the new one when you press **RETURN** to enter the new line. The old line still appears on the screen, but the Plus/4 ignores it. When you have two statements with the same line number, your Plus/4 uses only the last one entered. For example, in a brief program using the COLOR command to change the color of the screen background, a mistake might occur:

10 COKOR 0.3 ◄────── mistake
20 PRINT "PLUS/4"

Press the **RETURN** key to get to a fresh line, and just retype line 10 correctly:

10 COLOR 0,3   **RETURN**

Now the first line 10 is replaced by the second line 10. You can check this by typing LIST, which displays a line-by-line LISTing of

your program as it is stored in the computer's memory. When you LIST a program, all lines appear in correct order and the replaced lines don't appear:

LIST   **RETURN**

The screen reads:

10 COLOR 0,3
20 PRINT "PLUS/4"

Replacing lines in a program is also a good way to experiment with your computer. When you replace a line, the new one doesn't have to be anything like the old line. For example, instead of correcting the spelling of COLOR, you can type this:

                    ↙ space
10 PRINT "TEN IS SIX   "

                      RETURN

Now RUN the program and see what happens.

2. YOU CAN ERASE AN UNWANTED LINE just by typing the number of the line and pressing **RETURN** . The computer ignores the line even though it still appears on the screen. Type LIST to get the program LISTing to make sure the line is gone from the program.

10 PRINT "TEN IS SIX"   **RETURN**

20 PRINT "PLUS/4"   **RETURN**

10   **RETURN**

LIST   **RETURN**

20 PRINT "PLUS/4"

3. YOU CAN EDIT A LINE. Use the cursor keys to move to the place in the line that you want to change. Now just type over what you want to change. Press **RETURN** when you finish.

10 PRINT "IT IS ONE O'CLOCK"

If you want to change the time to TWO, move the cursor to the
O in ONE.

10 PRINT "IT IS [O] NE O'CLOCK"

And now just type TWO over ONE and press **RETURN**

10 PRINT "IT IS TWO O'CLOCK"   **RETURN**

4. YOU CAN OPEN UP SPACES IN A WORD OR LINE with
    the **INST** key (get this insert key by holding down
    SHIFT while you press **INST/DEL**). Hold the
    keys down until you open up as many spaces as you need.
    (Notice that the cursor stays in the same place while spaces
    open up to the right.) Then just type what you want to insert.

10 PRINT "CORE"   **RETURN**

To change this to COMMODORE-Plus/4, move the cursor to the
hyphen between the C and the O and press the **SHIFT** and
INST keys until enough space opens up. Don't bother to
count out the spaces. You can just guess and then open up more
if there aren't enough.

10 PRINT "C ■ ORE"

cursor ←

Now add the other letters:

10 PRINT "COMMODORE"   **RETURN**

5. YOU CAN ERASE CHARACTERS AND CLOSE UP SPACE
    with the **DEL** key (get this delete key by pressing
    INST/DEL). This key erases characters or spaces
    immediately to the LEFT of the cursor.

10 PRINT "AFTERNOON SCHEDULE"   **RETURN**

You can change this to WEEKLY SCHEDULE by moving the cursor
to the E in AFTERNOON, pressing the **INST/DEL** key three
times, and typing WEEKLY.

10 PRINT "AFT **E** RNOON SCHEDULE"

and press **INST/DEL** three times.

10 PRINT " **E** RNOON SCHEDULE"

Type in WEEKLY to
replace ERNOON
and press **RETURN**

A Little Longer Program

Now that you've experimented a little with your Plus/4, here's a pro-
gram to try that will take a little longer to type in. (Any experienced
programmer will tell you that, if nothing else, programming can help
you really improve your typing.)

First, clear the screen by holding down the **SHIFT** key while
you press the **CLR/HOME** key. This erases your screen.
Then, clear out old programs from memory by typing NEW and
pressing **RETURN** .

Type in this program exactly as it appears. Remember to type in the
line numbers and all punctuation marks. Use the tips for correcting
mistakes if you type something incorrectly. Don't forget to press
RETURN after each line.

NOTE: Remember, you can stop a program by pressing the
RUN/STOP key.

NEW

10 COLOR 1,8

20 PRINT "A FUNNY THING HAPPENED   ";

Be sure to leave a
space here

30 COLOR 1,3

40 PRINT "ON MY WAY TO THE KEYBOARD   ";

Be sure to leave a
space here

50 COLOR 1,7

60 PRINT " ♥♥♥♥♥♥ "

Get the hearts
by holding down
SHIFT
while you press
the S key 6 times

70 GOTO 60

RUN

After you STOP the program (by pressing the **RUN/STOP** key), try
typing LIST. When the program is displayed on your screen, recall the
tips for correcting errors and try changing this program to make it say
something more profound.

TIP: Want to slow down this program without stopping it? Just
hold down the **C=** key.

**THE
PLUS/4
TEXT
SCREEN**

Try typing this program. (Don't forget to press RETURN after entering
each line.)

NEW

10 PRINT " ♥ ";

press **SHIFT**
and S

52

20 GOTO 10

RUN

Now your screen fills with hearts. When the entire screen is covered with hearts, press the **RUN/STOP** key to end the program. This program shows you how big your Plus/4's screen is.

Now type this program:

NEW

10 PRINT " ♥ " ← press **SHIFT** and **CLR/HOME**

20 FOR X = 1 TO 40

30 PRINT " ♥ "; ← press **SHIFT** and S

40 NEXT X

RUN

When you RUN this program, the first row on your screen fills completely with hearts. There are 40 hearts altogether. Since the row is full, you can see that there are 40 positions across the screen. Each position across the screen is called a COLUMN.

Now type this program:

NEW

10 PRINT " ♥ " ← press **SHIFT** and **CLR/HOME**

20 FOR X = 1 TO 25

30 PRINT " ♦ " ← press **SHIFT** and Z

40 NEXT X

RUN

When you run this program, the first column on your screen fills with diamonds. There are 25 diamonds printed, but the first three disappear at the top of the screen because the word READY surrounded by two blank lines always appears at the end of the program. There are, then, 25 rows. A little deductive logic tells you that your Plus/4 has 40 columns and 25 rows. The Plus/4 has 1000 different positions on the screen for letters, numbers, graphic symbols, etc.

NOTE: Sometimes you'll type a particularly long line on your Plus/4, such as this:

10 PRINT "I LIKE YOUR TOUCH ON MY KEYBOARD. DO YOU COME HERE OFTEN?'"

(That's quite a line—over 50 characters long!)

You'll notice that as you type this, you run out of room on one row. But keep typing; the Plus/4 automatically moves on to the next row and continues printing there until your line is finished. You can type as many as 80 characters on one program line (up to two full rows).

Now try RUNning this one line program. The message is printed on two rows. If your line is longer than one row, the Plus/4 lets it spill over to the next row. The Plus/4 considers the line ended when you press the **RETURN** key, not when you type to the end of the row. You'll get used to this as you use your Plus/4.

Now type this program:

NEW

10 PRINT " ♥ ";

20 GOTO 10

RUN

leave a space on each side of the heart

press **SHIFT** and S

When you RUN this program, you can see that it's possible to tell the Plus/4 exactly where to PRINT something on the screen.

SCREEN WINDOWS


```
COMMODORE BASIC V3.5 60671 BYTES FREE
READY.

WINDOWS LET YOU DEFI
NE A SPECIFIC AREA O
F THE SCREEN AS YOUR
 WORKSPACE.  YOU CAN
 SET UP A WINDOW ANY
WHERE ON THE SCREEN.
```
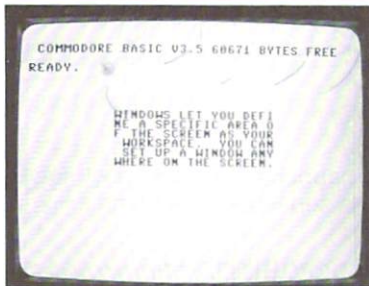
Windows let you define a specific area of the screen as your workspace. Everything you type (lines you type, LISTings of programs, etc.) after setting a window appears within the window's boundaries, not affecting the screen outside the window area. You can set up a window anywhere on the screen.

To set a window, follow these steps:
1. Move the cursor to the screen position you want as the top left corner of the window.
2. Press the ESCape key, and then the letter T.
3. Move the cursor to the position you want to be the bottom right corner of the window.
4. Press ESCape, followed by B. Your window is now set.

All screen output is confined to the 'box' you have defined. To cancel the window, press the home key twice. The window is then erased, and the cursor is positioned in the top left corner of the screen.

CHAPTER 5

NUMBERS
AND CALCULATIONS

- Numbers and basic operators

- Performing calculations

- Using variables

- Immediate mode

- Numeric functions

- Random numbers and other functions

You can use your Plus/4 like a simple calculator. Besides the standard + and − signs, your Plus/4 uses the *sign for multiplication and the / sign for division and fractions. (Computers use the * sign instead of an X for multiplication because a computer can't tell the difference between the letter X and the mathematical symbol ×.) You can use these operators and numbers in direct mode (no line numbers) or in a program. Neither numbers or operators should be in quotes for your Plus/4 to perform mathematical operations.

BASIC MATHEMATICAL OPERATORS		BASIC RELATIONAL OPERATORS	
Addition	+	Greater than	>
Subtraction	−	Less than	<
Division and fractions	/	Equals	=
Multiplication	*	Greater than or equal	=>
Exponentiation	↑	Less than or equal	<=
		Not equal to	<> or ><

> NOTE: Your Plus/4 doesn't accept commas as part of a number. For example, you must type 109401 instead of 109,401. If you put a comma in a number, your Plus/4 thinks you mean two numbers (separated by the comma), so your Plus/4 would read 109 and 401 instead of 109401.

FRACTIONS AND DECIMALS

You can write a fraction like this: .5
or like this: ½ ◄— Your Plus/4 is actually performing the division

If you put a fraction in a PRINT statement, your answer is always returned as a decimal or whole number. For example:

PRINT 139/493 + 5　**RETURN**
5.28194726

Here's an example that uses pi (3.14159256...), which represents the ratio of the circumference of a circle to its diameter. Use this value by just pressing the π key:

PRINT π/374　**RETURN**
8.39998036E-03

SCIENTIFIC NOTATION

What did your Plus/4 mean by the E-03 part of the above answer? Your Plus/4 displays decimal numbers in the range −999,999,999 to 999,999,999 in standard numerals. Numbers beyond this range (with more than nine digits) are automatically displayed in scientific notation You can enter numbers in yourself in this form and your Plus/4 will read them with no trouble (certainly less trouble than you had converting them). Scientific notation is often useful, since this special notation lets your Plus/4 display large numbers in fewer digits.

Here is how the number 198,505,478 would be written in scientific notation:

Only ONE digit is shown to the left of the decimal point

1.98505478E + 8

This number is the number of digit places the decimal point is moved
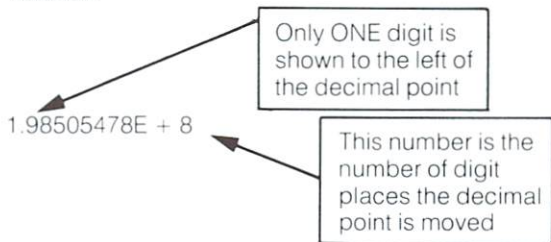
For a number less than one with several decimal places, the second number would be a − instead of a +, indicating that the decimal point is moved to the right.
For example:

.0003359 = 3.359E − 4

Other examples:

$$20 = 2E + 1$$ the decimal point is moved 1 digit left
$$105000 = 1.05E + 5$$ the decimal point is moved 5 digits left
$$.0666 = 6.66E − 2$$ the decimal point is moved 2 digits right

PERFORMING CALCULATIONS

To perform a calculation, type PRINT and then the math problem. Remember not to put the problem in quotes.
Type this program:

```
NEW
10 PRINT 1+2, 2-1
20 PRINT 2*2, 4/2
RUN
3      1
4      2
```

use the slash on the ? key

For the first time, PRINT didn't print exactly what you typed in the statement. Instead, your Plus/4 solved the calculations and PRINTed the answers. All you have to do to use PRINT to calculate is omit the quotation marks. Now try this:

```
NEW
10 PRINT "2001/2010"
20 PRINT 2*3
RUN
2001/2010
   6
```

one space is left for the answer's sign

Since the calculation in line 10 is in quotes, your Plus/4 just PRINTs the problem as if it were regular text: exactly as it appears between the quotation marks. The problem isn't solved, and no space is left for the sign of the number.

Now move the cursor back to line 10 and change the line to this:

```
10 PRINT "2*3+1=";2*3+1
```

don't forget the semicolon

```
RUN
2*3+1 =  7
```

this space is left for the answer's sign

```
   6
```

the answer for line 20 stays the same

If you want to both PRINT the problem AND solve it you have to type it twice: once in quotes and once out of quotes, like this:

IMMEDIATE MODE

You can put any calculation in a program, or get an immediate answer by typing PRINT and the problem without a line number and pressing RETURN, like this:

```
PRINT 3−6
−3
PRINT 24/(6+2)
3
```

With numbers as well as with commands and text, when you don't have a line number before a BASIC statement, you don't have to type RUN to tell the computer to follow the instruction; it's in IMMEDIATE, or DIRECT MODE. Having a line number means the statement is part of a BASIC program; it's in PROGRAM MODE. Either way is acceptable.

You can also include both a text statement in quotation marks and a mathematical problem to be solved in a single PRINT statement in immediate mode.

PRINT "2 TO THE 3RD POWER EQUALS";2↑3

> this arrow stands for exponentiation; get it by typing SHIFT and 0

2 TO THE 3RD POWER EQUALS 8

> the message is PRINTed and then the problem's solution is PRINTed

ORDER OF CALCULATION

The second example in the last section shows that you can perform more than one calculation in one line. Try typing this:

PRINT 200+50/5

Is the answer what you expected? Try this:

PRINT (200+50)/5

Your Plus/4 always performs calculations in a certain order. Problems are solved from left to right; within that general rule, some types of calculations are solved first. The order which your Plus/4 evaluates expressions is called the order of precedence of operators.

FIRST: Your Plus/4 checks for negative numbers (not subtraction, just negative numbers).

SECOND: Your Plus/4 solves any exponents.

THIRD: Your Plus/4 solves all multiplications and divisions, from left to right.

FOURTH: Your Plus/4 solves additions and subtractions, from left to right.

> NOTE: Your Plus/4 always solves any portion of the problem surrounded by parentheses first. You can even put parentheses within parentheses: $36 * (12 + (A / 3))$. The contents of the innermost parentheses are solved first.

Sometimes it's a good idea to put negative numbers in parentheses for clarity. For example, if you want to multiply 45 by -5, type it like this: $45*(-5)$. Your Plus/4 can understand with or without parentheses.

USING VARIABLES The example 36*(12+(A/3)) shows one of the most powerful features of a computer. When we used a letter instead of a number in a mathematical problem, we used a VARIABLE. A variable represents a value:

10 A = 3
20 PRINT "TOTAL:";A * 4

If you RUN this program, the screen result is:
TOTAL: 12
There are three types of variables you can use:

TYPE	SYMBOL	DESCRIPTION	EXAMPLES	SAMPLE VALUES
Floating point		real (decimal) or whole numbers	X, AB, T4	23.5, 12, 1.3E+2
Integer	%	whole numbers	X%, AI%	15, 102, 3
Text string	$	letters, numbers, and all other characters in quotes	X$, MS$	"TOTAL:", "DAY 1", "CBM"

Every time you want a variable to be an integer variable, the symbol for that variable would include the % sign. A variable that contains text MUST end with a $ as part of the variable. If it doesn't have that symbol, your Plus/4 considers it a floating point number. A variable without either of the symbols (% or $) is read as a floating point number (a "regular" number). Integer variables are a subset of floating point variables; they are numbers with no decimal places.

Always use the right variable type. If you try to do something like assign a word to an integer variable, your program won't work. This program shows you what variable can or can't be used in a given situation, and you can find out what happens when you try out different types of data:

10 REM THIS PROGRAM NEEDS NUMERIC DATA
20 PRINT "ENTER A NUMBER"
30 INPUT X% ◄——— this is the variable to be input
40 PRINT "NICE GOING, ACE!"
50 PRINT "I READ YOUR NUMBER AS"; X%

Try to enter these values and see what happens:
ONE FIFTH
.043
10

NUMERIC FUNCTIONS

Included in your Plus/4 BASIC 3.5 language are numeric functions, which are like the advanced calculations found on most scientific calculators (such as sine, cosine, tangent, etc.).

Most of the functions can be used by typing the name of the function and the number to be operated by the formula in parentheses, like this:

FUNCTION(X)

For example, to find out the sine of a variable, you would type:

PRINT SIN(X)

with X as any number you want to input.

You could also include one of the functions in a program line, as the following example shows:

```
10 FOR X = 1 TO 5
20 PRINT "THE SQUARE ROOT OF"; X; "IS"; SQR(X)
30 NEXT X
```

There is a complete listing of the numeric functions in the BASIC 3.5 Encyclopedia. Some of the more complex functions are explained in the following paragraphs.

RANDOM NUMBERS AND OTHER FUNCTIONS

Selecting a random number is like taking 10 pieces of paper, writing a number from 1 to 10 on each piece and putting the 10 pieces of paper into a hat and drawing one piece of paper. The number chosen is a RANDOM number. The number is put back into the hat, and another number is drawn. Each time a number is drawn, it is put back in the hat, keeping the pool of possible numbers the same. When a number is selected, there is no way of knowing what number is going to come up next, but you do know that the number will be between 1 and 10. This is the basis of RANDOM NUMBERS.

Random numbers are extremely useful in programming, providing the element of chance or (obviously) randomness. Random numbers usually have a range, meaning there's an upper limit and a lower limit to the numbers you can draw. In the hat example, the range of numbers is 1 to 10. The lower limit is "1" and the upper limit is "10", which means that any number from 1 to 10 can come up randomly each time a new number is selected.

Let's examine how your Plus/4 handles random numbers, and some of the things you can do with them. This program generates five completely random numbers:

10 FOR X=1 TO 5:PRINT RND(X):NEXT X

These random numbers are all rather complex, with several places on the right side of the decimal point... but most uses for random numbers require whole numbers. You can make your numbers come out as whole numbers (without decimal places) by using the INTeger function, which cuts off all the digits on the right side of the decimal point. The following gives you a formula for generating random numbers in any range you want. You can use this formula almost anywhere you would use a variable or number in your program.

INT(range*RND(1)) lower limit

The INT command tells the computer to cut off any decimal places and only give you whole numbers like 1, 45, or 320, instead of numbers like 1.223, 45.6677, or 320.59. Whole numbers are easier to work with when using random numbers.

Lower Limit in the formula refers to the lowest number you want the computer to choose from.

Range is how many numbers are in the total group.

For example, if you want to choose a random number from 1 to 5, the lower limit is 1 and the range is 5. If you want to choose a random number from 15 to 20, the lower limit would be 15 and the range is 6, because you are choosing from a pool of 6 numbers. If you're choosing numbers from 2 to 100, the lower limit is 2 and the range is 99. Now let's try out a program:

10 PRINT INT(5*RND(1)) + 1

Type RUN and press **RETURN** . RUN the program a few times. Each time you run the program, you get a random number from 1 to 5. Now let's print 15 random numbers, with the lower limit 1 and the range 5... note that all 15 numbers chosen are selected at random from between 1 and 5:

```
10 FOR X=1 TO 15                    sets loop for 15 times
20 PRINT INT(5*RND(1)) + 1
30 NEXT X

                                    selects RaNDom
                                    number
```

Type RUN and press RETURN.

An effective way to use this formula is to make it into a **user defined function.** User defined functions are extremely useful in mathematical calculations, and extremely easy to implement using your Plus/4. User defined functions allow you to program a formula, and then let your Plus/4 plug in values to be calculated. This can be used for many different purposes. Section 10 of the Encyclopedia section contains a listing of mathematical function derivatives which can be used to define functions.

Here is a statement utilizing the user defined function for generating random numbers:

```
10 DEF FNR(X)=INT(X*RND(1)) + 1
```

This gives us random numbers in the range from 1 to X. FNR is the name of the function defined by this statement.

EXAMPLE using a defined function:

```
10 DEF FNR(X)=INT(X*RND(1)) + 1
20 DO
30 COLOR 1, FNR(16), 5: REM PICK A COLOR FROM 1 TO 16
40 PRINT "THE SEARCH GOES ON"
50 LOOP
```

Using the defined function saves memory space when you use the function more than once, and makes your programs easier to read and understand.

CHAPTER 6

BEGINNING BASIC PROGRAMMING

- Introduction

- Programming modes

- Input/Output statements

- Control statements and loops

- Conditional statements

- Subroutines

- REMarks

Up until now, you may or may not have understood exactly what was going on in the programs that introduced you to some of the capabilities of your Plus/4. This chapter will explain some of the BASIC commands that were used in those programs. This chapter focuses on some of the more often used BASIC statements that you will need to construct your own programs. At the end of this chapter we will touch on some programming techniques. This chapter will give you a quick introduction to programming, but it is still an introduction. To really learn to program, we suggest you pick up a good book on BASIC at your local bookstore. (See the bibliography in Section 14 of the Encyclopedia for suggestions.) There are many versions of BASIC, each a little different. Your Plus/4 is equipped with an advanced version of the BASIC language called Commodore BASIC 3.5.

PROGRAMMING MODES

Your Plus/4 gives you two ways to use BASIC statements and commands: in direct mode and indirect mode. Direct mode is often referred to as immediate mode, and indirect mode is also known as program mode.

DIRECT, or IMMEDIATE MODE, as the name implies, executes statements and commands immediately (as soon as you press **RETURN** after typing in a command). You do not type line numbers when using commands or statements in direct mode. You only type the command or statement and press the **RETURN** key. This mode is used if you want your computer to perform calculations and give you an immediate result. Commands such as LIST, SAVE, LOAD, VERIFY and RUN are usually used in direct mode. Most (but not all) BASIC statements work in direct mode.

INDIRECT, or PROGRAM MODE, allows you to organize a series of BASIC statements into a set of instructions that will be performed in the order that you decide. Each of the lines in the program has a line number which tells the computer to execute the lines in a certain order. You've already seen several examples of program mode in Chapter 4. Remember that when you use program mode, you must press **RETURN** to enter each line of the program into the memory of your Plus/4. If you don't press **RETURN**, and just go to the next line, the line you typed has not been entered. Once the program is in memory, nothing will happen until you enter the RUN command. The RUN command tells the Plus/4 to execute the program starting with the lowest numbered line.

Lines are most often numbered by tens, since you'll frequently have to add lines in different places in the process of writing a program. You could, if need be, add nine new lines between line 10 and line 20 in a program. However, your Plus/4 features a BASIC command, RENUMBER, that allows you to add new lines and change the existing line numbers. This saves a lot of confusion that often occurs when changing and rearranging lines.

Input/Output (I/O) statements are used in programs to communicate with the person RUNning the program. Before the program is run, if all the data for the calculations is available, there is really little need for input statements. It is often more useful if the computer can get data from the person RUNning the program (we'll call him or her the program user). Programs are much more versatile if the data is not "set in stone" before running them. Output statements can be used by the computer to tell the person running the program the answers that the computer has calculated. Obviously, output statements are vital; there would be little sense in RUNning a program that had no output statements. (Kind of like a tree falling in a forest with no one around to hear; does it make a sound? Does it matter?)

Advanced programmers also use I/O statements to communicate with devices instead of with the program user. You've probably done this yourself, but not in a program—when you used LOAD or SAVE with your Datassette or Disk drive. LOAD is basically an input statement since the Plus/4 gets data (your program) from your Datassette or disk drive while SAVE is an output statement, as the Plus/4 sends data to those devices.

In this introduction to I/O statements, we will limit ourselves to a few of the most important ones, the ones that you'll need immediately. They are: PRINT, INPUT, GETKEY, and READ/DATA. PRINT is an output statement, while the others are input statements. (Remember that all BASIC I/O statements can be found in the BASIC Encyclopedia at the end of this book.)

Statement name: **PRINT**

Format: **PRINT** "text in quotes" or variables or numbers or calculations, etc.

You have used the PRINT statement often in programs in earlier chapters. From that, and from the format example above, you can see that PRINT is a very versatile statement. You can use it to PRINT out messages, pictures made out of graphic characters, perform calculations, display the value of a variable, and more. Since the PRINT statement is used so often, it pays to learn to use it well.

Use #1 Text Display
Suppose in your program, you want to inform the user that his or her checking account balance is negative, or that purple lizards are

not allowed in the control room. The easiest way would be to PRINT
your statement as a text string. Text strings are printed out exactly as
you type them in. They must be surrounded by double quotes (" ").
For example:

100 PRINT "YOU ARE BROKE!"

would tell the user that there is no money left, while

150 PRINT "YOU CAN'T BRING YOUR FRIEND INTO THE
CONTROL ROOM"

could be used in the second example.

Whatever appears between the quotation marks is known as a literal,
because it is PRINTed exactly as it appears. It doesn't matter whether it
is words, letters, numbers, punctuation marks, etc.

Certain keys, like the cursor and color keys, act differently when used
in text strings. Instead of changing the color or moving the cursor when
you type the key, a reverse character is printed in the string. When
the program is RUN, the character is translated into what you wanted
typed in the first place. This lets you clear the screen, change the color
your are PRINTing in, move the cursor, all within your program. For
example, try this:

10 PRINT " `SHIFT` `CLR/HOME` `CONTROL` 3 TESTING,
`CRSR-DOWN` `CONTROL` 7 TESTING"

Remember to type the following keys simultaneously when you use
the SHIFT and CONTROL keys. The reversed symbols are the signals
to the computer that tells it to perform the clear screen, color change
or move the cursor.

Use #2 Printing Numbers and Calculations
PRINT can display the answer to a calculation made within the print
statement. (SEE NUMBERS and CALCULATIONS). The Plus/4 per-
forms the operations needed to get the answer, then displays it on
the screen. For example:

100 PRINT 58*15,23,45+1000-45*(4-3)
    prints:
870   23   1000

This gets more interesting when variables are also used. User input can be displayed, and earlier calculations saved in variables can also be PRINTed out, or even used in additional calculations.

Examples:

TYPE:

```
10 R = 10 * 2: N = R − 5
20 PRINT "R IS ";R;" AND N IS ";N
30 PRINT "BUT R TIMES 2 IS";R*2
40 PRINT "AND N MINUS 2 IS";N − 2
```

Normally, after each PRINT statement, the cursor automatically goes to the beginning of the next line. You can override this by putting a semicolon (;) after the PRINT statement like this:

```
200 PRINT "THESE TWO SENTENCE PARTS WILL BE ";
210 PRINT "PRINTED ON THE SAME LINE"
```

Statement name: **INPUT**

Format: **INPUT** "optional message";variable to be input

The INPUT statement lets you get data from the program user through the keyboard, and use it in the program. The optional message lets you tell the user exactly what you are asking for; the message is printed when the INPUT statement is executed, along with a question mark. Then the Plus/4 waits for the user to type an answer, followed by pressing the **RETURN** key. The input from the user is placed in a variable. You can either get a string from the user by using a string variable (A$, for example), or a number by using a numeric variable. The INPUT statement can only be used in program mode.

