

C64 user's manual

commodore



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".

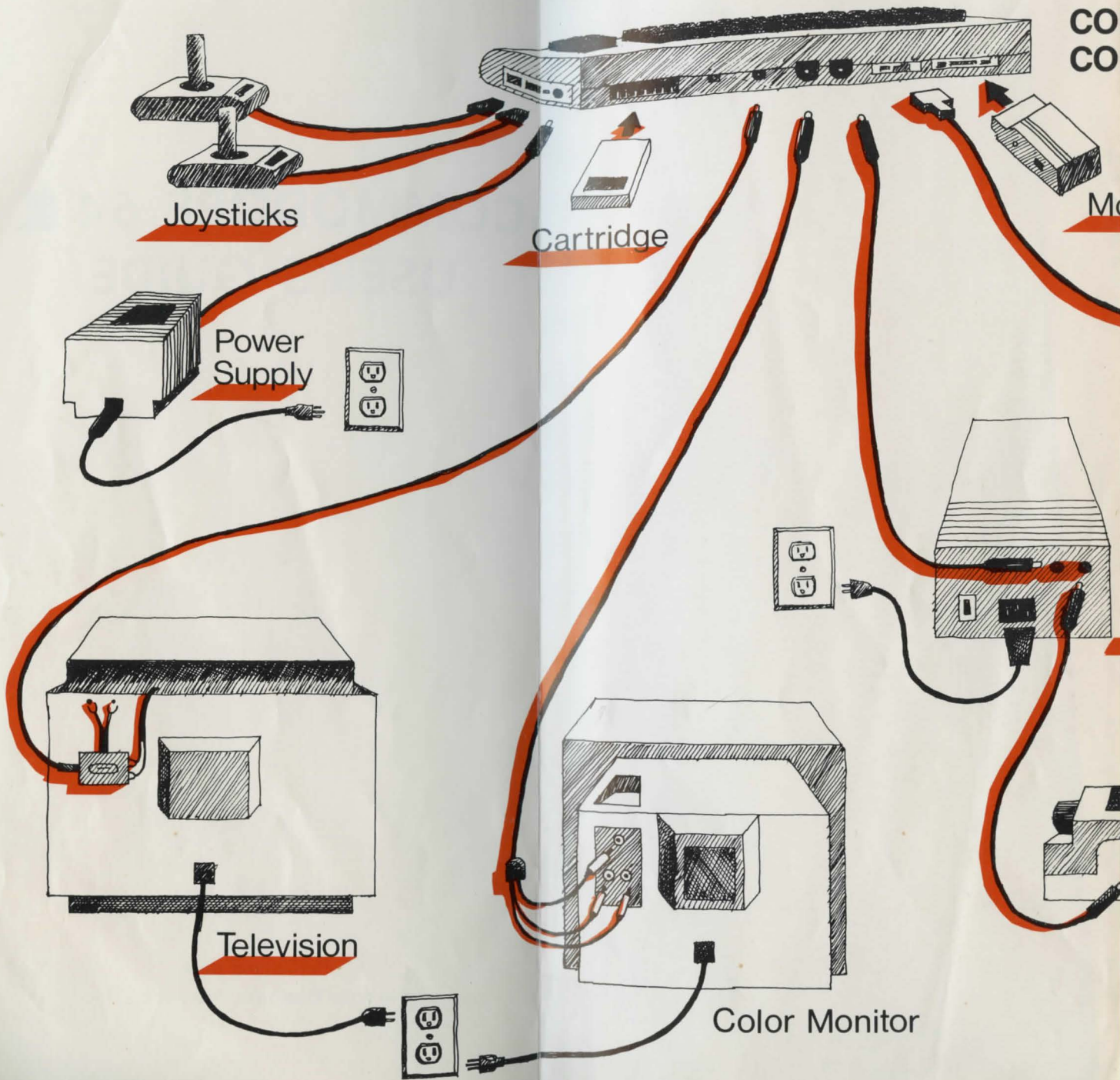
This booklet is available from the U.S. Government Printing Office, Washington, D.C. 20402, Stock No. 004-000-00345-4.