Examples:

TYPE:

```
10 PRINT "WHAT IS YOUR NAME";
20 INPUT A$
30 PRINT "I AM PLEASED TO MEET YOU";A$;"."
40 INPUT"HOW OLD ARE YOU";AG
50 PRINT AG;" IS A BIT OLDER THAN I AM."
RUN
```

Statement name: **GETKEY**

Format: **GETKEY** variable to be input

GETKEY is another way for you to enter data while the program is being RUN. The GETKEY statement accepts only one key at a time. Whatever key is pressed is assigned to the string variable you specified in the GET statement (A$, for example). GETKEY is useful because it allows you to enter data one character at a time without having to press the **RETURN** key after each character. The GETKEY statement may only be used in a program.

Example of GETKEY in a program:

```
1000 PRINT "PLEASE CHOOSE A, B, C, D, E, OR F"
1010 GETKEY A$
```

Statement Name: **READ/DATA**

Format: **READ** variables to be input
DATA data items to be read

The READ/DATA statements are used as a convenient way to assign values to variables. You can think of the READ statement as an INPUT statement that asks the Plus/4 for the data, rather than the user. The data is (naturally enough) kept in DATA statements. When the Plus/4 executes a READ statement, it looks at the next data item in the DATA statement, and assigns it to the variable in the READ statement.

The READ statement is always used with a DATA statement. A DATA statement is just a line of data (words or numbers) in a program. The READ statement is used to assign those values to variables. (For each variable listed in the READ statement, your Plus/4 "reads" a value from the DATA line for that variable.) A DATA statement is not executable and can appear anywhere in the program. The thing to remember about the READ statement is that the variable type must be the same as the type of data available in the DATA statement (number variables for numbers, text variables for text). Otherwise, a TYPE MISMATCH ERROR occurs.

Example:

```
10 READ A$,B$,C$,D$,E$
20 PRINT A$:PRINT B$:PRINT C$
30 PRINT D$: PRINT E$
```

40 DATA GROUCHO, HARPO, CHICO
50 DATA ZEPPO, GUMMO

The computer responds with:

GROUCHO
HARPO
CHICO
ZEPPO
GUMMO

CONTROL STATEMENTS AND LOOPS

It would be pretty boring if your computer could only execute program lines in order. The computer could only start at the beginning and go through each step in order until the end of the program. This would lead to very long programs; if you wanted to do the same thing twice (like PRINT "HELLO"), you would have to duplicate the program lines. With a small example like PRINTing HELLO, this doesn't make a lot of difference, but it could become difficult in larger programs. This is why computers have control statements. Control statements tell the computer to ignore the normal order of the program lines, and go to another line regardless of the sequence. The Plus/4 has several varieties of control statements: unconditional (like GOTO) which *always* transfer control; counting statements (like FOR/NEXT) which transfer control a specified number of times; and, for you structured programming fans out there, DO/LOOP.

Statement Name: **GOTO**

Format: **GOTO** line #

GOTO tells your computer to immediately go from the current line in your program to the line number specified in the GOTO statement. For example, if line 20 reads GOTO 40, your Plus/4 would jump to line 40, skipping any statements between 20 and 40.

Example using GOTO statement in a program:

TYPE:

10 PRINT "A PENNY SAVED IS BETTER THAN NOTHING"
20 GOTO 10

The computer responds by printing the message in line 10 again and again, until you press the STOP key, like this:

A PENNY SAVED IS BETTER THAN NOTHING
A PENNY SAVED IS BETTER THAN NOTHING
A PENNY SAVED IS BETTER THAN NOTHING

BREAK IN 10 ◄——— | If you press the **STOP** key |
READY.

This print statement will continue 'forever'. Every time your Plus/4 gets to the GOTO in line 20 it goes back to line 10. This is called an INFINITE

LOOP in computerese. While you might want to do this, usually you want to repeat only a certain number of times, or until something happens. That is why the FOR/NEXT and DO/LOOP statements are available in BASIC.

GOTO can also be used in direct mode. GOTO line #   will start the program at the line you specify, while keeping the variables the same (instead of clearing them as RUN does).

Statement Name: **FOR/NEXT**

Format: **FOR** variable = start value **TO** end value

some BASIC statements

NEXT variable

The FOR/NEXT statements let you create a loop that will repeat a certain number of times. The program statements between the FOR statement and the matching NEXT statement are repeated in the loop. The variable in the FOR statement acts as a counter. It is initially set at the start value you supply. Then, the program lines after the FOR are executed, until the computer gets to the matching NEXT statement. The NEXT tells your Plus/4 to add one to the counter. If the counter is less than or equal to the end value, the computer returns to the program line after the FOR statement. Otherwise, your Plus/4 continues with the first statement after the NEXT.

Example using a FOR/NEXT loop

```
10 PRINT,"COUNTUP..."
20 FOR J = 1 TO 10
30 PRINT "WE HAVE";J
40 NEXT J
50 PRINT "WE COUNTED UP TO";J
```

One more thing about FOR/NEXT: you can also specify a STEP value in the FOR statement. Instead of adding 1 to the counter variable, your Plus/4 adds your STEP value. If you use a STEP of 5 with the statement FOR M = 10 TO 30, for example, the counter would count 10, 15, 20, 25, 30 after each loop. The STEP command even lets you count backwards (by using a negative STEP value).

Another example, with a negative STEP:

```
10 PRINT "COUNTDOWN..."
20 FOR J = 10 TO 0 STEP −1
30 PRINT "WE ARE AT";J
40 NEXT J
50 PRINT "WE HAVE LIFT-OFF AT";J
```

Statement Name: DO UNTIL/WHILE...LOOP UNTIL/WHILE

Format: DO UNTIL [condition] WHILE [condition]

some BASIC statements

[EXIT]

LOOP UNTIL [condition] WHILE [condition]

The DO/LOOP statement combination is another way to create a loop. This statement combination is very powerful and versatile. The DO/LOOP method of loops is a common technique of structured programming languages. In this chapter we'll discuss just a few possible uses.

If you want to create an infinite loop, just start a section of program lines with DO, and end it with a LOOP statement, like this:

```
100 DO: PRINT "GOING UP"
110 LOOP
```

Press the **STOP** key to end the program.

A more useful form is to combine the DO/LOOP with the UNTIL statement. The loop will run continually unless the condition for UNTIL happens.

```
100 DO: INPUT "DO YOU LIKE YOUR COMPUTER";A$
110 LOOP UNTIL A$="YES"
120 PRINT "THANK YOU"
```

For the other ways you can use the DO/LOOP, see the BASIC Encyclopedia at the end of this book.

CONDITIONAL OR DECISION MAKING STATEMENTS

Conditional statements are used to make decisions. One of the most powerful abilities of a computer is to make decisions based on what is going on. One of the conditional statements available on the Plus/4 is known as IF/THEN statements.

Statement Name: **IF/THEN**

Format: **IF** condition **THEN** do this (only if the condition is true)
Basically, the IF/THEN statement works like this:

IF (this statement is true) THEN (do this statement)
Actually, you have always known how conditional statements work. How many times have you heard this famous line?:

IF you eat all your vegetables THEN you can have dessert. That may seem a bit trivial, but that is the gist of the IF/THEN statement.

If the condition in the IF statement is true, everything after the THEN is executed.

EXAMPLE:

```
10 INPUT"WHAT'S THE TENTH LETTER OF THE ALPHABET";A$
20 IF A$="J" THEN PRINT "RIGHT": GOTO 100
30 INPUT "IS THIS AN A";X$
40 IF X$="A" THEN 60
50 PRINT "WRONG, TRY AGAIN": GOTO 30
60 PRINT "TYPE A B"
70 GETKEY A$:IF A$="B"THEN PRINT "RIGHT"
100 PRINT "THAT'S ENOUGH OF THIS, ANYWAY"
```

In line 40 , we just say THEN 60. This actually means THEN GOTO 60, but since the THEN GOTO combination is used so often, BASIC allows you to leave off the GOTO. An optional step for the IF/THEN statement is the ELSE clause, that directs your computer to a specific action if the original IF condition was not met. An example showing the ELSE clause would be: IF B > 5 THEN 40 ELSE GOTO 10. The BASIC Encyclopedia explains the IF/THEN/ELSE statement more fully.

SUBROUTINES

If you have something in your program that has to be repeated in more than one place in your program, you have two choices: you can have duplicate routines, or you can create a subroutine. A subroutine is a section of your program that can be used from anywhere else in your program. When the subroutine is finished, the program automatically continues at the statement just after where the subroutine was called.

Statement Name: GOSUB/RETURN

Format: GOSUB line #

The GOSUB statement is used to call a subroutine. Like the GOTO statement. control is transferred to the line number specified in the statement. However, unlike the GOTO, the Plus/4 remembers where the GOSUB is located. When a RETURN is next encountered, control returns to just after the GOSUB statement.

Example:

```
5 T=0:FOR J = 1 TO 99
10 PRINT "GIVE ME A NUMBER FROM 1 TO 10"
20 INPUT N
30 IF N<1 THEN GOSUB 100:GOTO 20
40 IF N>10 THEN GOSUB 100:GOTO 20
50 T=T+N
60 NEXT J
70 PRINT "THE TOTAL IS"J
80 END
100 PRINT "THAT NUMBER IS OUT OF RANGE"
105 PRINT "PLEASE TYPE A NUMBER BETWEEN 1 AND 10
110 RETURN
```

If a RETURN is encountered when there are no active GOSUBs, you get a RETURN WITHOUT GOSUB ERROR. You should be careful that the computer never gets into one of your subroutines except by GOTO. One method is to group the GOSUB and GOTO statements together, protected from normal program execution by an END statement.

Statement Name: REM

Format: REM message

The REM statement is used to comment (or REMark) on your programs. The REM statement is not executed as part of the program; it is

a message that can be seen only when looking over the LISTing of a program. Often, if you don't comment, six months after you write the program you might forget what some part does. You can use REM statements to put in reminders, so you can more easily figure out what you really meant, or give others information with your messages.

Example:

1560 E= R/I*9:REM THIS FIGURES OUT A PITCHER'S ERA
100 INPUT A, B: REM A IS HEIGHT IN INCHES AND B IS WEIGHT

SUMMARY As we REMarked in the introduction, this would not be a complete tutorial on BASIC. We just gave you some of the BASICs. Every BASIC command in the Plus/4 is in the BASIC Encyclopedia, with format, description, and examples. Don't be afraid to experiment. If you are serious about learning BASIC, get some of the books on BASIC programming listed in the Section 14 of the Encyclopedia. Programming is like eating salted peanuts: once you start, you may not be able to stop.

CHAPTER 7
USING GRAPHICS AND COLOR

- Graphics characters

- Character animation

- Controlling colors

- High resolution graphics

- Points, lines, and labels

- Squares, circles, polygons, and painting

- Multi-color graphics

Each letter key contains 2 different graphic characters, as do the @, –, *, and English Pound keys. To print graphics characters, you must hold down the **SHIFT** or **C=** keys while you press the key for the graphics symbol you want.

When your Plus/4 is in upper-case/graphics mode, hold down **SHIFT** and press a letter key to display the graphics character on the right side of that letter key. These characters include the playing card suits, a solid and a hollow ball, and a set of lines and connecting characters that let you draw many different pictures on your screen.

Here are some examples to help you get used to these characters:

Exercise 1: Large Circle

Step 1: Press down the **SHIFT LOCK** key.
    2: Press the letter U then the letter I.
    3: Press the **RETURN** key.
    4: Press the letter J then the letter K.
    5: Press the **RETURN** key.

Exercise 2: Snake

Step 1: Press down the **SHIFT LOCK** key.
    2: Press U, then I, then U, then i, then U, then I.
    3: Press the **RETURN** key.
    4: Press K, then J, then K, then J, then K, then J.
    5: Press the **RETURN** key.

Exercise 3: Crooked Line

Step 1: Press down the **SHIFT LOCK** key.
    2: Press E, then D, then C, then *, then F, then R.
    3: Press the **RETURN** key.

Exercise 4: Two Crosses

Step 1: Press down the **SHIFT LOCK** key.
    2: Press M, then SPACE, then N, then SPACE, then —.
    3: Press the **RETURN** key.
    4: Press SPACE, then V, then SPACE, then *, then +, then *.
    5: Press the **RETURN** key.
    6: Press N, then SPACE, then M, then SPACE, then —.
    7: Press the **RETURN** key.

When you are finished, press **SHIFT LOCK** again so it pops up.

Did you wonder why the computer doesn't say SYNTAX ERROR when you hit **RETURN** ? After all, you had characters on the line that weren't commands that the computer can understand.

The reason is that the Plus/4 doesn't pay attention to the line you typed if you hold down **SHIFT** when you press **RETURN.** If you press **RETURN** without the **SHIFT** key, the computer tries to figure out what you mean when you're just drawing pictures.

So far we haven't talked about the graphics characters on the left side of the keys. These graphics work just like the right side characters, except that you hold down the **C=** key instead of **SHIFT** . There is no **C=** lock, so you must hold it down yourself.

You can print this set of graphics in either upper-case/graphics mode or upper/lower-case mode. They are always available.

The left side graphics characters include lines and angles used for drawing charts and tables. For example, here's how to underline a word:

First, move the cursor to the line below the word you want to underline. Then hold down the **C=** key and the T key, which prints an underline graphic. Hold these two keys down until the word is underlined.

Exercise 5: Half Bar

Step 1: Hold down the **C=** key with one hand during the whole exercise.
    2: Press D, then I, then I, then F.
    3: Press the **RETURN** key.

Exercise 6: Wedge

Step 1: Hold down the **C=** key and hit T, then Y, then U.
    2: Hold down the **CONTROL** key and hit 9.
    3: Hold down the **C=** key and hit I, then O, then P, then @, then SPACE.
    4: Press the **RETURN** key.

Exercise 7: Window

Step 1: Hold down the **C͟** key until step 4.
    2: Press A, then R, then S, then RETURN.
    3: Press Q.
    4: Let go of **C͟** (it's OK, honest).
    5: Hold down **SHIFT** and hit +.
    6: Let go of **SHIFT** and hold down **C͟** again. Don't let go of it any more.
    7: Press W, then **RETURN**.
    8: Press Z, then E, then X, then **RETURN**

The purpose of these exercises is to show you how the graphic symbols of the Plus/4 can be manipulated to create different shapes and figures. These are only a handful of the figures and representations you can develop. Now that you have a good idea of what is involved in using the graphic symbols to build different forms, you should experiment with them yourself, and see what you come up with.

CHARACTER ANIMATION

Movies are really a sequence of still pictures. Each picture is a little different from the one that came before. The projector shows each picture for a very short time, then goes on to the next one. The scene becomes animated.

Computer animation works the same way. First the computer draws one picture, then it changes the picture slightly. The Plus/4 is fast enough to allow objects to move smoothly all around the screen in your games and practical programs.

You can't type fast enough to create animation. A movie is animated at a rate of 30 pictures per second. The changes must be fast enough to fool the eye. So you must use a program to draw a picture, wait for a split second, then change to a new picture.

To get the program to create pictures we use the PRINT statement with the graphic characters. The simplest type of animation involves alternating two characters to get the effect of movement.

This program simulates the movement of a pulsing ball.

Type NEW and press **RETURN** before entering each new program. Remember to press **RETURN** after each line in all these programs.

```
10 PRINT " HOME    SHIFT  Q "
20 FOR L=1 TO 100
30 NEXT L
40 PRINT " HOME    SHIFT  W "
50 FOR L=1 TO 100
60 NEXT L
70 GOTO 10
```

Type these keys simultaneously

Type RUN and press **RETURN**.

To get a more interesting effect you can build a small picture from several graphic characters, then change a few of the characters while leaving others in the same place. This gives the effect of part of an object moving, demonstrated in the following program.

IMPORTANT TO NOTE: Each time **SHIFT** or **Cᴱ** is referred to, it should be typed at the SAME TIME as the key following it when entering the program.

```
10 PRINT " HOME   SHIFT  M  SHIFT  W  SHIFT  N "
20 PRINT "SPACE  Cᴱ + SPACE  "
30 PRINT " SHIFT  N  SPACE  SHIFT  M"
40 FOR L=1 TO 100: NEXT L
50 PRINT " HOME  SPACE  SHIFT  W SPACE"
60 PRINT " Cᴱ  T  Cᴱ + Cᴱ  T"
70 PRINT "SPACE  Cᴱ  G  Cᴱ  G"
80 FOR L=1 TO 100: NEXT L
90 GOTO 10
```

Type RUN and press **RETURN**

In both examples of animation so far, we've worked on only one area on the screen. The next step is to move the animated figure around. The TAB function helps when you want to move objects from the left edge. The following program portrays a snake crawling on the screen.

Remember that **SHIFT** and the following key are still typed together.

```
5 FOR A = 0 TO 30
10 PRINT " SHIFT   CLR  "
20 PRINT TAB (A) " SHIFT  U  SHIFT  I  SHIFT  U  SHIFT  I"
30 PRINT TAB (A) " SHIFT  K  SHIFT  J  SHIFT  K  SHIFT  J"
40 FOR L=1 TO 100: NEXT L
50 PRINT " SHIFT   CLR  "
60 PRINT TAB (A+1) " SHIFT  I  SHIFT  U  SHIFT  I  SHIFT  U "
70 PRINT TAB (A+1) " SHIFT  J  SHIFT  K  SHIFT  J  SHIFT  K "
80 FOR L=1 TO 100: NEXT L
90 NEXT A
```

Using characters like the ball (**SHIFT** Q), you can play video games on the screen. To move a ball, just erase the ball and replace it at a new position, as in this program.

```
10 PRINT " SHIFT  CLR  "
20 PRINT "SPACE SHIFT Q ▌";
30 FOR L=1 TO 50: NEXT L
40 GOTO 20
```

> Left cursor arrow
> leave a space here

Type RUN and press the **RETURN** key. Press the STOP key when
you want to stop moving the ball.

CONTROLLING COLORS

Separate colors can be put into each part of the screen. The border can be one color, the background a different one, and each character can have its own color. You already know how to set the character colors using the keyboard. The COLOR command adjusts the color of the other screen areas.

Turn the border of your screen red by typing the command COLOR 4,3 and pressing the **RETURN** key. The number 4 in the command stands for the border area, and color number 3 is red (the same number as on the key marked RED).

Now type COLOR 0, 7 and hit **RETURN** . The screen background turns blue. The number 0 stands for the background, while the 7 is blue (also the same as the keyboard).

The first number after the word COLOR stands for the area on the screen you want to change. Area 0 is the background, 1 is the character color, 4 is the border. You'll learn about areas 2 and 3 when you get into multi-color graphics later in this chapter.

Screen Area Numbers

AREA #	AREA NAME
0	Background
1	Character
2	Multi-color 1
3	Multi-color 2
4	Border

Each color also has an adjustable brightness level, called the *luminance*. You can add a number from 0 (darkest) through 7 (brightest) after the color number to vary the color. Type COLOR 4 ,3, 0 and hit **RETURN.** The border becomes a dark red. Type COLOR 4, 3, 7 and the border changes to a bright red.

Color Numbers

COLOR#	COLOR	COLOR#	COLOR
1	BLACK	9	ORANGE
2	WHITE	10	BROWN
3	RED	11	YELLOW GREEN
4	CYAN	12	PINK
5	PURPLE	13	BLUE GREEN
6	GREEN	14	LIGHT BLUE
7	BLUE	15	DARK BLUE
8	YELLOW	16	LIGHT GREEN

In short, the COLOR command looks like this:
COLOR area, color, luminance
Here is a quick program to show you all the Plus/4's colors:

First type NEW and hit **RETURN** . Don't forget to hit **RETURN**
after each program line.

```
10 COLOR 0, 7, 7
20 FOR M=0 TO 7
30 FOR N=1 TO 2
40 FOR L=1 TO 16
50 PRINT "  CONTROL    RVS ON  ";
60 READ A
70 COLOR 1, A, M
80 PRINT "      ";
90 NEXT L
100 PRINT
110 RESTORE
120 NEXT N,M
130 COLOR 1, 2, 4
200 DATA 7,14,4,13,6,16,11,8,10,9,3,12,5,15,2,1
```

Now type RUN and hit **RETURN** , to see a bright blue screen with
each of the other 15 colors shown at each luminance level.

NOTE: Like most of the BASIC graphic terms reviewed in this
chapter, COLOR may be referred to as a statement or com-
mand interchangeably. The distinction is unimportant in ex-
plaining COLOR, or any other graphic commands, since it is
based on whether the term is more often used in direct or pro-
gramming mode.

HIGH RESOLUTION GRAPHICS

Your Plus/4 screen contains 25 rows of 40 characters each, or 1000 total character positions on the screen. Each character is formed out of single dots, with 8 rows of 8 dots each making an entire character. Your screen has a total of 320 dots on each row, and 200 rows of dots, or 64,000 dots all together. Your Plus/4 has control over every single dot.

Using normal graphics, you have limited control over the individual dots. You must use the 256 characters in each character set built into the Plus/4, which lets you create many pictures. But think of how many you could create if you could control each dot by itself!

The high resolution graphics ability of the Plus/4 lets you do just that. You can use commands that let you draw and erase dots, lines, circles, and other shapes.

There is one limit to high-res graphics. The Plus/4 can only use two colors in each 8×8 character position. That is, each 8×8 space on the screen where characters usually go is limited to two colors (foreground and background color for that square). You can use different colors for each different character position, but only two colors within that position. Another graphic mode that will be covered later in this section, multi-color mode, allows up to four different colors per character position at the cost of the resolution available in the high-resolution mode.

This program utilizes some of the high resolution graphics capability of the Plus/4, in particular the GRAPHIC command. Start by typing NEW and hitting **RETURN** . Hit the **RETURN** key after typing each line. After typing in the entire program, type RUN and hit **RETURN** as usual.

```
10 COLOR 0,1
20 GRAPHIC 1,1
30 FOR L=2 TO 16
40 COLOR 1,L,2
50 DRAW 1,0,L*12 TO 319,L*12
60 DRAW 1,L*18,0 TO L*18,199
70 NEXT L
80 FOR L=1 TO 5000:NEXT
90 COLOR 1,2,3
100 GRAPHIC 0
```

Notice that the colors change near the intersections.

To switch from normal graphics (also called Text Mode) to high-res, just type the command GRAPHIC 2,1 and hit **RETURN**. What happens? The screen goes blank and the cursor reappears near the bottom of the screen. The Plus/4 divided up the screen into 2 separate sections: the top for graphics and the bottom five lines for text. If you don't want the bottom five lines for text, you can use the command GRAPHIC 1,1, but you won't be able to see any commands you type.

You can switch back and forth from graphics to text using the GRAPHIC command. The command GRAPHIC 0 switches the screen back to text, while GRAPHIC 2 switches back to high-res without erasing the screen. Adding ,1 after the command erases the screen.

In general the GRAPHIC command looks like this:

GRAPHIC mode, clear ◄━━━ | this part is optional |

Mode Number	Effect
0	Text
1	High-res
2	High-res + text
3	Multi-color
4	Multi-color + text

Clear Number	Effect
0	Don't clear screen
1	Clear screen

There is another way to clear the high-resolution screen. The command SCNCLR erases the screen without changing the graphic mode.

Once you use high-resolution graphics, the computer sets aside 10K of memory for your hi-res screen. This memory is taken from the BASIC program area. When you are through using graphics, you can reclaim this memory by using the command GRAPHIC CLR.

POINTS, LINES, AND LABELS

Type the commands GRAPHIC 2,1: DRAW 1,0,0 and hit **RETURN** Look closely at the upper left corner of the screen. The Plus/4 drew a black dot there.

In the DRAW command, the first number is either 1 (foreground color) or 0 (background). The next two numbers are for the row and column positions for the dot. So if you wanted to draw a dot at column 17, row 20, just type DRAW 1,17,20. To erase the same dot type DRAW 0,17,20.

The DRAW command can also draw a line between any two points. Just add the word TO and the coordinates of the other end, like this: DRAW 1,1,1 TO 100,100. This draws a line from 1,1 to 100,100.

If you are used to drawing graphs in math, you might get a little confused at first while using the computer. The coordinate system in the Plus/4 is different from what you're used to. In math the 0,0 point would either be at the center or the lower left corner of the screen, but on the computer it is the upper left corner. You'll get used to the system in the computer as you practice.

Once you have put a point or line on the screen, you can draw a line from it to any other point like this: DRAW 1 TO 150,50. This draws a line from the last point drawn to column 150, row 50. If your program uses a lot of DRAW TO commands, you could place the first dot at a position on the screen by using the LOCATE command, as in LOCATE 100,100.

The DRAW command can have several forms, such as:

| COMMAND	RESULT
DRAW color source, column, row	POINT
DRAW color source, column, row TO column, row	LINE
DRAW color source TO column, row	LINE DRAWN FROM LAST POINT

Color source is 0 for the background, 1 for the foreground

To erase points or lines on the screen, use the DRAW command followed by the number 0. If you created a point with DRAW 1,1,1, you can erase it with DRAW 0,1,1. A line created with DRAW 1,1,1 TO 100,100 is erased by DRAW 0,1,1 TO 100,100.

This program draws a curve based on the sine function. Type NEW and hit **RETURN**. Remember to hit the **RETURN** key after each line, then type RUN.

```
10 COLOR 0,1
20 COLOR 1,2
30 GRAPHIC 1,1
40 LOCATE 0,100
50 FOR X=1 TO 319
60 Y = INT (100+99 * SIN(X/20))
70 DRAW 1 TO X,Y
80 NEXT X
90 FOR L=1 TO 5000
100 NEXT L
110 GRAPHIC 0
```
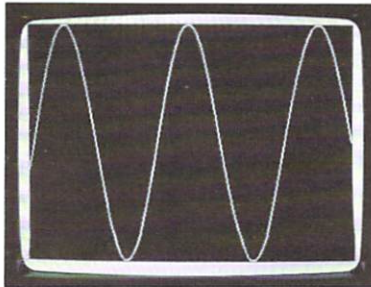

Don't type NEW after RUNning the last program. To plot the program differently, change line 70 to:

70 DRAW 1, X, Y

This program plots the same curve using points instead of lines.

The Char Command Graphs are more easily understood and useful if you label them. You can use the CHAR command to mix text right into a high resolution drawing. For instance, the statement CHAR 1,0,5,"HELLO" puts the word HELLO into the sixth row at the left edge of the screen. The first number after the word CHAR is either 1 (for draw) or 0 (for erase). The next two numbers are the column and row where the text appears.

Leave the last two programs in the computer: don't type NEW. Add these lines:

```
81 CHAR 1,0,0,"GRAPH OF":CHAR 1,0,1,"FORMULA"
82 CHAR 1,0,2,"Y=SIN(X)"
83 DRAW 1,0,100 TO 319,100,189,0 TO 189,199
84 CHAR 1,0,12,"X-AXIS": CHAR 1,22,0,"Y"
85 CHAR 1,22,2,"A": CHAR 1,22,3,"X"
86 CHAR 1,22,4,"I": CHAR 1,22,5,"S"
```
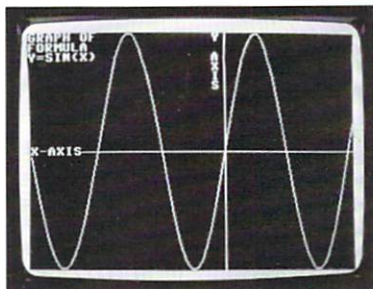
SQUARES, CIRCLES, POLYGONS, AND PAINTING

Drawing Rectangles

Using the DRAW command, you can draw pictures by plotting many dots or lines. To draw a square, you can use the command DRAW 1,0,0 TO 100,0 TO 100,100 TO 0,100 TO 0,0 (plotting each endpoint of the square) or you can just use the BOX command.

THE BOX COMMAND

Your Plus/4 includes a command to make it easier to draw squares and other rectangular shapes. The BOX command lets you pick the points of 2 opposite corners of the square. To duplicate the same box as in the above example, just use BOX 1,0,0,100,100. The number 1 again means that you want to draw and not erase. The next four numbers are the coordinates of the box's opposite corners, (0,0) at the upper-left corner and (100,100) near the middle of the screen.

The BOX command can form any rectangle just by changing the corners. You can even rotate the box by specifying an angle (in degrees) after the last coordinate, like this: BOX 1,50,50,100,100,45. The box is rotated 45 degrees clockwise.

If you would like to draw a solid box instead of just the outline, you just add a comma 1 after the angle. A solid box at the center of the screen is shown as BOX 1,100,50,220,150,,1. Notice that you need a comma to take the place for the angle, even though you don't want the box rotated.

Some typical forms of the BOX command are:

COMMAND	EFFECT
BOX on, column1, row1, column2, row2	Outline
BOX on, col1, row1, col2, row2, angle	Rotated
BOX on, col1, row1, col2, row2, , fill	Solid box
BOX off, col1, row1, col2, row2, angle, fill	Erase area of screen

Here are a couple of programs that use the BOX command:

Don't forget to type NEW then hit **RETURN** before entering each program, and press **RETURN** after typing in each line.

```
10 COLOR 0,1
20 COLOR 1,2
30 GRAPHIC 2,1
40 A =RND(1)* 20+ 10
```

```
50 FOR L=0 TO 359 STEP A
60 BOX 1, 100, 30, 220, 130, L
70 NEXT L
80 FOR L=1 TO 2000: NEXT L
90 GRAPHIC 0,1

 5 TRAP 60
10 GRAPHIC 2,1
20 DEF FNA(X)= INT( RND(1)* X)
30 COLOR 1, FNA(15)+1
40 BOX 1, FNA(320), FNA(160), FNA(320), FNA(160),, 1
50 GOTO 30
60 COLOR 1,2,3: GRAPHIC 0
```

Hit **RETURN** and type RUN after each program is completely entered. Hold down the **STOP** key to end the program.

The second program draws different colored squares all over the screen. You'll notice some parts of the screen changing when other parts near them change. The reason for this was discussed earlier in this chapter.

Drawing Circles   Your Plus/4 also has commands for drawing circles. Like the BOX command, we can vary the shape of the circle to form an oval (also called an ellipse), and we can rotate the oval. We can also just draw a section of the shape (called an arc).

This command draws a circle in the center of the screen: CIRCLE 1,160,100,50. This tells the Plus/4 to draw a circle with its center at row 160 and column 100, with a radius of 50. This actually produces an oval, since the dots on the screen are taller than they are wide. To change this to a real circle you must add a separate number to tell that the height is different from the width, like this: CIRCLE 1,160,100,50,42.

The Plus/4 can also draw a square, triangle or other polygon using the CIRCLE command. Just tell the computer how many degrees to go between points on the circle, like this: CIRCLE 1,160,100,50,42,,,120. This command draws a triangle, since each side is 120 degrees. (Omitting number values while including commas in a graphic command causes the computer to read standard default values for the missing number.) A simple formula to get the angle for a polygon with N sides is 360/N.

Here's a quick program for drawing polygons:

```
10 GRAPHIC 2,1
20 INPUT"HOW MANY SIDES";A
30 IF A<2 OR A>100 THEN PRINT "DON'T BE RIDICULOUS":
GOTO 20
40 CIRCLE 1,160,80,40,33,,,,360/A
50 GOTO 20
```

You can choose to draw only an arc instead of a whole circle.
The CIRCLE command accepts the starting and ending angles
in degrees, right after the height number. The command CIRCLE
1,160,100,50,42,90,180 displays only the lower right section of
the circle.

To rotate an oval, add the angle of clockwise rotation after the
command, like this example: CIRCLE 1,160,100,100,20,,,30.

The usual forms of the CIRCLE command are:

COMMAND	EFFECT
CIRCLE on, center column, center row, radius	oval
CIRCLE on, c-col, c-row, width, height	circle/oval
CIRCLE on, c-col, c-row, wid, ht, start, finish	arc
CIRCLE on, c-col, c-row, width, height,,,angle	rotate oval
CIRCLE on, c-col, c-row, wid, ht,,,,point angle	polygon

NOTE: When there are commas without numbers in a command
such as CIRCLE or BOX, the Plus/4 automatically interprets the
comma as an input of the default value for that parameter of the
command. For example, CIRCLE,160,40,100,100 is read by the
computer as CIRCLE 1, 160..., reading the default value of 1 for
the color source.

This next program uses the CIRCLE command for an interesting effect.
Type NEW then hit **RETURN** to erase the last program from memory
before typing in this program.

```
10 COLOR 0,1
20 COLOR 1,2
30 GRAPHIC 1,1
40 A = RND(1)* 20+ 10
50 FOR L=0 TO 359 STEP A
60 CIRCLE 1, 160, 100, 80, 40,,,L
70 NEXT L
80 FOR L=1 TO 2000: NEXT L
90 GRAPHIC 0,1
```

Here's a program you can try that uses the CIRCLE command to create
a simple design.

```
NEW
10 COLOR 0,1
20 COLOR 1,2
30 GRAPHIC 2,1
40 FOR L=1 TO 4
50 Y=50
60 IF L=2 OR L=4 THEN Y=100
70 X= L*35 + 50
80 CIRCLE 1, X, Y, 50, 42
90 NEXT L
100 PRINT "PLUS/4 CIRCLES"
```
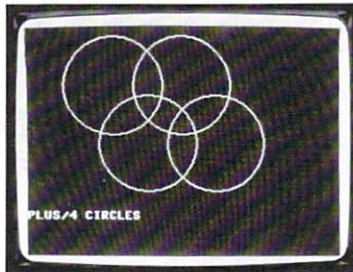
The Paint Command

The PAINT command fills in any enclosed area up to the boundaries formed by any lines drawn on the screen. If there are no drawn lines, the screen is filled right to the edge. The BOX command can be used to fill in boxes and rectangles with color. The PAINT command can color in irregular shapes and other non-uniform areas on the screen that can't be filled with other commands.

In the last exercise, we created a 4-ring symbol. By adding some PAINT commands to the program we can color in only the areas between the rings.

Add these lines to the last program:

```
110 FOR L=0 TO 1
120 PAINT 1,120 + 35 *L, 75
130 NEXT L
```

MULTI-COLOR GRAPHICS

The Plus/4 high resolution graphics give you control over every single dot or "pixel" on the screen, but you have seen that the ability to put colors close together is limited. Most high-res programs can use only one or two colors.

For including more different colors, your Plus/4 has a special "in-between" graphics mode called multi-color graphics. In multi-color graphics, you control half as many dots on each row as in high-res because each dot is twice as wide. You get 160 dots on each row, while still getting 200 rows. There is a trade-off for the use of multiple colors, which is slightly lower resolution.

To begin using multi-color graphics, review the GRAPHIC command earlier in this chapter. You'll see that the multi-color screen without text is GRAPHIC 3 and the multi-color screen with 5 lines of text is GRAPHIC 4.

Now look at the table listing the COLOR command. There are two areas that we haven't used yet, areas 2 and 3. These areas hold two extra colors. You can use any of the three colors (1, the text color; 2, an extra color; and 3, another extra color). These colors do not interfere with each other on the screen the way the high-res colors do in some previous programs in this chapter.

These two programs make use of multi-color graphics, the first with the rings and the second showing a "neon sign" effect.

```
10 COLOR 0,1
20 GRAPHIC 4,1
30 FOR L=1 TO 4
40 Q=L: IF Q>3 THEN Q=Q-3
50 COLOR Q,L+1
60 Y=50
70 IF L=2 OR L=4 THEN Y=100
80 X= L*18 + 25
90 CIRCLE Q, X, Y, 25, 42
100 NEXT L
```

Type NEW then hit **RETURN**. Don't forget to hit the **RETURN** key after each line. Type RUN and hit **RETURN** at the end.

```
10 COLOR 0,1
20 GRAPHIC 3,1
30 COLOR 3,1
40 TRAP 200
50 DRAW 3,10,10 TO 10,100: DRAW 3,10,55 TO 30,55
60 DRAW 3,30,10 TO 30,100: DRAW 3,50,10 TO 80,10
70 DRAW 3,65,10 TO 65,100: DRAW 3,50,100 TO 80,100
80 FOR L=0 TO 7
90 COLOR 3,2,L
100 FOR M=1 TO 100: NEXT M
110 NEXT L
120 COLOR 3,1
130 FOR M=1 TO 100: NEXT M
140 GOTO 80
200 GRAPHIC 0: COLOR 1,2,7
```

Color area 3, the second of the multicolor areas, has a special ability that none of the others has. Once you have drawn on the screen using area 3, you can change that color *everywhere* it appears on the screen by using the COLOR command. If you set the color with COLOR 3,5 and draw using that color, your graphics appear in purple. If you then change the color with COLOR 3,6, all the purple areas would change to green. This doesn't work with any other area.

The Plus/4 Programmer's Reference Guide contains more information about graphics.
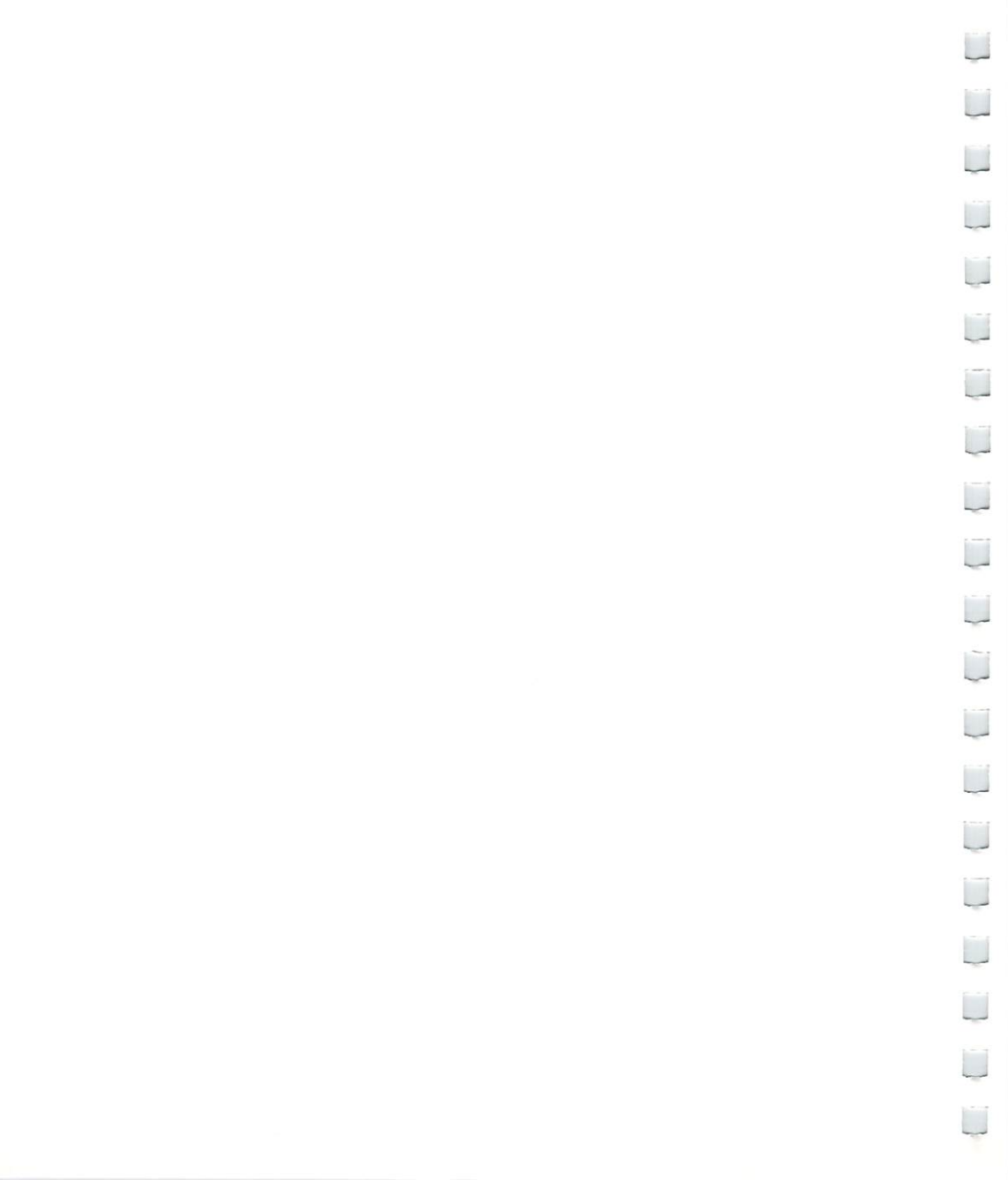
CHAPTER 8

MAKING SOUND AND MUSIC ON THE PLUS/4

INTRODUCTION

Here is a short program to make music on your Plus/4. Type in the program exactly as it appears, and remember to press **RETURN** at the end of each line. When the program is typed in, type RUN and then press **RETURN** . When the program asks you to enter a number, type any number from 0 to 1023 and press **RETURN** . To stop the program, enter a zero as your value.

```
10 VOL 8
20 DO
30 INPUT X
35 IF X>1023 OR X<0 THEN PRINT "0 TO 1023, PLEASE": GOTO 30
40 SOUND 1, X, 10
50 LOOP UNTIL X=0
```

Press the **RUN/STOP** key to stop the program.

Here's how to play a single note on your Plus/4. Type NEW and press **RETURN** to clear the Plus/4's memory.

First: Type NEW and press **RETURN**
       Type **VOL** 8 and press **RETURN**

Second: Type **SOUND 1,266,60** and press **RETURN**

You should hear a note play for about a second and then stop. If you don't hear anything, turn up the volume of your television or monitor and try it again.

These two steps are the only commands that you need to know to play music on your Plus/4. Let's look at what these two commands do.

**THE
VOLUME
COMMAND**

The VOL command controls the VOLume of the notes that the Plus/4 plays. Think of the first three letters of the word "volume" to remember the VOL command. The number that comes after VOL is the setting for the volume. Think of the VOL command as a volume knob on the Plus/4. When the knob points to zero, the volume is off and you won't hear anything. When the knob is set at 8, the volume is turned up all the way, and your Plus/4 plays as loud as it can.

Try the first example again and use a different number after the VOL command. The smaller the number, the softer the note is played.

THE SOUND COMMAND

The SOUND command tells your Plus/4 everything it needs to know about the sound you want to play. The SOUND command is followed by three numbers that describe the note:

SOUND voice, note value, duration

The first number in the sound command refers to voice. The number for voice can be a 1,2 or 3. The Plus/4 sound is produced by two different "voices", 1 for the first voice and 2 for the second voice. The third voice option applies to voice 2's ability to produce either a tone or noise.
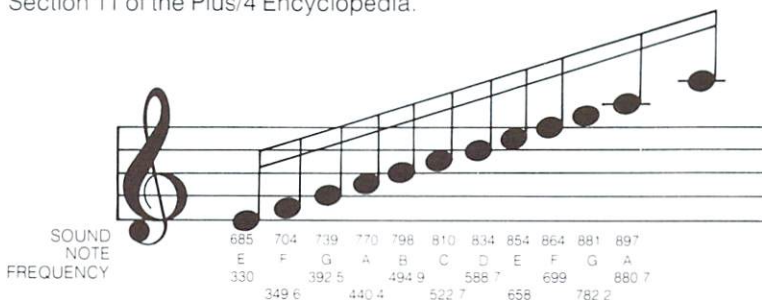
Voice 1—This voice plays only tones. Select this voice with a 1 after the SOUND command.

Voice 2—This voice is like voice 1, but can be used to play tones or noise for sounds. Type a 2 in the command to use this voice for tones, or a 3 to use this voice for noise, to make sound effects like thunder and rain.

The second number after the word SOUND is the note value (frequency). This can be any number from 0 to 1023. It tells your Plus/4 how low- or high-pitched a note to play. As the numbers get larger, the notes get higher. The highest values (in the 1023 neighborhood) are not audible to the human ear.

Note: Noise is "white" only in the range of 600-940 with Voice 3. You can use register values outside this range to create interesting sound effects.

Here is a chart that shows all of the notes in one scale, along with the note value to use. There is a complete chart of notes for the Plus/4 in Section 11 of the Plus/4 Encyclopedia.

SOUND NOTE		685	704	739	770	798	810	834	854	864	881	897
		E	F	G	A	B	C	D	E	F	G	A
FREQUENCY		330		392 5		494 9		588 7		699		880 7
			349 6		440 4		522 7		658		782 2	

Try the following program on your Plus/4:

NEW

```
10 VOL 7            ◄──────  sets VOLume
20 X = 0
30 DO
40 SOUND 1,X,5      ◄──────  plays note
50 X = X + 5
60 LOOP UNTIL X = 1020
70 VOL 0            ◄──────  turns off VOLume
80 END
```
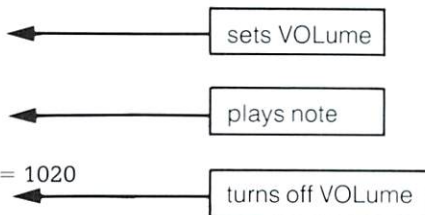
Type RUN and press **RETURN**. This program allows your Plus/4 to show off some of its musical range.

The third number after the word SOUND controls the duration (length) of the note. This tells the Plus/4 how long to play the note. This number can be anything from 0 to 65535. This number sets a timer, which counts time in sixtieths of a second. A duration of 60 keeps the note on for one second. A rule of thumb for duration is the larger the number, the longer the note stays on. In fact, if you use 65535, the note stays on for over 16 minutes. To turn a sound off, use a zero duration, which does not allow the sound to be produced.

A MUSICAL SOUND EFFECT

Sound effects can be created using either musical tones or noise. Combining simple BASIC programs and sound commands can generate unusual and entertaining effects. For instance, the FOR...NEXT... STEP loop can be used creatively in sound effects. This program uses a FOR...NEXT loop with a negative step, to count down from a high number to a lower one.

```
10 VOL 8
20 FOR S=1000 TO 700 STEP −25
30 SOUND 1, S, 1
40 NEXT S
```

sets VOLume at 8

creates loop, with downward STEPs

Type RUN and press **RETURN** to hear the sound effect. The key is line 20, which selects a number range from 1000 to 700 going down the scale, STEPping down 25 numbers at a time. Finally, line 30 instructs your Plus/4 to play each note for just an instant by setting the DURATION to 1, which is 1/60 of a second. Experimenting with different number and duration values can give you some very interesting effects.

CREATING A NOISE SOUND EFFECT

Using a value of 3 when selecting a voice in the SOUND command specifies noise. This is used to create sound effects with noise rather than tone. The following program uses voice 3 to create the sounds of a windstorm.

```
10 VOL 2                        ◄———  sets VOLume level
20 R=INT(RND(0)*10)+1
30 FOR X=1 TO R                       selects RaNDom
40 SOUND 3, 600+30*X, 10              number from 1 to 10
50 NEXT X
60 FOR X=R TO 1 STEP −1
70 SOUND, 3,600+30*X, 10
80 NEXT X
90 T=INT(RND(0)*100)+30
100 SOUND 3, 600, T
110 GOTO 20
```

Lines 30 and 60 set up loops for the note value (frequency) of the sound, one increasing and one decreasing, based on the random number from line 20. It is important to have variation in pitch, since windstorms have different forces of gusts of wind. Lines 40 and 70 are the SOUND commands that create the noise. Lines 90 and 100 set up a random delay to recreate the uneven nature of a windstorm with time lapses between howls. The program selects a RaNDom number that is used for the duration of another SOUND command. This SOUND command stays at the same pitch and provides a consistent background noise that serves as a counterpoint to the gusts of wind.

Creating sound effects using noise is extremely challenging, trying to capture the right elements of the sound you want exactly. To create good sound effects, you have to be willing to experiment.

Now that we've looked at some ways to create sound effects on your Plus/4, let's make some music. Here are a couple of programs to try:

The first program turns the keys from 1 through 8 into a piano. Type in the program and then type RUN.

```
5 SCNCLR                                        ← clears the screen
10 FOR X=1 TO 8: READ N(X): NEXT X
20 VOL 7
30 DO
40 GET A$: IF A$=" " THEN 40
50 A=ASC(A$): IF A<49 OR A>56 THEN 90
60 N=A-48
70 SOUND 1, N(N), 5
80 COLOR 0, N, 3
90 LOOP UNTIL A=32
100 VOL 0: COLOR 4, 2, 7
110 DATA 169, 262, 345, 383, 453, 516, 571, 596
```

Press numbers 1 through 8 to play notes. The screen border even changes colors with the different notes. When you finish playing, press the space bar to stop the program.

Here are the numbers to press for a familiar song:

Twinkle, Twinkle Little Star

```
1   1   5   5   6   6   5
4   4   3   3   2   2   1
5   5   4   4   3   3   2
5   5   4   4   3   3   2
1   1   5   5   6   6   5
4   4   3   3   2   2   1
```
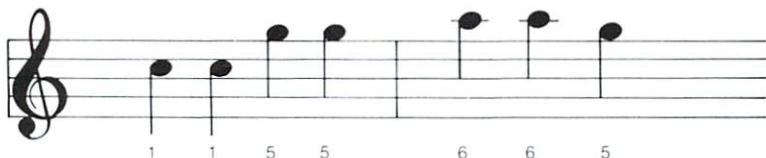
This program plays a song by reading a list of DATA statements. The DATA statements are in pairs. The first number is the note value for the SOUND command and the second number is the duration for the SOUND command.

Row Boat

```
10 VOL 8
20 DO
30 READ X, Y
40 SOUND 1, X, Y
45 FOR D=1 TO 550:NEXT
50 LOOP UNTIL X=0
60 END
100 DATA 169, 45, 169, 45, 169, 30
110 DATA 262, 15, 345, 45, 345, 30
120 DATA 262, 15, 345, 30, 383, 15
130 DATA 453, 60, 596, 45, 453, 45
140 DATA 345, 45, 169, 45, 453, 30
150 DATA 383, 15, 345, 30, 262, 15
160 DATA 169, 60
200 DATA 0, 0
```

This loop creates a brief delay between notes

This program plays notes going up and down scales at different speeds, and displays some color bars along with them.

```
10 VOL 8
20 DO
30 D=INT (RND(0) *5)+ 2: REM DURATION
40 S=INT (RND(0) *300)+ 700 : REM START
50 R=INT (RND(0) *(1020−S)): REM RANGE
60 P=INT (RND(0) *30)+ 5: REM STEP
70 T=SGN (RND(1) − .5): IF T=0 THEN 70
80 FOR Z=S TO S + T*R STEP P*T
90 SOUND 1, Z, D
100 Y=(Z AND 15) + 1: FOR X=1 TO D
110 PRINT CHR$(18);:COLOR 1, Y: PRINT "   ";
120 NEXT X, Z
130 LOOP
```

these REMark statements help you keep track of which line does what

leave a space here

THE GREAT PLUS/4 MUSIC MACHINE

The last program is a little longer. This is the "GREAT PLUS/4 MUSIC MACHINE". When you press a key from 1 through 9, the note is played, and a note appears on the staff on the correct line.

```
5 GOSUB 1000
6 FOR X=1 TO 9: READ N(X): NEXT X
8 CHAR 1, 8, 1,"*THE GREAT MUSIC MACHINE*"
10 VOL 7
20 DO
30 GET A$: IF A$="" THEN 30
35 A=ASC(A$): IF A<49 OR A>57 THEN 50
36 N= A − 48
40 SOUND 1, N(N), 4
45 GSHAPE N$, 150, 8 * (6+(9−N)), 4
46 FOR Z=1 TO 50: NEXT Z
47 GSHAPE N$, 150, 8 * (6+(9−N)), 4
50 LOOP UNTIL A=32
55 VOL 0: GRAPHIC 0: SCNCLR
60 END
100 DATA 345, 383, 453, 516, 571, 596, 643, 685, 704
1000 GRAPHIC 1,1
1010 FOR Y=60 TO 124 STEP 16
1020 DRAW 1, 100, Y TO 200, Y
1030 NEXT Y
1040 A$="FEDCBAGFE"
1050 FOR X=1 TO 9: C=13
1060 IF INT(X/2)= X/2 THEN C=14
1070 CHAR 1, C, X+6, MID$ (A$, X, 1), 0
1075 CHAR 1, C+10, X+6, RIGHT$ (STR$ (10−X), 1)
1080 NEXT X
1090 FOR X=1 TO 8: FOR Y=11 TO 16: DRAW 1, X, Y: NEXT Y, X
1100 Y=1: X=8: DRAW 1, 8, 16 TO X, Y
1110 SSHAPE N$, 1, 1, 8, 16
1120 GSHAPE N$, 1, 1, 4
1130 RETURN
```

(arrow pointing to line 20 DO) no space

As you can see, it's not hard to write your own sound programs. The ones in this chapter just give a taste of the music capabilities of your Plus/4. Don't be afraid to try new sounds and noises and create your own masterpiece.

PLUS/4 ENCYCLOPEDIA

The Plus/4 Encyclopedia contains information useful for both the computer novice and expert. Some sections are musts for beginners, like the BASIC 3.5 Encyclopedia, which lists and explains all the BASIC commands, statements and terminology. Other sections will be useful for those computerists whose knowledge extends beyond BASIC. TEDMON, the Plus/4's machine language monitor, provides some direction for machine language programming on the Plus/4. Other sections can be helpful for all Plus/4 owners . . . listings of error messages, programs to try out, the musical note table, and more. The Plus/4 Encyclopedia has something for all Plus/4 programmers, whether you're just beginning to experiment or you're a high-level programmer.

This manual has given you an introduction to the BASIC language, to give you a feel for computer programming and some of the vocabulary involved. This encyclopedia gives a complete list of the rules (SYNTAX) of the BASIC 3.5 language, along with a concise description of each. Experiment with these commands, and remember that you can't damage your Plus/4 by typing in programs, and that the best way to learn computing is by doing.

The encyclopedia provides formats and brief explanations and examples of the BASIC 3.5 commands and statements. It is not intended to teach BASIC. If you are interested in learning BASIC, Section 14 lists tutorial books that will help.

Commands and statements are listed in separate sections. Within the sections, the commands and statements are listed in alphabetical order. Commands are used mainly in direct mode, while statements are most often used in programs. In most cases, commands can be used as statements in a program if you prefix them with a line number. You are able to use many statements as commands by issuing them in direct mode (i.e., without line numbers).

The different types of operations in BASIC are listed in sections based on the following criteria:

- COMMANDS: the commands used to work with programs, edit, store, and erase them.
- STATEMENTS: the BASIC program statements used in numbered lines of programs.
- FUNCTIONS: the string, numeric, and print functions.
- VARIABLES AND OPERATORS: the different types of variables, legal variable names, and arithmetic and logical operators.

A fuller explanation of BASIC 3.5 commands is provided in the Plus/4 Programmer's Reference Guide, available from your Commodore dealer or your local bookstore.

The commands and statements presented in this section of the ency-
clopedia are governed by consistent format conventions designed to
make them as clear as possible. In most cases, there are several ac-
tual examples to illustrate what the actual command looks like. The
following example shows some of the format conventions that are
used in the BASIC commands and statements:

EXAMPLE: **LOAD**"*program name*",**D0, U8**   additional arguments
                                                    [possibly optional]
                 keywords   argument

The parts of the command or statement that you must type in exactly as
they appear are highlighted in boldface type in the format listing, while
the name of the command is in capital letters. The words that you don't
type in exactly, such as the name of a program, are printed in italics.
When quote marks (" ") appear (usually around a program or file
name), you should include them in the appropriate place according
to the format example.

- KEYWORDS, also called RESERVED WORDS, appear in upper-
  case letters and boldface type. YOU MUST ENTER THESE
  KEYWORDS EXACTLY AS THEY APPEAR. However, many key-
  words have abbreviations that you can also use (see Section 2).

Keywords are words that are part of the BASIC language that your
computer knows. Keywords are the central part of a command or
statement. They tell the computer what kind of action you want it to
take. These words cannot be used as variable names.

- ARGUMENTS (also called parameters) appear in lower-case italics.
  Arguments are the parts of a command or statement that you select;
  they complement keywords by providing specific information about
  the command or statement. For example, a keyword tells the com-
  puter to load a program, while an argument tells the computer which
  specific program to load and a second argument specifies which
  drive the disk containing the program is in. Arguments include
  filenames, variables, line numbers, etc.

- SQUARE BRACKETS [] show OPTIONAL arguments. You
  select any or none of the arguments listed, depending on
  your requirements.

- ANGLE BRACKETS <> indicate that you MUST choose one
  of the arguments listed.

- VERTICAL BAR   separates items in a list of arguments when your choices are limited to those arguments listed, and you can't use any other arguments. When the vertical bar appears in a list enclosed in SQUARE BRACKETS, your choices are limited to the items in the list, but you still have the option not to use any arguments.

- ELLIPSIS ..., a sequence of three dots, means that an option or argument can be repeated more than once.

- QUOTATION MARKS " " enclose character strings, filenames, and other expressions. When arguments are enclosed in quotation marks in a format, you must include the quotation marks in your command or statement. Quotation marks are not conventions used to describe formats; they are required parts of a command or statement.

- PARENTHESES () When arguments are enclosed in parentheses in a format, you must include the parentheses in your command or statement. Parentheses are not conventions used to describe formats; they are required parts of a command or statement.

- VARIABLE refers to any valid BASIC variable name, such as X, A$, or T%.

- EXPRESSION means any valid BASIC expression, such as A+B+2 or .5*(X+3).

BASIC COMMANDS

AUTO

AUTO [line#]

Turns on the automatic line numbering feature which eases the job of entering programs by typing the line numbers for you. As you enter each program line and press **RETURN** the next line number is printed on the screen, with the cursor in position to begin typing that line. The [line#] argument refers to the increment between line numbers. AUTO with NO ARGUMENT turns off auto line numbering, as does RUN. This statement is executable only in direct mode.

EXAMPLE:

AUTO 10 ◄——— automatically numbers line in increments of ten

AUTO 50 ◄——— automatically numbers line in increments of fifty

AUTO ◄——— turns OFF automatic line numbering

BACKUP

BACKUP Ddrive# TO Ddrive# [, ON Uunit#]

This command copies all the files on a diskette to another diskette on a dual drive system. You can copy onto a new diskette without first using the HEADER command to format the new diskette because the BACKUP command copies all the information on the diskette, including the format. You should always BACKUP important diskettes in case the original is lost or damaged.

Because the BACKUP command also HEADERS diskettes, it destroys any information on the diskette onto which you're copying information. So if you're backing up onto a previously used diskette, make sure it contains no programs you wish to keep. See also the COPY command.

NOTE: This command can only be used with dual disk drives.

EXAMPLE:

BACKUP D0 TO D1 ◄——— Copies all files from the disk in drive 0 to the disk in drive 1

BACKUP D0 TO D1, ON U9 ◄——— Copies all files from drive 0 to drive 1 in disk drive unit 9

COLLECT   COLLECT [Ddrive #] [,ON Uunit #]

Use this command to free up space allocated to improperly closed files and delete references to these files from the directory.

EXAMPLE:

COLLECT D0

CONT   CONT
(Continue)

This command is used to re-start the execution of a program that has been stopped by either using the STOP key, a STOP statement, or an END statement within the program. The program will resume execution where it left off. CONT will not work if you have changed or added lines of the program (or even just moved the cursor to a program line and hit **RETURN** without changing anything), if the program stopped due to an error, or if you caused an error before trying to re-start the program. The error message in this case is CAN'T CONTINUE ERROR.

COPY   COPY [Ddrive #,] "source file" TO [Ddrive #,] "other file" [,ON Uunit #]

COPYs a file on the disk in one drive (the source file) to the disk in the other on a dual disk drive only, or creates a copy of a file on the same drive (with a different file name).

EXAMPLES:

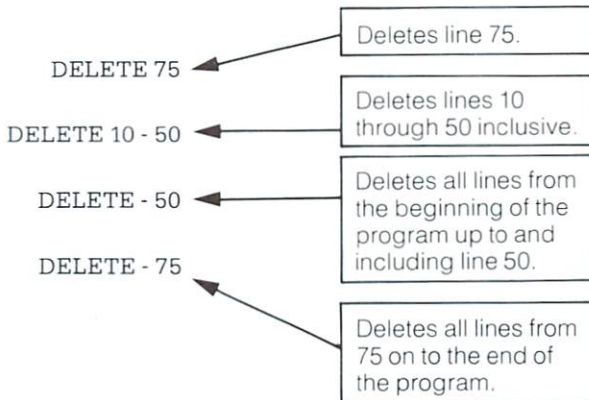|	
COPY D0, "test" to D1, "test prog" ◄	Copies "test" from drive 0 to drive 1, renaming it "test prog" on drive 1.
COPY D0, "STUFF" TO D1, "STUFF" ◄	Copies "STUFF" from drive 0 to drive 1.
COPY D0 TO D1 ◄	Copies all files from drive 0 to drive 1.
COPY "WORK.PROG" TO "BACKUP" ◄	Copies "WORK.PROG" as a program called "BACKUP" on the same drive.

DELETE   DELETE [first line #] [– last line #]

Deletes lines of BASIC text. This command can be executed only in direct mode.

EXAMPLES:

DELETE 75 — Deletes line 75.

DELETE 10 - 50 — Deletes lines 10 through 50 inclusive.

DELETE - 50 — Deletes all lines from the beginning of the program up to and including line 50.

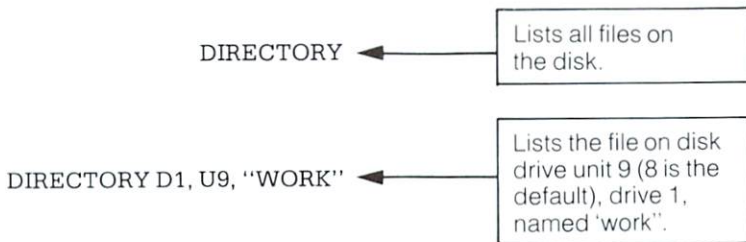DELETE - 75 — Deletes all lines from 75 on to the end of the program.

DIRECTORY   DIRECTORY [Ddrive #] [,Uunit #] [,"filename"]

Displays a disk directory on the Plus/4 screen. Use **CTRL** -S to pause the display (any other key restarts the display after a pause). Use the **C=** key (the Commodore key) to slow it down. The DIRECTORY command cannot be used to print a hard copy. You must load the disk directory (destroying the program currently in memory) to do that.

EXAMPLES:

DIRECTORY — Lists all files on the disk.

DIRECTORY D1, U9, "WORK" — Lists the file on disk drive unit 9 (8 is the default), drive 1, named 'work".

120

DIRECTORY "AB*" ◀──

> Lists all files start-
> ing with the letters
> "AB", like ABOVE,
> ABOARD, etc.

DIRECTORY D0, "FILE ?.BAK" ◀──

> The ? is a wild-card
> that matches any
> single character in
> that position: FILE
> 1.BAK, FILE 2.BAK,
> FILE 3.BAK all
> match the string.

NOTE: To print out the DIRECTORY of drive 0, unit 8, use
the following:

```
LOAD"$0",8
OPEN4,4:CMD4:LIST
PRINT#4:CLOSE4
```

DLOAD

DLOAD *"filename"* [,D*drive #*] [,U*unit #*]

This command loads a program from disk into current memory. (Use
LOAD to load programs on tape.) You must supply a program name.

EXAMPLES:

DLOAD "BANKRECS" ◀──

> Searches the disk
> for the program
> "BANKRECS" and
> LOADs it.

DLOAD (A$) ◀──

> LOADs a program
> from disk whose
> name is in the variable
> A$. You will get an
> error if A$ is empty.

The DLOAD command can be used within a BASIC program to find
and RUN another program on disk. This is called chaining.

DSAVE

DSAVE *"filename"* [,D*drive #*] [,U*unit #*]

This command stores a program on disk. (Use SAVE to store programs on tape.) You must supply a program name.

EXAMPLES:

DSAVE "BANKRECS" ◀─── | SAVEs the program "BANKRECS" to disk.

DSAVE (A$) ◀─── | SAVEs to disk program whose name is in the variable A$.

DSAVE "PROG 3", D0,U9 ◀─── | SAVEs the program "PROG 3" to the disk drive with a unit number (Serial bus address) of 9.

HEADER

HEADER *"diskname"* [,I*i.d. #*] ,D*drive #* [,ON U*unit #*]

Before you can use a new diskette for the first time you must format it with the HEADER command. If you want to erase an entire diskette for reuse you can use the HEADER command. This command divides the disk into sections called blocks, and it creates a table of contents, called a directory or catalog, on the disk. The *diskname* can be any name up to 16 characters long. The *i.d.* number is any 2 characters. Give each disk a unique i.d. number. Be careful when you HEADER a disk because the HEADER command erases all stored data. Giving no i.d. number allows you to perform a quick header. The old i.d. number is used. You can only use the quick header method if the disk was previously formatted, since the quick header only clears out the directory rather than formatting the disk.

EXAMPLES:

HEADER "MYDISK", I23, D1
HEADER "RECS", I45, D1, U8

HELP HELP

The HELP command is used after you get an error in your program.
When you type HELP, the line where the error occurred is listed, with
the portion containing the error displayed in flashing characters.

KEY KEY [key #, string]

There are eight (8) function keys available to the user on the Plus/4
computer: four unshifted and four shifted. The Plus/4 allows you to
define what each key does when pressed.

KEY without any parameter specified gives a listing displaying all the
current KEY assignments. The data you assign to a key is typed out
when that function key is pressed. The maximum length for all the
definitions together is 128 characters. Entire commands (or a
series of commands) can be assigned to a key. For example:

KEY 7, "GRAPHIC0" + CHR$(13) + "LIST" + CHR$(13)

causes the computer to select text mode and list your program
whenever the 'F7' key is depressed (in direct mode). The CHR$(13) is
the ASCII character for **RETURN** . Use CHR$(34) to incorporate a
double quote into a KEY string.

The keys may be redefined in a program. For example:

10 KEY 2,"TESTING" + CHR$(34):KEY3, "NO"
10 FORI=1 TO 8:KEY I, CHR$(I+132):NEXT

> defines the function
> keys as they are de-
> fined on the Commo-
> dore 64 and VIC 20

To restore all function keys to their default values, reset the Plus/4 by
turning it off and on, or press the RESET button.

LIST

LIST [first line #] [-[last line #]]

The LIST command lets you look at lines of a BASIC program that have been typed or LOADed into the Plus/4'S memory. When used alone (without any numbers following it), the Plus/4 gives a complete LISTing of the program on your screen, which may be slowed down by holding down the **C=** key, paused by **CTRL** S (unpaused by pressing any other key), or STOPped by hitting the **RUN/STOP** key. If you follow the word LIST with a line number, the Plus/4 only shows that line number. If you type LIST with two numbers separated by a dash, the Plus/4 shows all lines from the first to the second line number. If you type LIST followed by a number and just a dash, it shows all the lines from that number to the end of the program. And if you type LIST, a dash, and then a number, you get all the lines from the beginning of the program to that line number. Using these variations, you can examine any portion of a program, or easily bring lines to the screen for modification.

EXAMPLES:

Command	Description
LIST	Shows entire program.
LIST 100-	Shows from line 100 until the end of the program.
LIST 10	Shows only line 10.
LIST-100	Shows lines from the beginning until line 100.
LIST 10-200	Shows lines from 10 to 200, inclusive.

LOAD

LOAD ["filename"] [,device #] [,relocate flag]

This is the command to use when you want to use a program stored on cassette tape or on disk. If you type just LOAD and hit the **RETURN** key, the Plus/4 screen goes blank. Press play, and the Plus/4 starts
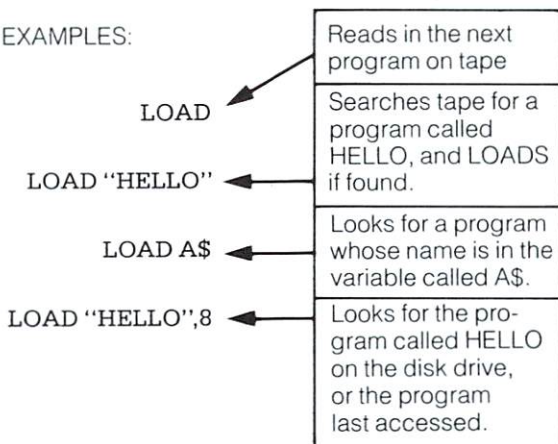
looking for a program on the tape. When it finds one, the Plus/4 prints FOUND *"filename"*. You can hit the ⬛ C= ⬛ key to LOAD. Once the program is LOADed, you can RUN, LIST, or change it.

You can also type the word LOAD followed by a program name, which is most often a name in quotes (*"program name"*). The name may be followed by a comma (outside of the quotes) and a number (or numeric variable), which acts as a device number to determine where the program is stored (disk or tape). If there is no number given, the Plus/4 assumes device number 1, which is the cassette tape recorder.

The other device commonly used with the LOAD command is usually the disk drive, which is device number 8.

EXAMPLES:

LOAD	Reads in the next program on tape
LOAD "HELLO"	Searches tape for a program called HELLO, and LOADS if found.
LOAD A$	Looks for a program whose name is in the variable called A$.
LOAD "HELLO",8	Looks for the program called HELLO on the disk drive, or the program last accessed.

The LOAD command can be used within a BASIC program to find and RUN the next program on a tape. This is called chaining.

The RELOCATE FLAG determines where in memory a program is loaded. A relocate flag of 0 tells the Plus/4 to load the program at the start of the BASIC program area, and a flag of 1 tells it to LOAD from the point where it was SAVEd. The default value of the relocate flag is 0. This is generally used only when loading machine language programs.

NEW

NEW

This command erases the entire program in memory and clears out any variables that may have been used. Unless the program was stored somewhere, it is lost until you type it in again. Be careful when you use this command.

The NEW command can also be used as a statement in a BASIC program. When the Plus/4 gets to this line, the program is erased and everything stops. This is not especially useful under normal circumstances.

RENAME

RENAME [,D*drive #*] "*old name*" TO "*new name*" [,U*unit #*]

Used to rename a file on a diskette.

EXAMPLE:

RENAME "TEST" TO "FINALTEST"

> Changes the name of the file "TEST" to "FINALTEST"

RENUMBER

RENUMBER [*new starting line #* [,*increment* [,*old starting line #*]]]

The new starting line is the number of the first line in the program after renumbering. It defaults to 10.

The increment is the spacing between line numbers, i.e. 10, 20, 30 etc. It also defaults to 10.

The old starting line number is the line number in the program where renumbering is to begin. This allows you to renumber a portion of your program. It defaults to the first line of your program.

This command can only be executed from direct mode.

EXAMPLES:

RENUMBER 20, 20, 1 ◄

> Starting at line 65, renumbers in increments of 10. Line 65 becomes line 10.

RENUMBER, , 65 ◄───── Starting at line 1,
renumbers the
program. Line 1
becomes line 20,
and other lines are
numbered in incre-
ments of 20.

RUN

RUN [*line #*]

Once a program has been typed into memory or LOADed, the RUN
command makes it start working. RUN clears all variables in the pro-
gram before starting program execution. If there is no number follow-
ing the command RUN, the computer starts with the lowest numbered
program line. If there is a number following the RUN command execu-
tion starts at that line. RUN may be used within a program.

EXAMPLES:

RUN ◄───── Starts program
working from lowest
line number.

RUN 100 ◄───── Starts program at
line 100.

SAVE

SAVE [*"filename"* [*,device #* [*,EOT flag*]]]

This command stores a program currently in memory onto a cassette
tape or disk. If you just type the word SAVE and hit **RETURN**
the Plus/4 attempts to store the program on the cassette tape. It has no
way of checking if there is already a program on the tape in that loca-
tion, so be careful with your tapes. If you type the SAVE command fol-
lowed by a name in quotes or a string variable name, the Plus/4 gives
the program that name, so it may be more easily located and retrieved
in the future. If you want to specify a device number for the SAVE, fol-
low the name by a comma (after the quotes) and a number or numeric
variable. Device number 1 is the tape drive, and number 8 is the disk.
After the number on a tape command, there can be a comma and a
second number, which is either 0 or 1. If the second number is 1, the
Plus/4 puts an END-OF-TAPE marker (EOT flag) after your program. If

trying to LOAD a program and the Plus/4 finds one of these markers rather than the program you are trying to LOAD, you get a FILE NOT FOUND ERROR.

EXAMPLES:

SAVE	Stores program to tape without a name.
SAVE "HELLO"	Stores on tape with the name HELLO
SAVE A$	Stores on tape with name in variable A$
SAVE "HELLO", 8	Stores on disk with name HELLO
SAVE "HELLO", 1, 2	Stores on tape with name HELLO and places an END-OF-TAPE marker after the program.

SCRATCH

SCRATCH "file name" [,D drive #] [,U unit #]

Deletes a file from the disk directory. As a precaution, you are asked "Are you sure" before the Plus/4 completes the operation. Type a Y to perform the SCRATCH or type N to cancel the operation. Use this command to erase unwanted files, to create more space on the disk.

EXAMPLE:

SCRATCH "MY BACK", D1	Erases the file MY BACK from the disk in drive 1

VERIFY

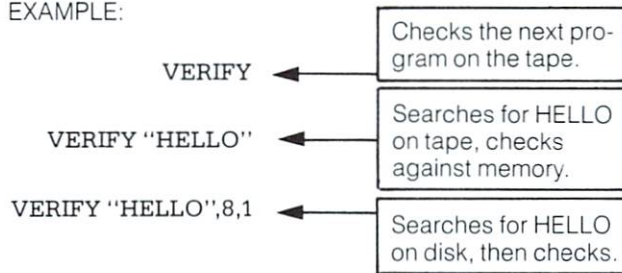VERIFY "filename" [,device #] [,relocate flag]

This command causes the Plus/4 to check the program on tape or disk against the one in memory. This is proof that the program you just SAVEd is really saved, in case your tape is bad or something isn't working. This command is also very useful for positioning a tape so that Plus/4 writes after the last program on the tape. All you do is tell the

Plus/4 to VERIFY the name of the last program on the tape. It will do so, and tell you that the programs don't match (which you already knew). Now the tape is where you want it, and you can store the next program without fear of erasing an old one.

VERIFY without anything after the command causes the Plus/4 to check the next program on tape, regardless of its name, against the program now in memory. VERIFY followed by a program name (in quotes) or a string variable searches the tape for that program and then checks. VERIFY followed by a name and a comma and a number checks the program on the device with that number (1 for tape, 8 for disk). The relocate flag is the same as in the LOAD command.

EXAMPLE:

VERIFY ◀	Checks the next program on the tape.	
VERIFY "HELLO" ◀	Searches for HELLO on tape, checks against memory.	
VERIFY "HELLO",8,1 ◀	Searches for HELLO on disk, then checks.	

BOX

BOX [color source #],a1, b1, a2, b2, [[,angle] [,paint]]

color source	Color source (0-3); default is 1 (foreground color)
a1, b1	Corner coordinate (scaled)
a2, b2	Corner opposite a1, b1 (scaled); default is the PC
angle	Rotation in clockwise degrees; default is 0 degrees
paint	Paint shape with color (0:off, 1:on); default is 0

This command allows you to draw a rectangle of any size anywhere on the screen. To get the default value, include a comma without entering a value. Rotation is based on the center of the rectangle. The Pixel Cursor (PC) is left at a2, b2 after the BOX statement is executed.

EXAMPLES:

BOX 1, 10, 10, 60, 60 ◄— Draws the outline of a rectangle

BOX , 10, 10, 60, 60, 45, 1 ◄— Draws a filled, rotated box (a diamond)

BOX , 30, 90, , 45, 1 ◄— Draws a filled, rotated polygon

CHAR

CHAR [color source #] ,x,y [,text string] [,reverse flag]

Alternate:

CHAR [color source #] ,x,y [,"text string"] [,reverse flag]

|	
color source	Color source (0 - 3)
x	Character column (0 - 39)
y	Character row (0 - 24)
string	String to print
reverse	Reverse field flag (0 = off, 1 = on)

Text (alphanumeric strings) can be displayed on any screen at a given location by the CHAR command. Character data is read from Plus/4 character ROM area. You supply the x and y coordinates of the starting position and the text string you want to display. Color source and reverse imaging are optional.

The string is continued on the next line if it attempts to print past the right edge of the screen. When used in TEXT mode, the string printed by the CHAR command works just like a PRINT string, including reverse field, cursors, flash on/off, etc. These control functions inside the string do not work when the CHAR command is used to display text in GRAPHIC mode.

> NOTE: when in multicolor mode, to display a character in multicolor 2, set the color source to 0 and reverse flag to 1. To display a character in multicolor 1, set the color source to 0 and the reverse flag to 0.

CIRCLE   CIRCLE [cs] ,[a,b] , xr [, [yr] [, [sa] [, [ea] [, [angle] [,inc]]]]]

cs . . . . Color source (0 - 3)
a,b . . . . Center coordinate (scaled)
                                        (defaults to the Pixel Cursor [PC])
xr . . . . X radius (scaled)
yr . . . . Y radius (default is xr)
sa . . . . Starting arc angle (default 0)
ea . . . . Ending arc angle (default 360)
angle . . . . Rotation in clockwise degrees
                                        (default is 0 degrees)
inc . . . . Degrees between segments
                                        (default is 2 degrees)
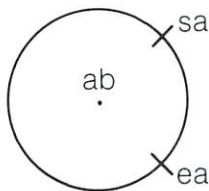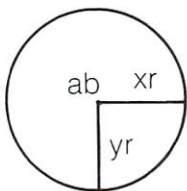
With the CIRCLE command you can draw a circle, ellipse, arc, triangle or an octagon. The final coordinate is on the circumference of the circle at the ending arc angle. Any rotation is about the center. Setting the Y radius equal to the X radius does not draw a circle, since the X and Y coordinates are scaled differently. Arcs are drawn from the starting angle clockwise to the ending angle. The segment increment controls the coarseness of the shape, with lower values for inc creating rounder shapes.

EXAMPLES:

CIRCLE , 160,100,65,10	Draws an ellipse.
CIRCLE , 160,100,65,50	Draws a circle.
CIRCLE , 60,40,20,18,,,,45	Draws an octagon.
CIRCLE , 260,40,20,,,,,90	Draws a diamond.
CIRCLE , 60,140,20,18,,,,120	Draws a triangle.

CLOSE

CLOSE file #

This command completes and closes any files used by OPEN statements. The number following the word CLOSE is the file number to be closed.

EXAMPLE:

CLOSE 2 ◄——————— Logical file 2 is closed.

CLR

CLR

This command erases any variables in memory, but leaves the program itself intact. This command is automatically executed when a RUN or NEW command is given, or when any editing is performed.

CMD

CMD file # [,write list]

CMD sends the output which normally would go to the screen (i.e. PRINT statement, LISTS, but not POKEs into the screen) to another device instead. This could be a printer, or a data file on tape or disk. This device or file must be OPENed first. The CMD command must be followed by a number or numeric variable referring to the file.

EXAMPLES:

OPEN 1,4 — OPENS device #4, which is the printer.

CMD 1 — All normal output now goes to the printer.

LIST — The LISTing goes to the printer, not the screen—even the word READY.

PRINT#1 — Set output back to the screen.

CLOSE 1 — Close the file.

COLOR

COLOR *source #, color # [,luminance #]*

Assigns a color to one of the 5 color sources:

Number	Source
0	background
1	foreground
2	multicolor 1
3	multicolor 2
4	border

Colors you can use are in the range 1-16 (BLACK, WHITE . . .). As an option, you can include the luminance level 0-7, with 0 being lowest and 7 being highest. Luminance defaults to 7. Luminance lets you select from eight levels of brightness for any color except black.

DATA

DATA *list of constants separated by commas*

This statement is followed by a list of items to be used by READ statements. The items may be numbers or words, and are separated by commas. Words need not be inside of quote marks, unless they con-

tain any of the following characters: SPACE, colon, or comma. If two commas have nothing between them, the value will be READ as a zero for a number, or an empty string. Also see the RESTORE statement, which allows the Plus/4 to re-read data.

EXAMPLE:

DATA 100, 200, FRED, "HELLO,MOM", , 3, 14, ABC123

DEF FN   DEF FN *name (variable)* = *expression*

(DEFine FunctioN)

This command allows you to define a complex calculation as a function. In the case of a long formula that is used several times within a program, this can save a lot of space. The name you give the function begins with the letters FN, followed by any legal numeric variable name. First you must define the function by using the statement DEF followed by the name you have given the function. Following the name is a set of parentheses () with a numeric variable (in this case, X) enclosed. Then you have an equal sign, followed by the formula you want to define. You can "call" the formula, substituting any number for X, using the format shown in line 20 of the example below:

EXAMPLE:

10 DEF FNA(X) = 12*(34.75 − X/.3) + X ◄──── The number 7 is inserted each place X is located in the formula given in the DEF statement.

20 PRINT FNA(7)

DIM   DIM *variable (subscripts)* [*,variable(subscripts)*]...

Before you can use an array of variables, the program must first execute a DIM statement to establish the DIMensions of that array (unless there are 11 or fewer elements in the array). The statement DIM is followed by the name of the array, which may be any legal variable name. Then, enclosed in parentheses, you put the number (or numeric variable) of elements in each dimension. An array with more than one dimension is called a matrix. You may use any number of dimensions, but keep in mind that the whole list of variables you are creating takes up space in memory, and it is easy to run out of memory if you get car-
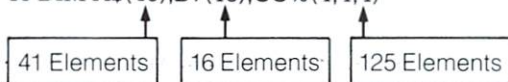
ried away. To figure the number of variables created with each DIM, multiply the total number of elements in each dimension of the array. (Each array starts with element 0.)

> NOTE: Integer (single-digit) arrays take up 2/5ths of the space of floating point arrays.

EXAMPLE:
10 DIM A$(40),B7(15),CC%(4,4,4)

| 41 Elements | 16 Elements | 125 Elements |

You can dimension more than one array in a DIM statement by separating the arrays by commas. If the program executes a DIM statement for any array more than once, you'll get a re'DIMed array error message. It is good programming practice to place DIM statements near the beginning of the program.

**DO/
LOOP/
WHILE/
UNTIL/
EXIT**

DO[UNTIL *boolean argument* WHILE *boolean argument*] *statements* [EXIT]

LOOP [UNTIL *boolean argument* WHILE *boolean argument*]

Performs the statements between the DO statement and the LOOP statement. If no UNTIL or WHILE modifies either the DO or the LOOP statement, execution of the intervening statements continues indefinitely. If an EXIT statement is encountered in the body of a DO loop, execution is transferred to the first statement following the LOOP statement. DO loops may be nested, following the rules defined for FOR-NEXT loops. If the UNTIL parameter is used, the program continues looping until the boolean argument is satisfied (becomes TRUE). The WHILE parameter is basically the opposite of the UNTIL parameter: the program continues looping as long as the boolean argument is TRUE. An example of a boolean argument is A=1, or G>=65.
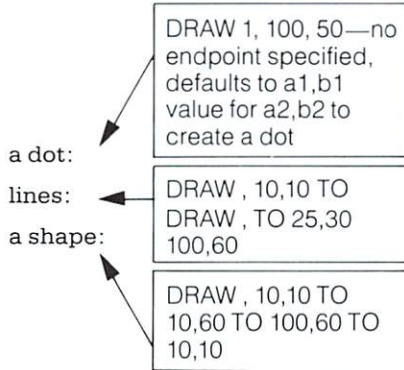
EXAMPLE:

DO UNTIL X=0 OR X=1
:
LOOP
DO WHILE A$=" ":GET
A$:LOOP

DRAW

DRAW *color source* #[, *a1, b1* [TO*a2, b2*...]]

With this command you can draw individual dots, lines, and shapes.
You supply color source (0-3), starting (a1, b1) and ending points
(a2, b2).

EXAMPLES:

a dot:

> DRAW 1, 100, 50—no
> endpoint specified,
> defaults to a1,b1
> value for a2,b2 to
> create a dot

lines:

a shape:

> DRAW , 10,10 TO
> DRAW , TO 25,30
> 100,60

> DRAW , 10,10 TO
> 10,60 TO 100,60 TO
> 10,10

END

END

When the program executes an END statement, the program stops
RUNning immediately. You may use the CONT command to re-start the
program at the statement following the END statement.

FOR...
TO...
STEP

FOR *variable = start value* TO *end value* [STEP *increment*]

This statement works with the NEXT statement to set up a section of the
program that repeats for a set number of times. You may just want the
Plus/4 to count up to a large number so the program pauses for a few

seconds, in case you need something counted, or something must be done a certain number of times (such as printing).

The loop variable is the variable that is added to or subtracted from during the FOR/NEXT loop. The start value and the end value are the beginning and ending counts for the loop variable.

The logic of the FOR statement is as follows: First, the loop variable is set to the start value. When the program reaches a line with the command NEXT, it adds the STEP increment (default = 1) to the value of the loop variable and checks to see if it is higher than the end of loop value. If it is not higher, the next line executed is the statement immediately following the FOR statement. If the loop variable is larger than the end of loop number, then the next statement executed is the one following the NEXT statement. See also the NEXT statement.

EXAMPLE:

```
10 FOR L = 1 TO 10
20 PRINT L
30 NEXT L
40 PRINT "I'M DONE! L = "L
```

This program prints the numbers from one to ten on the screen, followed by the message I'M DONE! L = 11.

The end of loop value may be followed by the word STEP and another number or variable. In this case, the value following the STEP is added each time instead of one. This allows you to count backwards, by fractions, or any way necessary.

You can set up loops inside one another. This is known as nesting loops. You must be careful to nest loops so that the last loop to start is the first one to end.

EXAMPLE OF NESTED LOOPS:

```
10 FOR L = 1 TO 100

20 FOR A = 5 TO 11 STEP 2

30 NEXT A

40 NEXT L
```

This FOR . . . NEXT loop is "nested" inside the larger one.

GET   GET *variable list*

The GET statement is a way to get data from the keyboard one character at a time. When the GET is executed, the character that was typed is received. If no character was typed, then a null (empty) character is returned, and the program continues without waiting for a key. There is no need to hit the **RETURN** key, and in fact the **RETURN** key can be received with a GET.

The word GET is followed by a variable name, usually a string variable. If a numeric were used and any key other than a number was hit, the program would stop with an error message. The GET statement may also be put into a loop, checking for an empty result, which waits for a key to be struck to continue. The GETKEY statement could also be used in this case. This command can only be executed within a program.

EXAMPLE:

10 GET A$:IF A$ <> "A" THEN 10 ◄──── This line waits for the A key to be pressed to continue.

GETKEY   GETKEY *variable list*

The GETKEY statement is very similar to the GET statement. Unlike the GET statement, GETKEY waits for the user to type a character on the keyboard. This lets it be used easily to wait for a single character to be typed.

This command can only be executed within a program.

EXAMPLE:

10 GETKEY A$ ◄──── This line waits for a key to be struck. Typing any key will continue the program.

GET#   GET# *file number,variable list*

Used with a previously OPENed device or file to input one character at a time. Otherwise, it works like the GET statement. This command can only be executed within a program.

EXAMPLE:

GET#1,A$

GOSUB GOSUB *line #*

This statement is like the GOTO statement, except that the Plus/4 re-
members where it came from. When a line with a RETURN statement is
encountered, the program jumps back to the statement immediately
following the GOSUB. The target of a GOSUB statement is called a
subroutine. A subroutine is useful if there is a routine in your program
that can be used by several different portions of the program. Instead
of duplicating the section of program over and over, you can set it up
as a subroutine, and GOSUB to it from the different parts of the pro-
gram. See also the RETURN statement.

EXAMPLE:

20 GOSUB 800 ◄——— means go to the
subroutine begin-
ning at line 800
and execute it.

:

:

800 PRINT "HI THERE":RETURN

GOTO
or GO TO GOTO *line #*

After a GOTO statement is executed, the next line to be executed will
be the one with the line number following the word GOTO. When used
in direct mode, GOTO line # allows you to start execution of the pro-
gram at the given line number without clearing the variables.

EXAMPLE:

10 PRINT"COMMODORE"   The GOTO in line 20
20 GOTO 10 ◄———   makes line 10 repeat
continuously until you
press **RUN/STOP**

GRAPHIC GRAPHIC *<mode [,clear option] / CLR>*

This statement puts the Plus/4 in one of its 5 graphic modes:

| mode	description
0	normal text
1	high-resolution graphics
2	high-resolution graphics, split screen
3	multicolor graphics
4	multicolor graphics, split screen

When executed, GRAPHIC 1-4 allocates a 10K bit-mapped area, and the start of the BASIC text area is moved above the hi-res area. This area remains allocated even if the user returns to TEXT mode (GRAPHIC 0). If 1 is given in the GRAPHIC statement as the second argument, the screen is also cleared. Executing a GRAPHIC CLR command de-allocates the 10K bit-mapped area, and makes it available once again for BASIC text and variables.

IF ...
THEN
[...:ELSE]

IF *expression* THEN *then-clause* [:ELSE *else-clause*]

IF...THEN lets the computer analyze a BASIC expression preceded by IF and take one of two possible courses of action. If the expression is true, the statement following THEN is executed. This expression may be any BASIC statement. If the expression is false, the program goes directly to the next line, unless an ELSE clause is present. The expression being evaluated may be a variable or formula, in which case it is considered true if nonzero, and false if zero. In most cases, there is an expression involving relational operators ($=$, $<$, $>$, $<=$, $>=$, $<>$, AND, OR, NOT).

The ELSE clause, if present, must be in the same line as the IF-THEN part. When an ELSE clause is present, it is executed when the THEN clause isn't executed. In other words, the ELSE clause executes when the IF expression is FALSE.
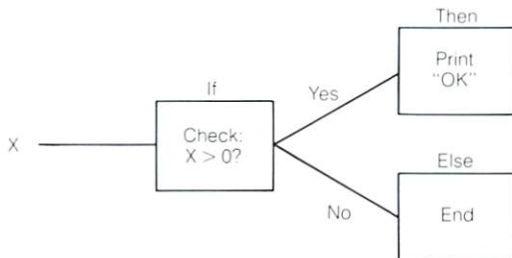
EXAMPLE:

50 IF X>0 THEN PRINT "OK":ELSE END

> Checks the value of X. If X is greater than 0, the THEN clause is executed, and the ELSE clause isn't. If X is less than 0, the ELSE clause is executed and the THEN clause isn't.

INPUT

INPUT [*prompt string;*] *variable list*

The INPUT statement allows the computer to ask for data from the person running the program and place it into a variable or variables. The program stops, prints a question mark (?) on the screen, and waits for the person to type the answer and hit the **RETURN** key.

The word INPUT is followed by a variable name or list of variable names separated by commas. There may be a message inside of quotes before the list of variables to be input. If this message (called a prompt) is present, there must be a semicolon (;) after the closing quote of the prompt. When more than one variable is to be INPUT, they should be separated by commas when typed in. If not, the computer asks for the remaining values by printing two question marks (??). If you press the **RETURN** key without INPUTting a value, the INPUT variable retains the value previously input for that variable. This statement can only be executed within a program.

EXAMPLE:

10 INPUT "PLEASE TYPE A NUMBER";A
20 INPUT "AND YOUR NAME";A$
30 INPUT B$
40 PRINT"BET YOU DIDN'T KNOW WHAT I WANTED!"

INPUT#

INPUT# *file number, variable list*

This works like INPUT, but takes the data from a previously OPENed file or device. No prompt string is allowed. This command can only be used in program mode.

EXAMPLE:
INPUT#2, A$, C, D$

LET   [LET] *variable expression*

The word LET is hardly ever used in programs, since it is not neces-
sary, but the statement itself is the heart of all BASIC programs. When-
ever a variable is defined or given a value, LET is always implied. The
variable name which is to get the result of a calculation is on the left
side of the equal sign, and the number or formula is on the right side.

EXAMPLE:

10 LET A = 5

20 B = 6

30 C = A * B + 3

40 D$ = "HELLO"

LOCATE   LOCATE *x-coordinate, y-coordinate*

The LOCATE command lets you put the pixel cursor (PC) anywhere
on the screen. The PC is the current location of the starting point of the
next drawing. Unlike the regular cursor, you can't see the PC, but you
can move it with the LOCATE command. For example:

LOCATE 160, 100

positions the PC in the center of the high resolution screen. You won't
see anything until you actually draw something. You can find out
where the PC is at any time by using the RDOT(0) function to get the
X-coordinate and RDOT(1) to get the Y-coordinate. The color source
of the dot at the PC can be found by printing RDOT(2). (In all drawing
commands where a color option is available, you may select a value
from 0 to 3, corresponding to the background, foreground, multicolor
1, or multicolor 2 as the color source.)

MONITOR   MONITOR

This command takes you out of BASIC into the built-in machine lan-
guage monitor program. The monitor is used to develop, debug, and
execute machine language programs more easily than from BASIC.
See the section on monitor commands for more information. (When
in the monitor, typing an X and hitting **RETURN** gets you back
to BASIC.)

NEXT

NEXT [*variable, . . . , variable*]

The NEXT statement is used with the FOR statement. When the computer encounters a NEXT statement, it goes back to the corresponding FOR statement and checks the loop variable. (See FOR statement for more detail.) If the loop is finished, execution proceeds with the statement after the NEXT statement. The word NEXT may be followed by a variable name, a list of variable names separated by commas, or no variable names. If there are no names listed, the last loop started is the one being completed. If the variables are given, they are completed in order from left to right.

EXAMPLE:

```
10 FOR L = 1 TO 10:NEXT
20 FOR L = 1 TO 10:NEXT L
30 FOR L = 1 TO 10:FOR M = 1 TO 10: NEXT M, L
```

ON

ON *expression* <GOTO/GOSUB> *line #1* [, *line #2,. . .*]

This command can make the GOTO and GOSUB statements into special versions of the IF statement. The word ON is followed by a formula, then either GOTO or GOSUB, and a list of line numbers separated by commas. If the result of the calculation of the formula (expression) is 1, the first line in the list is executed. If the result is 2, the second line number is executed, and so on. If the result is 0, or larger than the number of line numbers in the list, the next line executed is the statement following the ON statements. If the number is negative, an ILLEGAL QUANTITY ERROR results.

EXAMPLE:

10 INPUT X:IF X<0 THEN 10

20 ON X GOTO 50, 30, 30, 70

25 PRINT "FELL THROUGH":GOTO 10

30 PRINT "TOO HIGH":GOTO 10

50 PRINT "TOO LOW":GOTO 10

70 END

When X = 1, ON sends control to the first line number in the list .

When X = 2, ON sends control to the second line (30), etc.

OPEN OPEN file #, device # [,secondary address] [,"filename, type, mode"]

The OPEN statement allows the Plus/4 to access devices such as the cassette recorder and disk for data, a printer, or even the screen of the Plus/4. The word OPEN is followed by a logical file number, which is the number to which all other BASIC statements will refer. This number is from 1 to 255. There is always a second number after the first called the device number. Device number 0 is the Plus/4 keyboard, 3 is the Plus/4 screen, 1 is the cassette recorder, 4 is the printer, 8 is usually the disk. A zero (0) may be included in front of the device number digit (e.g., 08 for 8, which are interchangeable for the Plus/4). It is often a good idea to use the same file number as the device number because it makes it easy to remember which is which. Following the second number may be a third number called the secondary address. In the case of the cassette, this can be 0 for read, 1 for write, and 2 for write with end-of-tape marker at the end. In the case of the disk, the number refers to the channel number. In the printer, the secondary addresses are used to set the mode of the printer. See the Plus/4 Programmer's Reference Manual or the manual for each specific device for more information for more on secondary addresses. There may also be a string following the third number, which could be a command to the disk drive or the name of the file on tape or disk. The type and mode refer to disk files only. (File types are prg,seq, rel, and usr; modes are read and write.)

EXAMPLES:

10 OPEN 3,3	OPENs the SCREEN as a device.
10 OPEN 1,0	OPENs the keyboard as a device.
20 OPEN 1,1,0,"DOT"	OPENs the cassette for reading, file to be searched for is named DOT.
OPEN 4,4	OPENs a channel to use the printer.
OPEN 15,8,15	OPENs the command channel on the disk.

```
5 OPEN 8,8,12,"TESTFILE,SEQ,WRITE"
```
creates a sequential disk file for writing.

See also: CLOSE, CMD, GET#, INPUT#, and PRINT# statements, system variable ST, DS, and DS$.

PAINT

PAINT [color source] [,[a,b] [,mode]]

Color source . . . .(0-3); (default is 1, foreground color)
a,b . . . . .starting coordinate, scaled (default is at the PC)
mode . . . . .0 = paint an area defined by the color
source selected
1 = paint an area defined by any non-background source

The PAINT command lets you fill an area with color. It fills in the area around the specified point until a boundary of the same color (or any non-background color, depending on which mode you have chosen) is encountered.
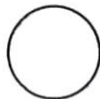
The final position of the PC will be at the starting point (a,b).

NOTE: If the starting point is already the color of color source you name (or any non-background when mode 1 is used), there is no change.

EXAMPLE:

```
10 CIRCLE , 160,100,65,50
```
draws outline of circle



```
20 PAINT , 160,100
```
fills in the circle with color

POKE

POKE *address, value*

The POKE command allows you to change any value in the Plus/4 RAM memory, and lets you modify many of the Plus/4 Input/Output registers. POKE is always followed by two numbers, (or equations). The first number is a location inside the Plus/4 memory. This could have any value from 0 to 65535. The second number is a value from 0 to 255, which is placed in the location, replacing any value that was there previously.

EXAMPLE:
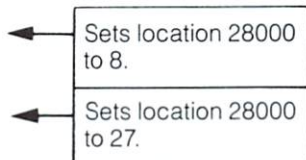
10 POKE 28000,8	◄──— Sets location 28000 to 8.
20 POKE 28*1000,27	◄──— Sets location 28000 to 27.

Note: PEEK is listed under FUNCTIONS.

PRINT

PRINT *printlist*

The PRINT statement is the major output statement in BASIC. While the PRINT statement is the first BASIC statement most people learn to use, there are many subtleties to be mastered here as well. The word PRINT can be followed by any of the following:

Characters inside of quotes	("text lines")
Variable names	(A, B, A$, X$)
Functions	(SIN(23), ABS(33))
Punctuation marks	(; ,)

The characters inside of quotes are often called literals because they are printed exactly as they appear. Variable names have the value they contain (either a number or a string) printed. Functions also have their number values printed. Punctuation marks are used to help format the data neatly on the screen. The comma divides the screen into 4 columns for data, while the semicolon doesn't add any spaces. Either mark can be used as the last symbol in the statement. This results in the next PRINT statement acting as if it is continuing the last PRINT statement.

EXAMPLE:

	RESULT	
10 PRINT "HELLO"	HELLO	
20 A$="THERE":PRINT "HELLO,"A$	HELLO,THERE	
30 A=4:B=2:PRINT A+B	6	
50 J=41:PRINT J;:PRINT J−1	41 40	
60 C=A+B:D=A−B:PRINT A;B;C,D	4 2 6	2

See also: POS(), SPC(), and TAB() FUNCTIONS.

PRINT

PRINT# *file #, print list*

There are a few differences between this statement and the PRINT.
First of all, the word PRINT# is followed by a number, which refers to
the device or data file previously OPENed. The number is followed by a
comma, and a list of things to be PRINTed. The comma and semicolon
act in the same manner for spacing as they do in the PRINT statement.
Some devices may not work with TAB and SPC.

EXAMPLE:

100 PRINT#1,"HELLO THERE!",A$,B$,

PRINT USING

PRINT [*#filenumber*] USING *format list; print list;*

These statements let you define the format of string and numeric items
you want to print to the screen, printer, or another device. Put the for-
mat you want in quotes. This is the format list. Then add a semicolon
and a list of what you want printed in the format for the print list. The list
can be variables or the actual values you want printed. For example:

 5 X=32: Y=100.23: A$="CAT"
10 PRINT USING "$##.##";13.25,X,Y
20 PRINT USING "###>#";"CBM",A$

When you RUN this, line 10 prints out:

$13.25 $32.00 $***** ◄───── prints ***** instead
of Y value because
Y has 5 digits, which
does not conform to
format list (as ex-
plained below)

Line 20 prints this:

CBM   CAT

> leaves three spaces
> before printing
> "CBM" as defined in
> format list

CHARACTER	NUMERIC	STRING
Pound Sign (#)	X	X
Plus (+)	X	
Minus (−)	X	
Decimal Point (.)	X	
Comma (,)	X	
Dollar Sign ($)	X	
Four Carets (↑↑↑↑)	X	
Equal Sign (=)		X
Greater Than Sign (>)		X

The pound sign (#) reserves room for a single character in the output field. If the data item contains more characters than you have # in your format field, the following occurs:

• For a numeric item, the entire field is filled with asterisks (*). No numbers are printed.

For example:

10 PRINT USING "####",X

For these values for X, this format displays:

A = 12.34	12
A = 567.89	568
A = 123456	****

For a STRING item, the string data is truncated at the bounds of the field. Only as many characters are printed as there are pound signs (#) in the format item. Truncation occurs on the right.

The plus (+) and minus (−) signs can be used in either the first or last position of a format field but not both. The plus sign is printed if the number is positive. The minus sign is printed if the number is negative.

If you use a minus sign and the number is positive, a blank is printed in the character position indicated by the minus sign.

If you don't use either a plus or minus sign in your format field for a numeric data item, a minus sign is printed before the first digit or dollar symbol if the number is negative and no sign is printed if the number is positive. This means that you can print one character more if the number is positive. If there are too many digits to fit into the field specified by the # and + / − signs, then an overflow occurs and the field is filled with asterisks (*).

A decimal point (.) symbol designates the position of the decimal point in the number. You can only have one decimal point in any format field. If you don't specify a decimal point in your format field, the value is rounded to the nearest integer and printed without any decimal places.

When you specify a decimal point, the number of digits preceding the decimal point (including the minus sign, if the value is negative) must not exceed the number of # before the decimal point. If there are too many digits an overflow occurs and the field is filled with asterisks (*).

A comma (,) lets you place commas in numeric fields. The position of the comma in the format list indicates where the comma appears in a printed number. Only commas within a number are printed. Unused commas to the left of the first digit appear as the filler character. At least one # must precede the first comma in a field.

If you specify commas in a field and the number is negative, then a minus sign is printed as the first character even if the character position is specified as a comma.

EXAMPLES:

FIELD	EXPRESSION	RESULT	COMMENT
##.#+	−.01	0.01−	Leading zero added.
##.#−	1	1.0	Trailing zero added.
####	−100.5	−101	Rounded to no decimal places.
####	−1000	****	Overflow because four digits and minus sign cannot fit in field.
###.	10	10.	Decimal point added.
#$##	1	$1	Leading $ sign.

A dollar sign ($) symbol shows that a dollar sign will be printed in the number. If you want the dollar sign to float (always be placed before

the number), you must specify at least one # before the dollar sign. If you specify a dollar sign without a leading #, the dollar sign is printed in the position shown in the format field. If you specify commas and/or a plus or minus sign in a format field with a dollar sign, your program prints a comma or sign before the dollar sign.

The four up arrows or carets (↑↑↑↑) symbol is used to specify that the number is to be printed in E + format. You must use # in addition to the ↑↑↑↑ to specify the field width. The ↑↑↑↑ can appear either before or after the # in the format field.

You must specify four carets (↑↑↑↑) when you want to print a number in E − format (scientific notation). If you specify more than one but fewer than four carets, you get a syntax error. If you specify more than four carets only the first four are used. The fifth caret is interpreted as a no text symbol.

An equal sign (=) is used to center a string in the field. You specify the field width by the number of characters (# and =) in the format field. If the string contains fewer characters than the field width, the string is centered in the field. If the string contains more characters than can be fit into the field, the right-most characters are truncated and the string fills the entire field. A greater than sign (>) is used to right justify a string in a field. You specify the field width by the number of characters (# and =) in the format field. If the string contains fewer characters than the field width, the string is right justified in the field. If the string contains more characters than can be fit into the field, the right-most characters are truncated and the string fills the entire field.

PUDEF

PUDEF *"1 through 4 characters"*

PUDEF lets you redefine up to 4 symbols in the PRINT USING statement. You can change blanks, commas, decimals points, and dollar signs into some other character by placing the new character in the correct position in the PUDEF control string.

Position 1 is the filler character. The default is a blank. Place a new character here when you want another character to appear in place of blanks.

Position 2 is the comma character. Default is a comma.

Position 3 is the decimal point.

Position 4 is the dollar sign.

EXAMPLES:

10 PUDEF "*" ← PRINTs * in the place of blanks.

20 PUDEF " @" ← PRINTs @ in place of commas.

30 PUDEF " .," ← PRINTs decimal points in place of commas, and commas in place of decimal points.

40 PUDEF " ,.£" ← PRINTs English pound sign in place of $, decimal points in place of commas, and commas in place of decimal points. Other signs are the default values.

READ

READ *variable list*

This statement is used to get information from DATA statements into variables, where the data can be used. The READ statement variable list may contain both strings and numbers. Care must be taken to avoid reading strings where the READ statement expects a number, which produces an ERROR message.

EXAMPLE:

READ A$, G$, 45

REM

REM *message*

The REMark is just a note to whoever is reading a LIST of the program. It may explain a section of the program, give information about the author, etc. REM statements in no way affect the operation of the program, except to add to its length (and therefore slow it down). The word REM may be followed by any text, although use of graphic characters gives strange results.

EXAMPLE:

10 NEXT X: REM THIS LINE IS UNNECESSARY

RESTORE   RESTORE [line #]

When executed in a program, the pointer to the item in a DATA statement which is to be read next is reset to the first item in the list. This gives you the ability to re-READ the information. If a [line #] follows the RESTORE statement, the pointer is set to that line. Otherwise the pointer is reset to the first DATA statement in the program.

EXAMPLE:

RESTORE 200

RESUME   RESUME [line # /NEXT]

Used to return to execution after TRAPping an error. With no arguments, RESUME attempts to re-execute the line in which the error occurred. RESUME NEXT resumes execution at the next statement following the statement containing the error; RESUME line # will GOTO the specific line and begin execution there.

RETURN   RETURN

This statement is always used with the GOSUB statement. When the program encounters a RETURN statement, it goes to the statement immediately following the last GOSUB command executed. If no GOSUB was previously issued, then a RETURN WITHOUT GOSUB ERROR message is delivered, and program execution is stopped.

SCALE   SCALE <1/0>

The scaling of the bit maps in multicolor and high resolution modes can be changed with the SCALE command. Entering:

SCALE 1

turns scaling on. Coordinates may then be scaled from 0 to 1023 in both X and Y rather than the normal scale values, which are:

multicolor mode .... $X = 0$ to 159	$Y = 0$ to 199
high resolution mode .... 0 to 319	0 to 199

Scaling can be turned off by entering 'SCALE 0'.

SCNCLR

SCNCLR

Clears the current screen, whether graphics, text, or both
(split screen).

SOUND

SOUND *voice #, frequency control, duration*

This statement produces a SOUND using one of three voices with a
frequency control in the range 0 − 1023 for a duration of 0 − 65535
60ths of a second.

V	Voice
1	Voice 1 (tone)
2	Voice 2 (tone)
3	Voice 2 (white noise)

If a SOUND for voice N is requested, and the previous SOUND for the
same N is still playing, BASIC waits for the previous SOUND to com-
plete. SOUND with a duration of 0 is a special case. It causes BASIC
to turn off the current SOUND for that voice immediately, regardless
of the time remaining on the previous SOUND. See the MUSIC NOTE
TABLE (SECTION 11) for the frequency control values that correspond
to real notes.

EXAMPLE:

SOUND 2, 800, 360 ◄———
> Plays a note using
> voice 2 with fre-
> quency set at 800
> for one minute

SSHAPE/GSHAPE

SSHAPE and GSHAPE are used to save and restore rectangular areas
of multicolor or high resolution screens using BASIC string variables.
The command to save an area is:

SSHAPE *string variable, a1,b1 [,a2,b2]*

string variable .... String name to save data in

a1,b1 .... Corner coordinate (scaled)
a2,b2 .... Corner coordinate opposite (a1,b1)
                                                   (default is the PC)

Because BASIC limits string lengths to 255 characters, the size of the area you may save is limited. The string size required can be calculated using one of the following (unscaled) formulas:

$L(mcm) = INT ((ABS(a1-a2) + 1) / 4 + .99) * (ABS(b1 - b2) + 1) + 4$
$L(h\text{-}r) = INT ((ABS(a1-a2) + 1) / 8 + .99) * (ABS(b1 - b2) + 1) + 4$
(mcm) refers to multi-color mode; (h-r) is high resolution.

The shape is saved row by row. The last four bytes of the string contain the column and row lengths less one (i.e.: ABS (a1−a2)) in low/high byte format (if scaled divide the lengths by 3.2 (X) and 5.12 (Y)).

The command to display a saved shape on any area of the screen:

GSHAPE *string* [, [*a,b*] [,*mode*]]

string .... Contains shape to be drawn
a,b .... Top left coordinate telling where
                                                   to draw the shape (scaled —
                                                   the default is the PC)
mode .... Replacement mode:
                                                   0: place shape as is (default)
                                                   1: place field inverted shape
                                                   2: OR shape with area
                                                   3: AND shape with area
                                                   4: XOR shape with area

EXAMPLES:

SSHAPE "SHIP",0,0   Saves shape on screen area from the upper left corner to the cursor under the name "SHIP"

GSHAPE "SHIP",,,1   Displays inverted "SHIP" shape with the top left corner positioned where the cursor is located

STOP   STOP

This statement halts the program. A message, BREAK IN LINE #,
where the # is the line number containing the STOP. The program
can be re-started at the statement following STOP if you use the
CONT command. The STOP statement is usually used while de-
bugging a program.

SYS   SYS *address*

The word SYS is followed by a decimal number or numeric variable
in the range 0 to 65535. The program begins executing the machine
language program starting at that memory location. This is similar
to the USR function, but does not pass a parameter. See the Plus/4
Programmer's Reference Guide for information about machine
language programs.

TRAP   TRAP [*line #*]

When turned on, TRAP intercepts all error conditions (including the
STOP KEY) except "UNDEF'D STATEMENT ERROR". In the event of
any execution error, the error flag is set, and execution is transferred
to the line number named in the TRAP statement. The line number in
which the error occurred can be found by using the system variable
EL. The specific error condition is contained in system variable ER.
The string function ERR$(ER) gives the error message correspond-
ing to any error condition ER.

> NOTE: An error in a TRAP routine cannot be trapped.
> The RESUME statement can be used to resume execution.
> TRAP with no *line #* argument turns off error TRAPping.

TRON   TRON

TRON is used in program debugging. This statement begins trace
mode. When you are in trace mode, as each statement executes, the
line number of that statement is printed.

TROFF TROFF

This statement turns trace mode off.

VOL VOL *volume level*

Sets the current VOLume level for SOUND commands. VOLume may be set from 0 to 8, where 8 is maximum volume, and 0 is off. VOL affects both voices.

WAIT WAIT *address, value 1* [, *value 2*]

The WAIT statement is used to halt the program until the contents of a location in memory changes in a specific way. The address must be in the range from 0 to 65535. Value 1 and value 2 must be in the range from 0 to 255.

The content of the memory location is first exclusive-ORed with value 2 (if present), and then logically ANDed with value 1. If the result is zero, the program checks the memory location again. When the result is not zero, the program continues with the next statement.

There are a few concepts that apply to all of the bit map graphics statements. First is the concept of the Pixel Cursor (PC). The PC is similar to the cursor in text mode; it is the position where the next dot is to be drawn. Unlike the text cursor, the PC is invisible. All drawing commands use the PC. In addition, the locate command allows you to reposition the PC without drawing anything.

Wherever you would use X,Y coordinates in a drawing command, you can use RELATIVE coordinates instead. Relative coordinates are based on the current value of the PC. To use relative coordinates, just place a + or − in front of your coordinates. A plus sign before the X value moves the PC to the right. A minus sign before the X value moves the PC to the left. Similarly, a minus sign before the Y coordinate moves the PC up, while a plus sign moves the PC down. For example:

LOCATE +100,−25 ◄──── moves the PC right 100 pixels and up 25.

DRAW1,+10,+10TO100,100 ◄──── draws a line 10 pixels right and 10 pixels below the current value of the PC to the absolute point 100,100.

You can also specify a distance and an angle relative to the current PC by separating the two parameters by a semicolon.

For example:

LOCATE 50;45 ◄──── moves the PC from its current location by a distance of 50 dots at an angle of 45 degrees.

FUNCTIONS

Numeric Functions

Numeric functions are classified as such because they return numbers. The functions they perform range from calculating mathematical functions to specifying a screen location. Numeric functions follow the form:

FUNCTION (argument)
where the argument can be a numerical value, variable, or string.

ABS(X) (absolute value)
The absolute value function returns the positive value of the argument X.

ASC(X$)
This function returns the ASCII code (number) of the first character of X$.

ATN(X) (arctangent)
Returns the angle whose tangent is X, measured in radians.

COS(X) (cosine)
Returns the value of the cosine of X, where X is an angle measured in radians.

DEC (hexadecimal-string)
Returns decimal value of hexadecimal-string (0<hexadecimal–string<FFFF)

EXAMPLE:

N=DEC("F4")

EXP(X)

Returns the value of the mathematical constant e (2.71828183) raised to the power of X.

FNxx(x)
Returns the value of the user-defined function xx created in a DEF FNxx statement.

INSTR (string 1, string 2 [starting-position])
Returns position of string 2 in string 1 at or after the [starting-position]. The starting-position defaults to the beginning of string 2. If no match is found, a value of 0 is returned.

EXAMPLE:

PRINT INSTR("THE CAT IN THE HAT", "CAT")

the result is 5, because CAT starts at the fifth character in string 1

INT(X) (integer)

Returns the integer portion of X, with all decimal places to the right of the decimal point removed. The result is always less-than or equal to X. Thus, any negative numbers with decimal places become the integer less-than their current value (e.g. INT(-4.5)=-5).

If the INT function is to be used for rounding up or down, the form is INT(X $+/-$ .5).

EXAMPLE:

X=INT(X*100 + .5)/100

Rounds to the next highest penny.

JOY (n)

When n =1 .... Position of joystick #1
n = 2 .... Position of joystick #2

Any value of 128 or more means the fire button is also depressed. The direction is indicated as follows:

```
                            UP
fire = 128 +                 1
                8                     2
LEFT   7                     0              3   RIGHT
                6                     4
                             5
                           DOWN
```

EXAMPLE:

JOY(2)=135

joystick #2 fires to the left

LOG(X) (logarithm)

This returns the natural log of X. The natural log is log to the base e (see EXP(X)). To convert to log base 10, divide by LOG(10).

PEEK(X)
This function gives the contents of memory location X, where X is located in the range of 0 to 65535, returning a result from 0 to 255. This is often used in conjunction with the POKE statement.

RCLR(N)
Returns current color assigned to source N ($0 < N < 4$)
($0=$ background, $1=$ foreground, $2=$ multicolor 1, $3=$ multicolor 2, $4=$ border).

RDOT(N)
Returns information about the current position of the pixel cursor (PC) at XPOS/YPOS.

N — 0 for XPOS
    1 for YPOS
    2 color source

RGR(X)
Returns current graphic mode (X is a dummy argument).

RLUM(N)
Returns current luminance level assigned to source N

RND(X) (random number)
This function returns a random number between 0 and 1. This is useful in games, to simulate dice rolls and other elements of chance, and is also used in some statistical applications. The first random number should be generated by the formula RND($-$TI), to start things off differently every time. After this, the number in X should be a 1, or any positive number. (X represents the seed, or what the RaNDom number is based on.) If X is zero, RND is re-seeded from the hardware clock every time RND is used. A negative value for X seeds the random number generator using X and gives a random number sequence. The use of the same negative number for X as a seed results in the same sequence of random numbers. A positive value gives random numbers based on the previous seed.

To simulate the rolling of a die, use the formula INT(RND(1)*6+1). First the random number from 0-1 is multiplied by 6, which expands the range to 0-6 (actually, greater than zero and less then six). Then 1 is added, making the range 1 to under 7. The INT function chops off all the decimal places, leaving the result as a digit from 1 to 6.

To simulate 2 dice, add two of the numbers obtained by the above formula together.

EXAMPLE:

`100 X=INT(RND(1)*6)+INT(RND(1)*6)+2`	Simulates 2 dice.
`100 X=INT(RND(1)*1000)+1`	Number from 1-1000.
`100 X=INT(RND(1)*150)+100`	Number from 100-249.

SGN(X) (sign)
This function returns the sign, as in positive, negative, or zero, of X. The result is + 1 if positive, 0 if zero, and − 1 if negative.

SIN(X) (sine)
This is the trigonometric sine function. The result is the sine of X, where X is an angle in radians.

SQR(X) (square root)
This function returns the square root of X, where X is a positive number or 0. If X is negative, an ILLEGAL QUANTITY ERROR results.

TAN(X) (tangent)
This gives the tangent of X, where X is an angle in radians.

USR(X)
When this function is used, the program jumps to a machine language program whose starting point is contained in memory locations 1281 and 1282. The parameter X is passed to the machine language program in the floating point accumulator. Another number is passed back to the BASIC program through the calling variable. In other words, this allows you to exchange a variable between machine code and BASIC. See the Plus/4 PROGRAMMER'S REFERENCE GUIDE for more details on this, and on machine language programming.

VAL(X$)
This function converts the string X$ into a number, and is essentially the inverse operation from STR$. The string is examined from the left-most character to the right, for as many characters as are in recognizable number format. If the Plus/4 finds illegal characters, only the portion of the string up to that point is converted.

EXAMPLE:

`10 X=VAL("123.456")` ← X=123.456

`10 X=VAL("3E03")` ← X=3000

`10 X=VAL("12A13B")` ← X=12

10 X=VAL("RIUO17*")	X=0
10 X=VAL("−1.23.23.23")	X=−1.23

String Functions

String functions differ from numeric functions in that they return characters, graphics or numbers from a string (defined by quotation marks) instead of a number.

CHR$(X)
This function returns a string character whose ASCII code is X.

ERR$(N)
Returns string describing error condition N (see TRAP)

HEX$(N)
Returns a 4 character string containing the hexadecimal representation of value N (0 < N < 65535)

LEFT$(X$,X)
This returns a string containing the leftmost X characters of X$.

LEN(X$)
Returns the number of characters (including spaces and other symbols) in the string X$.

MID$(X$,S,X)
This returns a string containing X characters, starting from the Sth character in X$. MID$ can also be used on the left side of assignment statement as a pseudo-variable as well as a function. MID$ (string variable, starting position, length) = source string.

This function reassigns values of positions (starting position) through (starting position + length) of source string to the characters of string variable in corresponding locations. Length defaults to the length of string variables, and an error results if (starting position + length) is greater than the length of the source string.

EXAMPLE:

```
10 A$="THE DOG IN THE HAT":
20 PRINT A$
30 MID$(A$,5,3)="CAT"
40 PRINT A$
```

RIGHT$(X$,X)
This returns the right-most X characters in X$.

STR$(X)
This returns a string which is identical to the PRINTed version of X$.

EXAMPLE:

A$=STR$(X)

Other Functions

FRE(X)
This function returns the number of unused bytes available in memory. X is a dummy argument.

POS(X)
This function returns the number of the column (0-79) where the next PRINT statement begins on the screen. X is a dummy argument.

SPC(X)
This is used in the PRINT statement to skip over X spaces. X can have a value from 0-255.

TAB(X)
This is used in the PRINT statement. The next item to be printed is in column number X. X can have a value from 0 to 255.

π (PI)
The π symbol, when used in an equation, has the value of 3.14159265.

VARIABLES & OPERATORS

The Plus/4 uses three types of variables in BASIC. These are: normal numeric, integer numeric, and string (alphanumeric) variables.

Variables

Normal **NUMERIC VARIABLES,** also called **floating point variables,** can have any value from $^{-38}$ to $^{+38}$, with up to nine digits of accuracy. When a number becomes larger than nine digits can show, as in N^{-10} or N^{10}, the computer displays it in scientific notation form, with the number normalized to 1 digit and eight decimal places, followed by the letter E and the power of ten by which the number is multiplied. For example, the number 12345678901 is displayed as 1.234356789E+10.

INTEGER VARIABLES can be used when the number is from +32767 to −32768, and with no fractional portion. An integer variable is a number like 5, 10, or − 100. Integers take up less space than floating point variables, particularly when used in an array.

STRING VARIABLES are those used for character data, which may contain numbers, letters, and any other character that the Plus/4 can make. An example of a string variable is "Plus/4".

Variable Names

Variable names may consist of a single letter, a letter followed by a number, or two letters. Variable names may be longer than 2 characters, but only the first two are significant.

An integer variable is specified by using the percent (%) sign after the variable name. String variables have the dollar sign ($) after their names.

EXAMPLES:

Numeric Variable Names: A, A5, BZ
Integer Variable Names: A%, A5%, BZ%
String Variable Names: A$, A5$, BZ$

ARRAYS are lists of variables with the same name, using an extra number (or numbers) to specify an element of the array. Arrays are defined using the DIM statement, and may be floating point, integer, or string variables arrays. The array variable name is followed by a set of parentheses ( ) enclosing the number of the variable in the list.

EXAMPLES:

A(7),BZ%(11),A$(87)

Arrays may have more than one dimension. A two dimensional array

may be viewed as having rows and columns, with the first number identifying the row and the second number in the parentheses identifying the column (as if specifying a certain grid on a map).

EXAMPLES:

A(7,2),BZ%(2,3,4),Z$(3,2)

Reserved Variable Names

There are seven variable names which are reserved for use by the Plus/4, and may not be used for another purpose. These are the variables DS, DS$, ER, EL, ST,TI, and TI$. You also can't use KEYWORDS such as TO and IF, or any names that contain KEYWORDS, such as SRUN, RNEW, or XLOAD as variable names.

ST is a status variable for input and output (except normal screen/keyboard operations). The value of ST depends on the results of the last input/output operation. A more detailed explanation of ST is in the Plus/4 Programmer's Reference Guide, but in general, if the value of ST is 0 the operation was sucessful.

TI and TI$ are variables that relate to the real-time clock built into the Plus/4. The system clock is updated every 1/60th of a second. It starts at 0 when the Plus/4 is turned on, and is reset only by changing the value of TI$. The variable TI gives you the current value of the clock in 1/60ths of a seconds.

TI$ is a string that reads the value of the real-time clock as a 24 hour clock. The first two characters of TI$ contain the hour, the 3rd and 4th characters are the minutes, and the 5th and 6th characters are the seconds. This variable can be set to any value (so long as all characters are numbers), and will be automatically updated as a 24 hour clock.

EXAMPLE:

TI$ = "101530" sets the clock to 10:15 and 30 seconds (AM)

The value of the clock is lost when the Plus/4 is turned off. It starts at zero when the Plus/4 is turned on, and is reset to zero when the value of the clock exceeds 235959 (23 hours, 59 minutes and 59 seconds).

The variable DS reads the disk drive command channel, and returns the current status of the drive. To get this information in words, PRINT DS$. These status variables are used after a disk operation, like a DLOAD or DSAVE, to find out why the red error light on the disk drive is blinking.

ER, EL, and ERR$ are variables used in error trapping routines. They are usually only useful within a program. ER returns the last error encountered since the program was RUN. EL is the line where the error occurred. ERR$ is a function which allows your program to print one of the BASIC error messages. PRINT ERR$(ER) prints out the proper error message.

The ARITHMETIC operators include the following signs:

+	addition
−	subtraction
*	multiplication
/	division
↑	raising to a power (exponentiation)

On a line containing more than one operator, there is a set order in which operations always occur. If several operators are used together, the computer assigns priorities as follows: First, exponentiation, then multiplication and division, and last, addition and subtraction. If two operations have the same priority, then calculations are performed in order from left to right. If you want these operations to occur in a different order, Plus/4 BASIC allows you to give a calculation a higher priority by placing parentheses around it. Operations enclosed in parentheses will be calculated before any other operation. You have to make sure that your equations have the same number of left parentheses as right parentheses, or you will get a SYNTAX ERROR message when your program is run.

There are also operators for equalities and inequalities, called RELATIONAL operators. Arithmetic operators always take priority over relational operators.

=	is equal to
<	is less than
>	is greater than
< = or = <	is less than or equal to
> = or = >	is greater than or equal to
< > or > <	is not equal to

Finally, there are three LOGICAL operators, with lower priority than both arithmetic and relational operators:

AND
OR
NOT

These are used most often to join multiple formulas in IF...THEN statements. When they are used with arithmetic operators, they are evaluated last (i.e., after + and −).

EXAMPLES:

IF A=B AND C=D THEN 100   requires both A=B & C=D to be true.
IF A=B OR C=D THEN 100   allows either A=B or C=D to be true.
A=5:B=4:PRINT A=B   displays a value of 0
A=5:B=4:PRINT A>B   displays a value of -1
PRINT 123 AND 15:PRINT 5 OR 7   displays 11 and 7

**Basic 3.5
Abbreviations**

KEYWORD	ABBREVIATION			TYPE
ABS	a	**SHIFT**	B	function—numeric
ASC	a	**SHIFT**	S	function—numeric
ATN	a	**SHIFT**	T	function—numeric
AUTO	a	**SHIFT**	U	command
BACKUP	b	**SHIFT**	A	command
BOX	b	**SHIFT**	O	statement
CHAR	ch	**SHIFT**	A	statement
CHR$	c	**SHIFT**	H	function—string
CIRCLE	c	**SHIFT**	I	statement
CLOSE	cl	**SHIFT**	O	statement
CLR	c	**SHIFT**	L	statement
CMD	c	**SHIFT**	M	statement
COLLECT	col	**SHIFT**	L	command
COLOR	co	**SHIFT**	L	statement
CONT	c	**SHIFT**	O	command
COPY	co	**SHIFT**	P	command
COS		none		function—numeric
DATA	d	**SHIFT**	A	statement
DEC		none		function—numeric
DEF FN	d	**SHIFT**	E	statement
DELETE	de	**SHIFT**	L	command
DIM	d	**SHIFT**	I	statement
DIRECTORY	di	**SHIFT**	R	command
DLOAD	d	**SHIFT**	L	command
DO		none		statement
DRAW	d	**SHIFT**	R	statement
DSAVE	d	**SHIFT**	S	command
END	e	**SHIFT**	N	statement
ERR$	e	**SHIFT**	R	function—string
EXP	e	**SHIFT**	X	function—numeric
FOR	f	**SHIFT**	O	statement
FRE	f	**SHIFT**	R	function—numeric
GET	g	**SHIFT**	E	statement
GETKEY	getk	**SHIFT**	E	statement
GET#		none		statement
GOSUB	go	**SHIFT**	S	statement
GOTO	g	**SHIFT**	O	statement
GRAPHIC	g	**SHIFT**	R	statement

KEYWORD	ABBREVIATION		TYPE
GSHAPE	g **SHIFT** S		statement
HEADER	he **SHIFT** A		command
HEX$	h **SHIFT** E		function—string
IF...GOTO	none		statement
IF...THEN...ELSE	none		statement
INPUT	none		statement
INPUT#	i **SHIFT** N		statement
INSTR	in **SHIFT** S		function—numeric
INT	none		function—numeric
JOY	j **SHIFT** O		function—numeric
KEY	k **SHIFT** E		command
LEFT$	le **SHIFT** F		function—string
LEN	none		function—numeric
LET	l **SHIFT** E		statement
LIST	l **SHIFT** I		command
LOAD	l **SHIFT** O		command
LOCATE	lo **SHIFT** C		statement
LOG	none		function—numeric
LOOP	lo **SHIFT** O		statement
MID$	m **SHIFT** I		function—string
MONITOR	m **SHIFT** O		statement
NEW	none		command
NEXT	n **SHIFT** E		statement
ON...GOSUB	on...go **SHIFT** S		statement
ON...GOTO	on...g **SHIFT** O		statement
OPEN	o **SHIFT** P		statement
PAINT	p **SHIFT** A		statement
PEEK	p **SHIFT** E		function—numeric
POKE	p **SHIFT** O		statement
POS	none		function—numeric
PRINT	?		statement
PRINT#	p **SHIFT** R		statement
PRINT USING	?us **SHIFT** I		statement
PUDEF	p **SHIFT** U		statement
RCLR	r **SHIFT** C		function—numeric
RDOT	r **SHIFT** D		function—numeric
READ	r **SHIFT** E		statement
REM	none		statement

KEYWORD	ABBREVIATION			TYPE
RENAME	re	**SHIFT**	N	command
RENUMBER	ren	**SHIFT**	U	command
RESTORE	re	**SHIFT**	S	statement
RESUME	res	**SHIFT**	U	statement
RETURN	re	**SHIFT**	T	statement
RGR	r	**SHIFT**	G	function—numeric
RIGHT$	r	**SHIFT**	I	function—string
RLUM	r	**SHIFT**	L	function—numeric
RND	r	**SHIFT**	N	function—numeric
RUN	r	**SHIFT**	U	command
SAVE	s	**SHIFT**	A	command
SCALE	sc	**SHIFT**	A	statement
SCNCLR	s	**SHIFT**	C	statement
SCRATCH	sc	**SHIFT**	R	command
SGN	s	**SHIFT**	G	function—numeric
SIN	s	**SHIFT**	I	function—numeric
SOUND	s	**SHIFT**	O	statement
SPC(	s	**SHIFT**	P	function—special
SQR	s	**SHIFT**	Q	function—numeric
SSHAPE	s	**SHIFT**	S	statement
STatus		none		reserved—numeric variable
STOP	s	**SHIFT**	T	statement
STR$	st	**SHIFT**	R	function—string
SYS	s	**SHIFT**	Y	statement
TAB(	t	**SHIFT**	A	function—special
TAN		none		function—numeric
TI		none		reserved—numeric variable
TI$		none		reserved—string variable
TRAP	t	**SHIFT**	R	statement
TROFF	tro	**SHIFT**	F	statement
TRON	tr	**SHIFT**	O	statement
UNTIL	u	**SHIFT**	N	statement
USR	u	**SHIFT**	S	function—special
VAL		none		function—numeric
VERIFY	v	**SHIFT**	E	command
VOL	v	**SHIFT**	O	statement
WAIT	w	**SHIFT**	A	statement
WHILE	w	**SHIFT**	H	statement

Converting Standard BASIC Programs To Commodore BASIC 3.5

If you have programs written in a BASIC other than Commodore BASIC, some minor adjustments may be necessary before running them on the Plus/4. Here are some hints to make the conversions easier.

String Dimensions

Conversion Programs

Delete all statements that are used to declare the length of strings. A statement such as DIM A$(I,J), which DIMensions a string array for J elements of length I, should be converted to the Commodore BASIC statement DIM A$(J).

Some BASICs use a comma or ampersand for string concatenation (linking). Each of these must be changed to a + sign, which is the Commodore BASIC 3.5 operator for string concatenation.

In Commodore BASIC, the MID$, RIGHT$, and LEFT$ functions are used to take substrings of strings. Forms such as A$(I) to access the Ith character in A$, or A$(I,J) to take a substring of A$ from position I to J, must be changed as follows:

Other BASIC	Commodore BASIC 3.5
A$(I) = X$	MID$(A$,I,J) = X$
A$(I,J) = X$	MID$(A$,I,J) = X$

Multiple Assignments

To set B and C equal to zero, some BASICs allow statements of the form: 10 LET B = C =0

Commodore BASIC would interpret the second sign as a logical operator and set B = −1 if C = 0. Instead, convert this statement to: 10 C = 0: B = 0

Multiple Statements

Some BASICs use a backslash (/) to separate multiple statements on a line. With 3.5 BASIC, use a colon (:) to separate all statements.

Mat Functions

Programs using the MAT functions available on some BASICs must be rewritten using FOR...NEXT loops to execute properly.

Reprogramming Function Keys

You can reprogram the function keys to match the function keys on the Commodore 64 and VIC-20. (This also makes program conversions from those machines easier).

To reprogram the keys, put the following line into your program:

10 FOR I=1 TO 8:KEY I, CHR$(I+132): NEXT

Now whenever you type a function key, it sends a non-printing character, from 133 to 140, like the Commodore 64 does. To check for this in a program, you can use this method;

20 GETKEY A$: IF ASC(A$)=133 THEN PRINT "FUNCTION KEY 1 HIT": GOTO 20
30 IF ASC(A$) > 133 AND ASC(A$) < 141 THEN PRINT "SOME OTHER FUNCTION KEY HIT"
40 GOTO 20

After your program is done, you have to redefine the keys again if you want them to say directory, dload, etc. You can do this by hand, in a program, or by resetting the Plus/4.

These error messages are printed by BASIC. You can also PRINT the messages through the use of the ERR$() function. The error number refers only to the number assigned to the error for use with this function.

Error Messages

ERROR # ERROR NAME

1	**TOO MANY FILES**	There is a limit of 10 files OPEN at one time.
2	**FILE OPEN**	An attempt was made to open a file using the number of an already open file.
3	**FILE NOT OPEN**	The file number specified in an I/O statement must be opened before use.
4	**FILE NOT FOUND**	Either no file with that name exists (disk) or an end-of-tape marker was read (tape).
5	**DEVICE NOT PRESENT**	The required I/O device not available.
6	**NOT INPUT FILE**	An attempt made to GET or INPUT data from a file that was specified as output only.
7	**NOT OUTPUT FILE**	An attempt made to send data to a file that was specified as input only.
8	**MISSING FILE NAME**	An OPEN, LOAD, or SAVE to the disk drive generally requires a file name.
9	**ILLEGAL DEVICE NUMBER**	An attempt made to use a device improperly (SAVE to the screen, etc.)
10	**NEXT WITHOUT FOR**	Either loops are nested incorrectly, or there is a variable name in a NEXT statement that doesn't correspond with one in a FOR.
11	**SYNTAX**	A statement is unrecognizable by BASIC. This could be

		because of missing or extra parenthesis, misspelled keyword, etc.
12	**RETURN WITHOUT GOSUB**	A RETURN statement encountered when no GOSUB statement was active.
13	**OUT OF DATA**	A READ statement encountered, without data left unREAD.
14	**ILLEGAL QUANTITY**	A number used as the argument of a function or statement is outside the allowable range.
15	**OVERFLOW**	The result of a computation is larger than the largest number allowed (1.701411833E+38)
16	**OUT OF MEMORY**	Either there is no more room for program and program variables, or there are too many DO, FOR, or GOSUB statements in effect.
17	**UNDEF'D STATEMENT**	A line number referenced doesn't exist in the program.
18	**BAD SUBSCRIPT**	The program tried to reference an element of an array out of the range specified by the DIM statement.
19	**REDIM'D ARRAY**	An array can only be DIMensioned once. If an array is referenced before that array is DIM'd, an automatic DIM (to 10) is performed.
20	**DIVISION BY ZERO**	Division by zero is not allowed.
21	**ILLEGAL DIRECT**	INPUT or GET statements are only allowed within a program.

22	**TYPE MISMATCH**	This occurs when a number is used in place of a string or vice-versa.
23	**STRING TOO LONG**	A string can contain up to 255 characters.
24	**FILE DATA**	Bad data read from a tape file.
25	**FORMULA TOO COMPLEX**	Simplify the expression (break into 2 parts or use fewer parentheses).
26	**CAN'T CONTINUE**	The CONT command does not work if the program was not RUN, there was an error, or a line has been edited.
27	**UNDEF'D FUNCTION**	A user defined function referenced that was never defined.
28	**VERIFY**	The program on tape or disk does not match the program in memory.
29	**LOAD**	There was a problem loading. Try again.
30	**BREAK**	The stop key was hit to halt program execution.
31	**CAN'T RESUME**	A RESUME statement encountered without TRAP statement in effect.
32	**LOOP NOT FOUND**	The program has encountered a DO statement and cannot find the corresponding LOOP.
33	**LOOP WITHOUT DO**	LOOP encountered without a DO statement active.
34	**DIRECT MODE ONLY**	This command is allowed only in direct mode, not from a program.
35	**NO GRAPHICS AREA**	A command (DRAW, BOX, etc.) to create graphics encountered before the

GRAPHIC command
was executed.

36   **BAD DISK**   An attempt failed to HEADER
a diskette, because the quick
header method (no ID) was
attempted on an unformatted
diskette, or the diskette is bad.

**DESCRIPTION
OF
DOS
ERROR
MESSAGES**

These error messages are returned through the DS and DS$ reserved variables.

NOTE: Error message numbers less than 20 should be ignored with the exception of 01, which gives information about the number of files scratched with the SCRATCH command.

20	**READ ERROR** (block header not found)	The disk controller is unable to locate the header of the requested data block. Caused by an illegal sector number, or the header has been destroyed.
21	**READ ERROR** (no sync character)	The disk controller is unable to detect a sync mark on the desired track. Caused by misalignment of the read/writer head, no diskette is present, or unformatted or improperly seated diskette. Can also indicate a hardware failure.
22	**READ ERROR** (data block not present)	The disk controller has been requested to read or verify a data block that was not properly written. This error message occurs in conjunction with the BLOCK commands and indicates an illegal track and/or sector request.
23	**READ ERROR** (checksum error in data block)	This error message indicates that there is an error in one or more of the data bytes. The data has been read into the DOS memory, but the checksum over the data is in error. This message may also indicate grounding problems.
24	**READ ERROR** (byte decoding error)	The data or header has been read into the DOS memory, but a hardware error has been created due to an invalid bit pattern in the data

		byte. This message may also indicate grounding problems.
25	**WRITE ERROR** (write-verify error)	This message is generated if the controller detects a mismatch between the written data and the data in the DOS memory.
26	**WRITE PROTECT ON**	This message is generated when the controller has been requested to write a data block while the write protect switch is depressed. Typically, this is caused by using a diskette with a write protect tab over the notch.
27	**READ ERROR** (checksum error in header)	The controller has detected an error in the header of the re-quested data block. The block has not been read into the DOS memory. This message may also indicate grounding problems.
28	**WRITE ERROR** (long data block)	The controller attempts to detect the sync mark of the next header after writing a data block. If the sync mark does not appear within a pre-determined time, the error message is generated. The error is caused by a bad diskette format (the data extends into the next block), or by hardware failure.
29	**DISK ID MISMATCH**	This message is generated when the controller has been requested to access a diskette which has not been initialized. The message can also occur if a diskette has a bad header.
30	**SYNTAX ERROR** (general syntax)	The DOS cannot interpret the command sent to the command channel. Typically, this is caused

by an illegal number of file names, or patterns are illegally used. For example, two file names may appear on the left side of the COPY command.

31	**SYNTAX ERROR** (invalid command)	The DOS does not recognize the command. The command must start in the first position.
32	**SYNTAX ERROR** (invalid command)	The command sent is longer than 58 characters.
33	**SYNTAX ERROR** (invalid file name)	Pattern matching is invalidly used in the OPEN or SAVE command.
34	**SYNTAX ERROR** (no file given)	The file name was left out of a command or the DOS does not recognize it as such. Typically, a colon(:) has been left out of the command.
39	**SYNTAX ERROR** (invalid command)	This error may result if the command sent to command channel (secondary address 15) is unrecognized by the DOS.
50	**RECORD NOT PRESENT**	Result of disk reading past the last record through INPUT#, or GET# commands. This message will also occur after positioning to a record beyond end of file in a relative file. If the intent is to expand the file by adding the new record (with a PRINT# command), the error message may be ignored. INPUT or GET should not be attempted after this error is detected without first repositioning.
51	**OVERFLOW IN RECORD**	PRINT# statement exceeds record boundary. Information is truncated. Since the carriage re-

turn which is sent as a record terminator is counted in the record size, this message will occur if the total characters in the record (including the final carriage return) exceeds the defined size.

| 52 | **FILE TOO LARGE** | Record position within a relative file indicates that disk overflow will result. |

| 60 | **WRITE FILE OPEN** | This message is generated when a write file that has not been closed is being opened for reading. |

| 61 | **FILE NOT OPEN** | This message is generated when a file is being accessed that has not been opened in the DOS. Sometimes, in this case, a message is not generated; the request is simply ignored. |

| 62 | **FILE NOT FOUND** | The requested file does not exist on the indicated drive. |

| 63 | **FILE EXISTS** | The file name of the file being created already exists on the diskette. |

| 64 | **FILE TYPE MISMATCH** | The file type does not match the file type in the directory entry for the requested file. |

| 65 | **NO BLOCK** | This message occurs in conjunction with the B-A command. It indicates that the block to be allocated has been previously allocated. The parameters indicate the track and sector available with the next highest number. If the parameters are zero (0), then all blocks higher in number are in use. |

66	**ILLEGAL TRACK AND SECTOR**	The DOS has attempted to access a track or block which does not exist in the format being used. This may indicate a problem reading the pointer to the next block.
67	**ILLEGAL SYSTEM T OR S**	This special error message indicates an illegal system track or sector.
70	**NO CHANNEL** (available)	The requested channel is not available, or all channels are in use. A maximum of five sequential files may be opened at one time to the DOS. Direct access channels may have six opened files.
71	**DIRECTORY ERROR**	The BAM does not match the internal count. There is a problem in the BAM allocation or the BAM has been overwritten in DOS memory. To correct this problem reinitialize the diskette to restore the BAM in memory. Some active files may be terminated by the corrective action. NOTE: BAM = Block Availability Map
72	**DISK FULL**	Either the blocks on the diskette are used or the directory is at its entry limit. DISK FULL is sent when two blocks are available on the 1541 to allow the current file to be closed.
73	**DOS MISMATCH** (73, CBM DOS V2.6 1541)	DOS 1 and 2 are read compatible but not write compatible. Disks may be interchangeably read with either DOS, but a disk formatted on one version cannot be written upon with the other version because the format is different. This error is displayed whenever an

attempt is made to write upon a disk which has been formatted in a non-compatible format. (A utility routine is available to assist in converting from one format to another.) This message may also appear after power up.

74 **DRIVE NOT READY** An attempt has been made to access the Floppy Disk Drive without any diskette present.

Introduction

TEDMON is a built-in machine language program which lets you easily write machine language programs. TEDMON includes a machine language monitor, a mini assembler, and a disassembler.

Machine language programs written using TEDMON can run by themselves, or be used as very fast 'subroutines' for BASIC programs since TEDMON has the ability to coexist peacefully with BASIC.

Tedmon Commands

A	**ASSEMBLE**	Assemble a line of 6502 code
C	**COMPARE**	Compare two sections of memory and report differences.
D	**DISASSEMBLE**	Disassemble a line of 6502 code.
F	**FILL**	Fill memory with the specified byte.
G	**GO**	Start execution at the specified address.
H	**HUNT**	Hunt through memory for all occurrences of certain bytes.
L	**LOAD**	Load a file from tape or disk.
M	**MEMORY**	Display the hexadecimal values of Memory locations.
R	**REGISTERS**	Display the 6502 Registers.
S	**SAVE**	Save to tape or disk.
T	**TRANSFER**	Transfer code from one section of memory to another.
V	**VERIFY**	Compare memory with tape or disk.
X	**EXIT**	Exit TEDMON.
.	**(period)**	Assembles a line of 6502 code
>	**(greater than)**	Modifies memory
;	**(semi-colon)**	Modifies 6502 Register displays

Location $7F8 controls whether TEDMON looks at ROM or RAM above $8000. If this location is set to 0, TEDMON displays BASIC and the KERNAL when commanded to do a disassembly or memory dump

above $8000. If this location is set to $80 TEDMON displays the RAM under BASIC and the KERNAL. This is often convenient for machine language program development. Note that location $7F8 does not affect the GO command. The GO command starts execution in the current memory map (ROM on or RAM on) regardless of the setting of location $7F8.

Using Tedmon

Enter TEDMON by typing:

MONITOR

TEDMON responds by displaying the 6502 registers and flashing the cursor. The cursor is your prompt that lets you know that TEDMON is waiting for your commands.

Command Descriptions

COMMAND: **A**

PURPOSE: Enter a line of assembly code.

SYNTAX: **A** *<address>* *<opcode mnemonic>* *<operand>*

<address> A hexadecimal number indicating the location in memory to place the opcode.

<opcode mnemonic> A standard MOS technology assembly language mnemonic, e.g. LDA, STX, ROR, etc.

<operand> The operand, when required, can be of any of the legal addressing modes. (For zero-page modes a 2 digit hex number is required whose value is less than $100. For non-zero page addresses 4 digit hex numbers are required.)

A **RETURN** is used to indicate the end of the assembly line. If there are any errors on the line, a question mark is displayed to indicate an error, and the cursor moves to the next line. The screen editor can be used to correct any errors on the line.

After a line of code is successfully assembled, the assembler prints a prompt containing the next legal memory location for an instruction, so A and the line number do not have to be typed more than once when typing assembly language programs into the Plus/4.

EXAMPLE:

.A 1200 LDX #$00
.A 1202

NOTE: A period (.) is equal to the ASSEMBLE command.

EXAMPLE:

. 2000 LDA #$23

COMMAND: **C**

PURPOSE: Compare two areas of memory

SYNTAX: **C** *<address 1>* *<address 2>* *<address 3>*

<Address 1> is a hexadecimal number indicating the start address of
the area of memory to compare against.

<Address 2> is a hexadecimal number indicating the end address of
the area of memory to compare against.

<Address 3> is a hexadecimal number indicating the start address of
the other area of memory to compare with.

If the two areas of memory are the same, then TEDMON prints a
RETURN, indicating that the second area of memory is the same
as the first. The addresses of any bytes in the two areas which are
different are printed out on the screen.

COMMAND: **D**

PURPOSE: Disassemble machine code into assembly language
mnemonics and operands.

SYNTAX: **D** [*<address>*] [*<address 2>*]

<address> A hexadecimal number setting the address to start
the disassembly.

<address 2> An optional hexadecimal ending address of code
to be disassembled.

The format of the disassembly is only slightly different than the input
format of an assembly. The difference is that the first character of a

disassembly is a period rather than an A (for readability), and the Hexadecimal of the code is listed as well.

A disassembly listing can be modified using the screen editor. Make any changes to the mnemonic or operand on the screen, then hit a carriage return. This enters the line and call the assembler for further modifications.

A disassembly can be paged. Typing a D causes the next page of disassembly to scroll onto the screen.

```
EXAMPLE:    D    3000  3004
            .    3000  A900      LDA  #$00
            .    3002  FF        ???
            .    3003  DO   2B   BNE  $3030
```

COMMAND: **F (FILL)**

PURPOSE: Fill a range of locations with a specified byte.

SYNTAX: **F** <address 1> <address 2> <byte>

<address 1> The first location to fill with the <byte>

<address 2> The last location to fill with the <byte>

<byte value> A 1 or 2 digit hexadecimal number to be written

This command is useful for initializing data structures or any other RAM area.

EXAMPLE: **F 0400 0518 EA**

Fills memory locations from $0400 to $0518 with $EA (a NOP instruction.)

COMMAND: **G**

PURPOSE: Begin execution of a program at a specified address.

SYNTAX: **G** [<address>]

<address> An optional argument specifying the new value of the program counter and address where execution is to start. When <address> is left out, execution begins at the current PC. (The current PC can be viewed using the R command.)

The GO command restores all registers (displayable by the R command) and begins execution at the specified starting address. Caution

is recommended in using the GO command. To return to TEDMON after executing a machine language program, use the BRK instruction.

EXAMPLE: **G 140C**

Execution begins at location $140C.

COMMAND: **H (HUNT)**

PURPOSE: Hunt through memory within a specified range for all occurrences of a set of bytes.

SYNTAX: **H** <address 1> <address 2> <data>

<address 1> beginning address of hunt procedure

<address 2> ending address of hunt procedure

<data> data set to search for data may be hexadecimal or an ASCII string. An ASCII is specified by preceding the first character with a single quote, eg, 'STRING. Data may be single or multiple element argument. When multiple and in hexadecimal each number must be separated by a space.

EXAMPLE: H C000 FFFF 'READ

H A000 A101 A9 FF 4C

> Search for ASCII string READing from C000 to FFFF

> Search for data $A9, $ff, $4C, from A100 to A101

COMMAND: **L (LOAD)**

PURPOSE: Load a file from cassette or disk.

SYNTAX: **L** <"filename">, <device>

<filename> is any legal Plus/4 filename inside quotes.

<device> is a hexadecimal number indicating the device to load from.

1 is cassette

---

8 is disk   (or 9, A, etc.)

The Load command causes a file to be loaded into memory. The starting address is contained in the first two bytes of the file (a program file). In other words, the LOAD command always loads a file into the same place it was saved from. This is very important in machine language work, since few programs are completely relocatable. The file is loaded into memory until the end of file (EOF) is found.

EXAMPLE: L "SCREEN", 1

reads a file from
cassette.

L "TANK", 8

reads a file from disk.

COMMAND: **M (MEMORY DISPLAY)**

PURPOSE: To display memory as a hexadecimal and ASCII dump within the specified address range.

SYNTAX: **M** [<*address 1*>] [<*address*>]

[<*address 1*>] First address of memory dump. Optional. If omitted, one page is displayed. The first byte is the last address specified.

[<*address 2*>] Last address of memory dump. Optional. If omitted, one page is displayed. The first byte is the data of [<*address 1*>].

Memory is displayed in the following format:

>A048 41 E7 00 AA AA 00 98 56 45 :A!.*..VE

Memory content may be edited using the screen editor. Move the cursor to the data to be modified and type the desired correction and hit return. If there is a bad RAM location or an attempt to modify ROM has occurred, an error flag (?) is displayed.

An ASCII dump of the data is displayed in REVERSE (to contrast the dump with other data displayed on the screen) to the right of the hex data. When a character is not printable, it is displayed as a reversed period (.).

As with the Disassembly command, you can page down by typing M and RETURN.

EXAMPLE

```
M 1C00
>1C00  41  E7  00  AA  AA  00  98  56  45   :A!.*..VE
>1C08  41  E7  00  AA  AA  00  98  56  45   :A!.*..VE
>1C10  41  E7  00  AA  AA  00  98  56  45   :A!.*..VE
>1C18  41  E7  00  AA  AA  00  98  56  45   :A!.*..VE
>1C20  41  E7  00  AA  AA  00  98  56  45   :A!.*..VE
>1C28  41  E7  00  AA  AA  00  98  56  45   :A!.*..VE
>1C30  41  E7  00  AA  AA  00  98  56  45   :A!.*..VE
>1C38  41  E7  00  AA  AA  00  98  56  45   :A!.*..VE
>1C40  41  E7  00  AA  AA  00  98  56  45   :A!.*..VE
>1C48  41  E7  00  AA  AA  00  98  56  45   :A!.*..VE
>1C50  41  E7  00  AA  AA  00  98  56  45   :A!.*..VE
>1C58  41  E7  00  AA  AA  00  98  56  45   :A!.*..VE
```

COMMAND: > (GREATER THAN sign)

PURPOSE: Can be used to set 1 to 8 memory locations at a time.

SYNTAX: > *address data byte 1* <*data byte 2 . . . 8*>

address: First memory address to set
data byte 1: Data to be put at address
<*data byte 2 to data byte 8*>: Data to be placed in the successive memory locations following the first address. Optional

EXAMPLE

>2000 08	places an 08 at location 2000
>3000 23 45 65	places a 23 at location 3000, a 45 at 3001, and a 65 at 3002

COMMAND: **R** (REGISTER DISPLAY)

PURPOSE: Show important 6502 registers. The program status regis-
ter, the program counter, the accumulator, the X and Y index registers
and the stack pointer are displayed.

SYNTAX: **R**

EXAMPLE: R
           PC SR AC XR YR SP
   ;   1002 01 02 03 04 F6

NOTE: ; (semi-colon) can be used to modify register displays in
the same fashion as > can be used to modify memory registers.

COMMAND: **S** (SAVE) ;

PURPOSE: Save the contents of memory onto tape or disk.

SYNTAX: S <"filename">,<device>,<address 1>,<address 2>

<"filename"> Any legal Plus/4 filename. To save the data, the filename
must be enclosed in double quotes. Single quotes cannot be used.

<device> Two possible devices are cassette and disk. To save on
cassette, use device 1: The device number of the Plus/4 disk drive is
usually 8. However, this can be changed (for instance, when using
more than one disk drive). See your Plus/4 DISK DRIVE MANUAL.

<address 1> Starting address of memory to be saved.

<address 2> Ending address of memory to be saved + 1. All data up
to but not including the byte of data at this address is saved.

The file created by this command is a program file. The first two bytes
contain the standing address <address 1> of the data. The file may
be recalled using the L command.

EXAMPLE: S "GAME", 8, 0400, 0BFF

Saves memory from $0400 to $0BFF onto disk.

COMMAND: **T** (TRANSFER)

PURPOSE: Transfer segments of memory from one memory
area to another.

SYNTAX: **T** <*address 1*> <*address 2*> <*address 3*>

<*address 1*> Starting address of data to be moved

<*address 2*> Ending address of data to be moved

<*address 3*> Starting address of new location (where the data will go)

Data can be moved from low memory to high memory or vice-versa. Additional memory segments of any length can be moved forward or backward any number of bytes (i.e., shifted).

EXAMPLE: T 1400 1600 1401

Shifts data from $1400 up to and including $1600 one byte higher in memory.

COMMAND: **V** (VERIFY)

PURPOSE: Verify a file on cassette or disk with the memory contents.

SYNTAX: **V** <*"filename"*>, <*device*>

<*filename*> is any legal Plus/4 filename.

<*device*> is a hexidecimal number indicating which device the file is on; cassette is 1 or 01, disk is 8 or 08, 09, etc.

The Verify command compares a file to memory contents. The Plus/4 responds VERIFYING. If an error is found, the word ERROR is added; if the file is successfully verified, the flashing cursor reappears.

EXAMPLE: V "WORKLOAD", 8

COMMAND: **X** (eXit)

PURPOSE: Exit to BASIC

SYNTAX: **X**

When the X command is given, the machine stack pointer is set to the current stack pointer value (see the R command). If this is modified in any way, after exiting to BASIC use the BASIC CLR command to reset the pointers.

SECTION 6

**Screen
Display
Codes**

The following chart lists all of the characters built into the Commodore character sets. It shows which numbers should be POKEd into screen memory (locations 3072 to 4095) to get a desired character. (Remember to set color memory—2048 to 3071) Also shown is which character corresponds to a number PEEKed from the screen.

Two character sets are available, but only one set at a time. This means that you cannot have characters from one set on the screen at the same time you have characters from the other set displayed. The sets are switched by holding down the **SHIFT** and **C⁼** keys simultaneously.

From BASIC, PRINT CHR$(142) will switch to upper-case/graphics mode and PRINT CHR$(14) switches to upper/lower-case mode.

Any number on the chart may also be displayed in REVERSE. The reverse character code may be obtained by adding 128 to the values shown.

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	T	t	20	(		40
A	a	1	U	u	21	)		41
B	b	2	V	v	22	*		42
C	c	3	W	w	23	+		43
D	d	4	X	x	24	,		44
E	e	5	Y	y	25	−		45
F	f	6	Z	z	26	.		46
G	g	7	[		27	/		47
H	h	8	£		28	0		48
I	i	9	]		29	1		49
J	j	10	↑		30	2		50
K	k	11	←		31	3		51
L	l	12	**SPACE**		32	4		52
M	m	13	!		33	5		53
N	n	14	"		34	6		54
O	o	15	#		35	7		55
P	p	16	$		36	8		56
Q	q	17	%		37	9		57
R	r	18	&		38	:		58
S	s	19	'		39	;		59

SET 1	SET 2	POKE
<		60
=		61
>		62
?		63
[graphic]		64
♠	A	65
[graphic]	B	66
[graphic]	C	67
[graphic]	D	68
[graphic]	E	69
[graphic]	F	70
[graphic]	G	71
[graphic]	H	72
[graphic]	I	73
[graphic]	J	74
[graphic]	K	75
[graphic]	L	76
[graphic]	M	77
[graphic]	N	78
[graphic]	O	79
[graphic]	P	80
[graphic]	Q	81
[graphic]	R	82
♥	S	83
[graphic]	T	84
[graphic]	U	85
⊠	V	86
○	W	87
♣	X	88
[graphic]	Y	89
♦	Z	90
[graphic]		91
[graphic]		92
[graphic]		93
[graphic]	[graphic]	94
[graphic]	[graphic]	95
SPACE		96
[graphic]		97
[graphic]		98
[graphic]		99
[graphic]		100
[graphic]		101
[graphic]		102
[graphic]		103
[graphic]		104
[graphic]	[graphic]	105
[graphic]		106
[graphic]		107
[graphic]		108
[graphic]		109
[graphic]		110
[graphic]		111
[graphic]		112
[graphic]		113
[graphic]		114
[graphic]		115
[graphic]		116
[graphic]		117
[graphic]		118
[graphic]		119
[graphic]		120
[graphic]		121
[graphic]	✓	122
[graphic]		123
[graphic]		124
[graphic]		125
[graphic]		126
[graphic]		127

Codes from 128-255 are reversed images of codes 0-127.

This appendix shows you what characters will appear if you PRINT CHR$(X), for all possible values of X. It will also show the values obtained by typing PRINT ASC("X"), where X is any character you can type. This is useful in evaluating the character received in a GET statement, converting upper/lower-case, and printing character based commands (like switch to upper/lower-case) that could not be enclosed in quotes.

ASCII and CHR$ Codes

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	0	↑	17	"	34	3	51
	1	RVS ON	18	#	35	4	52
	2	CLR HOME	19	$	36	5	53
	3	INST DEL	20	%	37	6	54
	4		21	&	38	7	55
WHT	5		22	'	39	8	56
	6		23	(	40	9	57
	7		24	)	41	:	58
DISABLES SHIFT C=	8		25	*	42	;	59
ENABLES SHIFT C=	9		26	+	43	<	60
	10	ESCAPE	27	,	44	=	61
	11	RED	28	−	45	>	62
	12	⟶	29	.	46	?	63
RETURN	13	GRN	30	/	47	@	64
SWITCH TO LOWER CASE	14	BLU	31	0	48	A	65
	15	SPACE	32	1	49	B	66
	16	!	33	2	50	C	67

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
D	68	(graphic)	97	(graphic)	126	(graphic)	155
E	69	(graphic)	98	(graphic)	127	PUR	156
F	70	(graphic)	99		128	←	157
G	71	(graphic)	100	(graphic)	129	YEL	158
H	72	(graphic)	101	FLASH ON	130	CYN	159
I	73	(graphic)	102	FLASH OFF	131	SPACE	160
J	74	(graphic)	103		132	(graphic)	161
K	75	(graphic)	104	f1	133	(graphic)	162
L	76	(graphic)	105	f3	134	(graphic)	163
M	77	(graphic)	106	f5	135	(graphic)	164
N	78	(graphic)	107	f7	136	(graphic)	165
O	79	(graphic)	108	f2	137	(graphic)	166
P	80	(graphic)	109	f4	138	(graphic)	167
Q	81	(graphic)	110	f6	139	(graphic)	168
R	82	(graphic)	111	HELP	140	(graphic)	169
S	83	(graphic)	112	SHIFT RETURN	141	(graphic)	170
T	84	(graphic)	113	SWITCH TO UPPER CASE	142	(graphic)	171
U	85	(graphic)	114		143	(graphic)	172
V	86	(graphic)	115	BLK	144	(graphic)	173
W	87	(graphic)	116	↓	145	(graphic)	174
X	88	(graphic)	117	RVS OFF	146	(graphic)	175
Y	89	(graphic)	118	CLR HOME	147	(graphic)	176
Z	90	(graphic)	119	INST DEL	148	(graphic)	177
[	91	(graphic)	120	(graphic)	149	(graphic)	178
£	92	(graphic)	121	(graphic)	150	(graphic)	179
]	93	(graphic)	122	(graphic)	151	(graphic)	180
↑	94	(graphic)	123	(graphic)	152	(graphic)	181
←	95	(graphic)	124	(graphic)	153	(graphic)	182
(graphic)	96	(graphic)	125	(graphic)	154	(graphic)	183

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
▛	184	▗	186	▜	188	▟	190
▙	185	▖	187	▛	189	▞	191

CODES	192-223	SAME AS	96-127
CODES	224-254	SAME AS	160-190
CODE	255	SAME AS	126

The following chart lists which memory locations control placing characters on the screen, and the locations used to change individual character colors, as well as showing character color codes.

Screen And Color Memory Maps

SCREEN MEMORY MAP

COLUMN

0	10	20	30	39

3072
3112
3152
3192
3232
3272
3312
3352
3392
3432
3472
3512
3552
3592
3632
3672
3712
3752
3792
3832
3872
3912
3952
3992
4032

3111

0

10 ROW

20

24

4071

COLOR MEMORY MAP
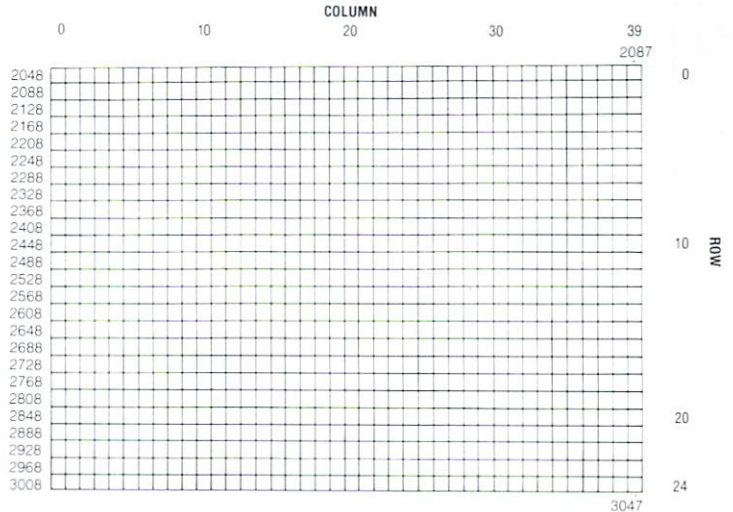
The actual values to change a character's color are:

1 BLACK	9 ORANGE
2 WHITE	10 BROWN
3 RED	11 YELLOW-GREEN
4 CYAN	12 PINK
5 PURPLE	13 BLUE-GREEN
6 GREEN	14 LIGHT BLUE
7 BLUE	15 DARK BLUE
8 YELLOW	16 LIGHT GREEN

The luminance of the color is selected by multiplying the luminance value (0-7) by 16, and added to the number for the color. To make the character flash, increase the color value by 128.

PLUS/4 Memory Register Map

REG			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	$FF00				TIMER # 1 RELOAD VALUE, BITS 0-7 (LOW)					
1	$FF01				TIMER # 1 RELOAD VALUE, BITS 8-15 (HIGH)					
2	$FF02				TIMER # 2 RELOAD VALUE, BITS 0-7 (LOW)					
3	$FF03				TIMER # 2 RELOAD VALUE, BITS 8-15 (HIGH)					
4	$FF04				TIMER # 3 RELOAD VALUE, BITS 0-7 (LOW)					
5	$FF05				TIMER # 3 RELOAD VALUE, BITS 8-15 (HIGH)					
6	$FF06		TEST	ECM	BMM	BLANK	# ROWS	Y2	Y1	Y0
7	$FF07		RVS OFF	PAL/	FREEZE	MCM	# COLS	X2	X1	X0
8	$FF08					KEYBOARD LATCH				
9	$FF09		IRQ	I-T3	NC	I-T2	I-T1	I-LP	I-RAS	NC
10	$FF0A		NC	EI-T3	NC	EI-T2	EI-T1	EI-LP	EI-RAS	RC8
11	$FF0B		RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
12	$FF0C		NC	NC	NC	NC	NC	NC	C9	CUR8
13	$FF0D		CUR7	CUR6	CUR5	CUR4	CUR3	CUR2	CUR1	CUR0
14	$FF0E		SND1-7	SND1-6	SND1-5	SND1-4	SND1-3	SND1-2	SND1-1	SND1-0
15	$FF0F		SND2-7	SND2-6	SND2-5	SND2-4	SND2-3	SND2-2	SND2-1	SND2-0
16	$FF10		NC	NC	NC	NC	NC	NC	SND2-9	SND2-8
17	$FF11		SND-REL	NOISE	V2-SEL	V1-SEL	VOL3	VOL2	VOL1	VOL0
18	$FF12		NC	NC	BMB2	BMB1	BMB0	R BANK	S1-9	S1-8
19	$FF13		CB5	CB4	CB3	CB2	CB1	CB0	SCLOCK	STATUS
20	$FF14		VM4	VM3	VM2	VM1	VM0	NC	NC	NC
21	$FF15	BKGD0	NC	LUM2	LUM1	LUM0	COLOR3	COLOR2	COLOR1	COLOR0
22	$FF16	BKGD1	NC	LUM2	LUM1	LUM0	COLOR3	COLOR2	COLOR1	COLOR0
23	$FF17	BKGD2	NC	LUM2	LUM1	LUM0	COLOR3	COLOR2	COLOR1	COLOR0
24	$FF18	BKGD3	NC	LUM2	LUM1	LUM0	COLOR3	COLOR2	COLOR1	COLOR0
25	$FF19	BKDG4	NC	LUM2	LUM1	LUM0	COLOR3	COLOR2	COLOR1	COLOR0
26	$FF1A		NC	NC	NC	NC	NC	NC	BRE9	BRE8
27	$FF1B		BRE7	BRE6	BRE5	BRE4	BRE3	BRE2	BRE1	BRE0
28	$FF1C		NC	NC	NC	NC	NC	NC	NC	VL8
29	$FF1D		VL7	VL6	VL5	VL4	VL3	VL2	VL1	VL0
30	$FF1E		H8	H7	H6	H5	H4	H3	H2	H1
31	$FF1F		NC	BL3	BL2	BL1	BL0	VSUB2	VSUB1	VSUB0
62	$FF3E					ROM SELECT				
63	$FF3F					RAM SELECT				

ADDRESS	CONTENTS	NOTES
	< - >	
$FFFE-FFFF	<IRQ VECTOR	> *
$FFFC	<RES VECTOR	> *
$FFFA	<NMI VECTOR (NOT USED)	> * ROM BANK HIGH (cont)
	<	> *
$FF81-FFF5	<KERNAL JUMP TABLE	> *
$FF40-FF80	<	>
$FF00-FF3F	<TED CHIP	> *
	<	> *
$FE00-FEFF	<DMA DISK SYSTEM	> * TED Chip and I/O
$FDE0-FDEF	<	> * space appear in
$FDD0-FDDF	<CARTRIDGE BANK PORT	> * all memory maps.
$FD10-FD1F	<6529 PARALLEL PORT	> *
$FD00-FD0F	<ACIA	> *
	<	>
$FCD0-FCFF	<	> ROM banking routines
	<	> (appear in all ROM maps)
$D800-FCFF	<	> *
	<	> *
$D000-D7FF	<CHARACTER ROM	> * ROM BANK HIGH
	<	> *
$C000-D7FF	<MORE BASIC	> *
	<	>
$8000-BFFF	<BASIC	> ROM BANK LOW
	<	>
$4000-FFFF	<RAM, ALSO START OF BASIC TEXT	>
	<AREA WHEN HIRES GRAPHICS ARE USED	>
$2000-3FFF	<BIT MAP SCREEN DATA	>
	<	>
$1C00-1FFF	<HIRES SCREEN VIDEO MATRIX	>
	<	>
$1800-1BFF	<HIRES SCREEN ATTRIBUTE BYTES	>
	<	>
$1000-	<BASIC TEXT AREA (BIT MAP OFF)	>
	<	>
$0C00-0FFF	<TEXT VIDEO MATRIX (SCREEN MEMORY)	>
	<	>
$0800-0BFF	<TEXT ATTRIBUTE BYTES (COLOR MEMORY)	>
	<	>
$0000-07FF	<SYSTEM STORAGE	>
	< - >	

*NOTE: In the 64K RAM system, RAM goes from $0000-$FCFF, and from $FF40-$FFFF.

Functions that are not intrinsic to BASIC 3.5 may be calculated as follows:

Deriving Mathematical Functions

FUNCTION	BASIC EQUIVALENT
SECANT	SEC(X)=1/COS(X)
COSECANT	CSC(X)=1/SIN(X)
COTANGENT	COT(X)=1/TAN(X)
INVERSE SINE	ARCSIN(X)=ATN(X/SQR(−X*X+1))
INVERSE COSINE	ARCCOS(X)=−ATN(X/SQR(−X*X+1))+π/2
INVERSE SECANT	ARCSEC(X)=ATN(X/SQR(X*X−1))
INVERSE COSECANT	ARCCSC(X)=ATN(X/SQR(X*X−1))+(SGN(X)−1*π/2)
INVERSE COTANGENT	ARCOT(X)=ATN(X)+π/2
HYPERBOLIC SINE	SINH(X)=(EXP(X)−EXP(−X))/2
HYPERBOLIC COSINE	COSH(X)=(EXP(X)=+EXP(−X))/2
HYPERBOLIC TANGENT	TANH(X)=EXP(−X)/(EXP(x)+EXP(−X))*2+1
HYPERBOLIC SECANT	SECH(X)=2/(EXP(X)+EXP(−X))
HYPERBOLIC COSECANT	CSCH(X)=2/(EXP(X)−EXP(−X))
HYPERBOLIC COTANGENT	COTH(X)=EXP(−X)/(EXP(X)−EXP(−X))*2+1
INVERSE HYPERBOLIC SINE	ARCSINH(X)=LOG(X+SQR(X*X+1))
INVERSE HYPERBOLIC COSINE	ARCCOSH(X)=LOG(X+SQR(X*X−1))
INVERSE HYPERBOLIC TANGENT	ARCTANH(X)=LOG((1+X)/(1−X))/2
INVERSE HYPERBOLIC SECANT	ARCSECH(X)=LOG((SQR(−X*X+1)+1)/X)
INVERSE HYPERBOLIC COSECANT	ARCCSCH(X)=LOG((SGN(X)*SQR(x*x+1)/x))
INVERSE HYPERBOLIC COTANGENT	ARCCOTH(X)=LOG((X+1)/(x−1))/2
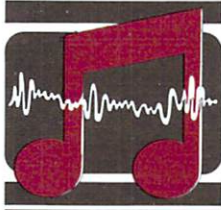
**Musical
Note
Table**

NOTE	SOUND REGISTER VALUE	ACTUAL FREQUENCY (HZ)
A	7	110
B	118	123.5
C	169	130.8
D	262	146.8
E	345	164.7
F	383	174.5
G	453	195.9
A	516	220.2
B	571	246.9
C	596	261.4
D	643	293.6
E	685	330
F	704	349.6
G	739	392.5
A	770	440.4
B	798	494.9
C	810	522.7
D	834	588.7
E	854	658
F	864	699
G	881	782.2
A	897	880.7
B	911	989.9
C	917	1045
D	929	1177
E	939	1316
F	944	1398
G	953	1575

The above table contains the sound register values of four octaves of
notes. The sound register values are used as the second parameter
of the SOUND command. To use the first note in the table (A—sound

register value 7) use the 7 as the second number after the SOUND
command—SOUND 1,7,30.

Use the following formula to find the sound register values for frequencies other than those in the table:

SOUND REGISTER VALUE = 1024—(111860.781/FREQUENCY)

Both the table of sound register values and the above formula are for
NTSC televisions. This is the television standard used throughout the
United States and all of Canada. If you are in a country where PAL is
the television standard, you should use the following formula to calculate new sound register values for the entire table:

SOUND REGISTER VALUE = 1024—(111840.45/FREQUENCY)

```
5 GRAPHIC 3, 1: GRAPHIC 0, 1
10 INPUT "SHOULD I CLEAN UP MY MESS"; A$
20 INPUT "SHOULD I ROTATE"; B$
30 INPUT "SHOULD I VARY MOTION"; C$
40 INPUT "SHOULD I PICK THE START"; D$
50 IF A$="Y" THEN DIM A (3,200)
60 DEF FNA (X) = INT(RND(1) * X)
70 IF D$="Y" THEN X1=FNA(80)+80: X2=FNA(80)+80:
   Y1=FNA(100)+100
75 IF D$="Y" THEN Y2=FNA(100)+100
80 IF D$<>"Y" THEN X1=80: X2=80: Y1=100: Y2=100
90 GRAPHIC 3: FOR L=1 TO 3: COLOR L, FNA(15)+2, FNA(8):
   NEXT
100 IF C1<1 THEN COLOR FNA(3)+1, FNA(15)+2, FNA(8):
    C1=FNA(40)+20
110 IF C2<>0 THEN 140: ELSE XA=FNA(11)-5: XB=FNA(11)-5:
    YA=FNA(13)-6
115 YB=FNA(13)-6
120 IF C$="Y" THEN C2=FNA(10)+5
130 IF B$="Y" THEN XB=-XA: YB=-YA
140 IF C3<1 THEN C=FNA(3)+1: C3=FNA(10)
145 IF A$="Y" THEN DRAW 0, A(0,P), A(1,P) TO A(2,P), A(3,P)
150 X1=X1+XA: X2=X2+XB: Y1=Y1+YA: Y2=Y2+YB
160 IF X1<0 OR X1>159 THEN XA=-XA: X1=X1+XA
170 IF X2<0 OR X2>159 THEN XB=-XB: X2=X2+XB
180 IF Y1<0 OR Y1>199 THEN YA=-YA: Y1=Y1+YA
190 IF Y2<0 OR Y2>199 THEN YB=-YB: Y2=Y2+YB
200 DRAW C, X1, Y1 TO X2, Y2
210 IF A$="Y" THEN A(0,P)=X1: A(1,P)=Y1: A(2,P)=X2:
    A(3,P)=Y2: P=P+1
215 IF A$="Y" THEN IFP>200THENP=0
220 C1=C1-1: C2=C2-1: C3=C3-1: GOTO 100
```

**Sound
Effects**

Wolf Whistle
```
10 VOL7
20 FORL=400TO800STEP20
30 SOUND1,L,3:NEXT
40 FORL=300TO600STEP40
50 SOUND1,L,3:NEXT
60 FORL=600TO300STEP-40
70 SOUND1,L,3:NEXT
```

Computer Maniac
```
10 VOL7
20 FORL=1TO100
30 SOUND1,INT(RND(0)*500)+400,4
40 NEXT
```

Telephone
```
10 VOL7
20 FORL=1TO5
30 FORM=1TO60
40 SOUND1,466,1
50 SOUND1,1020,1
60 NEXT
70 FORZ=1TO2000:NEXT
80 NEXT
```

Busy Signal
```
10 VOL7
20 FORL=1TO15
40 SOUND1,466,20
50 SOUND1,1020,15
80 NEXT
```

Bubbles
```
10 VOL7
20 GRAPHIC1,1
30 FORM=1TO50
40 GOSUB80
50 SOUND1,900-R*20,(YR/2)+50
60 CIRCLE1,X,Y,R,YR
70 NEXT:GRAPHIC0:END
80 X=INT(RND(0)*280)+20
90 Y=INT(RND(0)*160)+20
100 R=INT(RND(0)*40)+5
110 YR=R/1.3
120 RETURN
```

Zap Beam
```
10 VOL7
20 FORM=1TO 20
30 FORL=900TO850STEP-10
40 SOUND1,L,1
```

```
50 NEXT
60 FORL=850TO900STEP10
70 SOUND1,L,1
80 NEXT
100 NEXT
```

Music Lines

```
10 VOL7
15 X1=0:Y1=0
20 GRAPHIC1,1
30 GETA$:IFA$=" "THENGRAPHIC0:END
40 GOSUB80
45 FORL=1TODSTEP2
50 SOUND1,X*3,5
55 SOUND2,Y*3,5
60 DRAW1,X,Y
65 X=X+2*DX:Y=Y+2*DY
70 NEXT:GOTO30
80 X=X1:X1=INT(RND(0)*280)+20
90 Y=Y1:Y1=INT(RND(0)*160)+20
100 A=X1-X:B=Y1-Y:D=SQR(A*A+B*B)
110 DX=A/D:DY=B/D
120 RETURN
```

Introduction

Your Commodore Plus/4 has a built in RS-232 interface for connection to any RS232 modem, printer, or other device. To connect a device to your Plus/4 requires a cable and some additional programming. The Commodore modem connects directly.

RS-232 on the Plus/4 is standard RS-232 format, but the voltages are TTL levels (0 to 5V) rather than the normal RS-232 -12 to 12 volt range. The cable between the Commodore Plus/4 and the RS-232 device should accommodate the necessary voltage conversions. (The Commodore RS-232 interface cartridge handles this properly.)

The RS-232 interface software can be accessed from BASIC or from the KERNAL for machine language programming. This section addresses the use of RS-232 from BASIC. For more detailed information, and for use from machine language, consult the Commodore Plus/4 Programmers Reference Guide. RS-232 from BASIC level uses the normal BASIC commands: OPEN, CLOSE, CMD, INPUT#, GET#, PRINT#, and the reserved variable ST. INPUT# and GET# recall data from the receiving buffer, while PRINT# and CMD place data into the transmitter.

**Opening
The
RS-232
Channel**

Only one RS-232 channel should be open at any time; a second OPEN statement causes the receive buffer pointer to be reset, causing any characters in the buffer to be lost.

Up to 4 characters may be sent in the filename field. The first two are control and command register characters; the other two are reserved for future system options. Baud rate, parity, and other options can be selected through this feature.

Basic Syntax:

OPEN *lf,*2,0,*<">control register command register<">*

EXAMPLE:
OPEN 2,2,0, CHR$(5)+ CHR$(15)

lf—Normal logical file i.d. (1 – 255). If lf > 127, then linefeed follows carriage return.

control register—Single byte character (see Figure 1) (required to specify baud rate)

command register—Single byte character (see Figure 2)

Getting Data From An RS-232 Channel

When receiving data, the Commodore Plus/4 receiver buffer holds 127 characters before the buffer overflows. This is indicated by the RS-232 status word (ST from BASIC). All characters received when the buffer is full are lost. Obviously, it pays to keep the buffer as empty as possible.

Receiving RS-232 data at high speeds calls for use of machine language, since the speed of BASIC is a limitation.

CONTROL REGISTER

7	6	5	4	3	2	1	0

CONTROL REGISTER
The Control Register is used to select the desired mode for the 6551. The word length, number of stop bits, and clock controls are all determined by the Control Register, which is depicted in Figure 1.

STOP BITS

| 0 = 1 Stop Bit |
| 1 = 2 Stop Bits
1 Stop Bit if Word Length
= 8 Bits and Parity*
1½ Stop Bits if Word Length
= 5 Bits and No Parity |

WORD LENGTH

BIT		DATA WORD
6	5	LENGTH
0	0	8
0	1	7
1	0	6
1	1	5

RECEIVER CLOCK SOURCE

| 0 = External Receiver Clock |
| 1 = Baud Rate Generator |

*This allows for 9-bit transmission (8 data bits plus parity).

BAUD RATE GENERATOR

0	0	0	0	16x EXTERNAL CLOCK
0	0	0	1	50 BAUD
0	0	1	0	75
0	0	1	1	109.92
0	1	0	0	134.58
0	1	0	1	150
0	1	1	0	300
0	1	1	1	600
1	0	0	0	1200
1	0	0	1	1800
1	0	1	0	2400
1	0	1	1	3600
1	1	0	0	4800
1	1	0	1	7200
1	1	1	0	9600
1	1	1	1	19,200

	7	6	5	4	3	2	1	0
HARDWARE RESET	0	0	0	0	0	0	0	0
PROGRAM RESET	—	—	—	—	—	—	—	—

Figure 1. Control Register Format

Basic Syntax:

Recommended: **GET#** If, <string variable>

NOT Recommended: **INPUT#** If, <variable list>

COMMAND REGISTER
The Command Register is used to control Specific
Transmit/Receive functions and is shown in Figure 2.

COMMAND REGISTER

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

PARITY CHECK CONTROLS

BIT			OPERATION
7	6	5	
—	—	0	Parity Disabled—No Parity Bit Generated—No Parity Bit Received
0	0	1	Odd Parity Receiver and Transmitter
0	1	1	Even Parity Receiver and Transmitter
1	0	1	Mark Parity Bit Transmitted. Parity Check Disabled
1	1	1	Space Parity Bit Transmitted. Parity Check Disabled

DATA TERMINAL READY

| 0 = Disable Receiver and All Interrupts (DTR high) |
| 1 = Enable Receiver and All Interrupts (DTR low) |

RECEIVER INTERRUPT ENABLE

| 0 = IRQ Interrupt Enabled from Bit 3 of Status Register |
| 1 = IRQ Interrupt Disabled |

NORMAL/ECHO MODE FOR RECEIVER

| 0 = Normal |
| 1 = Echo (Bits 2 and 3 must be "0") |

TRANSMITTER CONTROLS

BIT		TRANSMIT INTERRUPT	RTS LEVEL	TRANSMITTER
3	2			
0	0	Disabled	High	Off
0	1	Enabled	Low	On
1	0	Disabled	Low	On
1	1	Disabled	Low	Transmit BRK

	7	6	5	4	3	2	1	0
HARDWARE RESET	0	0	0	0	0	0	0	0
PROGRAM RESET	—	—	—	0	0	0	0	0

*These bits must be set to the given values

Figure 2. Command Register Format

NOTES

If the word length is less than 8 bits, all unused bit(s) are assigned a value of zero.

If a GET# does not find any data in the buffer, the character "" (a null) is returned.

If INPUT# is used, then the system hangs until a non-null character and a following carriage return is received. Thus, if the CTS or DSR line(s) disappear during character INPUT#, the system hangs in a RESET-only state. This is why the INPUT# and CHRIN routines are NOT recommended.

Sending Data To An RS-232 Channel

Basic Syntax:

CMD *If*—acts same as in BASIC specifications

PRINT# *If,<variable list>*

Closing An RS-232 Data Channel

Closing an RS-232 file discards all data in the buffer at the time of execution, stops all RS-232 transmitting and receiving, and sets the RTS and Sout lines high.

Basic Syntax:

CLOSE *If*

Table 1. RS-232 Port Lines

PIN ID	DESCRIPTION	EIA	ABV	OUT
C	RECEIVED DATA	(BB)	Sin	IN
D	REQUEST TO SEND	(CA)	RTS	OUT
E	DATA TERMINAL READY	(CD)	DTR	OUT
F	RING INDICATOR	(CE)	RI	IN
H	RECEIVED LINE SIGNAL	(CF)	DCD	IN
J	UNASSIGNED	( )	XXX	IN
K	CLEAR TO SEND	(CB)	CTS	IN
L	DATA SET READY	(CC)	DSR	IN
B	RECEIVED DATA	(BB)	Sin	IN
M	TRANSMITTED DATA	(BA)	Sout	OUT
A	PROTECTIVE GROUND	(AA)	GND	
N	SIGNAL GROUND	(AB)	GND	

MODES:

```
| 7 | | 6 | | 5 | | 4 | | 3 | | 2 | | 1 | | 0 |  (Machine lang.—rsstat)
                                          :_ PARITY ERROR BIT
                                   :_____ FRAMING ERROR BIT
                            :_____ RECEIVER BUFFER OVERRUN BIT
                     :_____ UNUSED BIT
              :_____ CTS SIGNAL MISSING BIT
       :_____ UNUSED BIT
   :_____ DSR SIGNAL MISSING BIT
:_____ BREAK DETECTED BIT
```

Figure A-3. RS-232 Status Register

NOTES

If the BIT=0, then no error has been detected.

The RS-232 status register can be read from BASIC using the variable ST.

If ST is read by BASIC or by using the KERNAL READST routine the RS-232 status word is cleared upon exit. If multiple uses of the STATUS word are necessary, the ST should be assigned to another variable, i.e.

SR=ST: REM ASSIGNS ST TO SR

The RS-232 status is read (and cleared) only when the RS-232 channel was the last external I/O used.

This program opens the telecommunications channel to allow you to communicate using a modem.

```
100 OPEN 5, 2, 2, CHR$(22)+CHR$(5): REM ALLOCATE BUFFER
        AND OPEN CHANNEL
110 DIM F%(255), T%(255)
120 FOR J= 32 TO 64: T%(J)=J:NEXT
130 T%(13)=13: T%(20)=8: RV=18: CT=0
220 FOR J=65 TO 90: K=J+32: T%(J)=K: NEXT
230 FOR J=91 TO 95: T%(J)=J: NEXT
240 FOR J=193 TO 218: K=J-128: T%(J)=K: NEXT
250 T%(146)=16: T%(133)=16
255 T%(137)=3: T%(134)=17: T%(138)=19
260 FOR J=0 TO 255
270 K=T%(J)
280 IF K<>0 THEN F%(K)=J: F%(K+128)=J
290 NEXT
300 PRINT" "CHR$(147)
310 GET#5, A$
320 IF A$="" THEN 360
330 PRINT" "CHR$(157);CHR$(F%(ASC(A$)));
340 IF F%(ASC(A$))=34 THEN PRINT CHR$(27)"O";
350 GOTO 310
360 PRINT CHR$(RV)" "CHR$(157);CHR$(146);: GET A$
370 IF A$<>"" THEN PRINT#5,CHR$(T%(ASC(A$)));
380 CT=CT+1
390 IF CT=8 THEN CT=0: RV=164-RV
410 GOTO 310
```

The following lists include a sampling of the computer and programming books available. The title of the book is listed first, followed by the author and publisher.
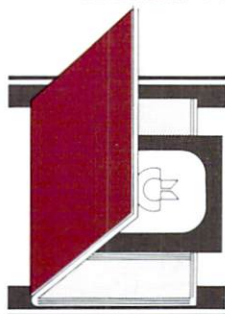
Commodore Books

Books For Commodore Products

VIC 20 Programmer's Reference Guide
Commodore 64 Programmer's Reference Guide
Commodore Plus/4 Programmer's Reference Guide
Mastering Your VIC 20
Four VIC 20 Computer Books:
VIC Revealed, Nick Hampshire
VIC Games, Nick Hampshire
VIC Graphics, Nick Hampshire
Stimulating Simulations for the VIC, C.W. Engel
Introduction to BASIC, Part 1 and 2, Andrew Colin
Commodore Software Encyclopedia, Third Edition

BASIC Programming

Armchair BASIC: An Absolute Beginner's Guide to Programming in BASIC, Fox & Fox, Osborne/McGraw-Hill
BASIC Handbook, Second Edition, Lien, Compusoft
Basic Commodore 64 BASIC, Coan, Hayden
Elementary BASIC, Ledgard & Singer, SRA
How to Build a Program, Emmerichs, Dilithium Press
Instant Freeze-Dried Computer Programming in BASIC, Brown
My Computer Likes Me When I Speak in BASIC, Albrecht, Dilithium Press
Nailing Jelly to a Tree, Willis & Danley, Dilithium Press
The Programmer's Book of Rules, Ledin & Ledin, Lifetime Learning Publishers
Technical BASIC, Kassab, Prentice-Hall

Machine Language Programming

Machine Language for Beginners, Mansfield, COMPUTE! Books
Programming the 6502, Zaks, Sybex
6502 Assembly Language Programming, Leventhal, Osborne/McGraw-Hill
6502 Micro Chart, Micro Logic Corp.
6502 Software Design, Scanlon, Sams
The 6502 Software Gourmet Guide & Cookbook, Findlay, Hayden

INDEX

U

Uppercase/graphics modes
  accessing 24, 26, 28
  printing graphics 84-89
  SHIFT key 24, 28
Upper/lowercase graphics
  accessing 24, 26, 28
  printing graphics 84-89
  SHIFT key 24, 28

V

Variables
  floating point 64, 164
  integer 64, 164
  reserved names 116-117,
    165-166
  text string 64, 73, 164
  types 64, 164-166
  see also Assigning data
VERIFY command 70, 128-129,
  171, 192
Voices 106-107
VOLume 105, 156, 171

W

WAIT statement 156, 171
Windowing 57, 86

About the Commodore ® PLUS/4 ™ User's Manual . . .

The Commodore Plus/4, the "productivity machine," is perfect
for finances, accounting or any small business application. As
a home computer, the Commodore Plus/4 has 64K of memory,
with advanced color, graphic, sound and BASIC programming
capabilities. This easy-to-read User's Manual gives you all the
information you need to set up your equipment, use software
and learn about computing with your new Commodore Plus/4.

Even if you've never used a computer before, you can follow
the step-by-step instructions and explanations to get into com-
puting right away. For those already familiar with computing, the
Commodore Plus/4 Encyclopedia contains a volume of useful
information about the Commodore Plus/4 including a complete
review of every command in the BASIC language built into the com-
puter. This manual also contains explanations of the advanced
features of the Commodore Plus/4 and tells you how to get the
most out of these expanded capabilities.

You can learn how to use the Commodore Plus/4's built-in
integrated software packages—the spreadsheet, graphics, word
processing and file manager—by reviewing the Commodore
Plus/4 Integrated Software User's Manual, also included with your
new computer.

C: commodore
COMPUTERS

310196