You should use only the cables, accessories, and peripherals recommended by Commodore for your Commodore 64. 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 64

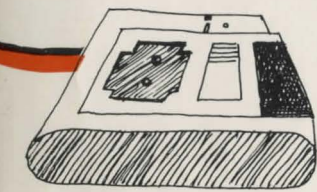
USER'S GUIDE

Published by
Commodore Business Machines, Inc.



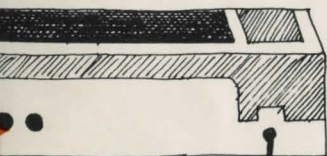
COMMODORE 64 COMPUTER

dem



Datassette

Disk
Drive



Printer



Commodore 64 Equipment Interconnection Diagram

Note 1: Connections shown are for Commodore equipment only. Connection location or type may be different for non-Commodore equipment. See your equipment manual for details.

Note 2: You can use either a TV set or a monitor as your visual display unit. Follow the diagram for whichever type of display unit you are using.

See Section 1 for more information on connecting equipment to your Commodore 64 computer.

iii	INTRODUCTION
v	HOW TO USE THIS GUIDE
1	UNPACKING AND SETTING UP How to unpack and set up your Commodore 64 computer and any accessory Commodore equipment you may have purchased with your computer
17	THE C64 KEYBOARD How to use the Commodore 64's Keyboard to enter information and perform special functions
25	USING SOFTWARE What software is and how to use it with the C64
33	BASIC—A PROGRAMMING LANGUAGE Introducing the BASIC language through some elementary commands and some simple programs
51	PROGRAMMING IN BASIC Additional BASIC commands and more sophisticated programming techniques
59	ADVANCED CONCEPTS How to use even more powerful BASIC commands, statements, functions and programming techniques
73	GRAPHICS, COLOR AND SPRITES Exploring the C64's exceptional graphics, color and animation capabilities
105	MUSIC AND SOUND Introducing the C64's versatile music and sound capabilities
115	APPENDICES
159	BASIC 2.0 ENCYCLOPEDIA
191	GLOSSARY
203	INDEX

1
2
3
4
5
6
7
8

Copyright© 1984 by Commodore Electronics Limited.

This manual is copyrighted and all rights are reserved by Commodore Electronics Limited. This document may not, in whole or in part, be copied, photocopied, reproduced, translated or reduced to any electronic medium or machine readable form without prior consent, in writing, from Commodore Electronics Ltd.

Commodore BASIC V2.0

Copyright© 1982 by Commodore Electronics Limited.
Copyright© 1977 by Microsoft, all rights reserved.

INTRODUCTION

THE COMMODORE 64

—YOUR KEY TO THE INFORMATION AGE

The Commodore 64 personal computer (more simply, the C64) is a powerful, sophisticated yet easy to use information processing system. With the C64, you can process almost any kind of information—business, personal, educational, recreational, scientific, financial and more. And with the C64 you can present this information in almost any form—words, numbers, pictures and sound.

With the wide-ranging capabilities of the C64 at your disposal, you can do all this:

- **Word Processing**—Type a draft, make changes or correct mistakes electronically, then print out a perfect final copy. Create form letters and mailing lists. Save all your material in electronic files and recall it with a few keystrokes.
- **Business Calculations**—Electronically create spreadsheets, do budgeting and payrolls, create “what if” scenarios, do complex statistical analysis, calculate tax and income data, and control your investment portfolio. Do general ledgers, accounts receivable and accounts payable. Create full-color graphs and charts based on your numerical data. Even use the C64 to balance your checkbook.
- **Data Base and File Management**—Create your own electronic files and data bases. Store and control all the letters and documents you write and all the numerical, statistical and financial data you generate. Keep track of inventories and collections. Create and update status reports. Even file recipes. (The C64 can electronically adjust a recipe that serves, say, four to serve, say, a party of 10.) In short, save, delete, change or combine any and all of your information at will.
- **Telecommunications**—Electronically “mail” almost any kind of information, almost anywhere. Access information services—like CompuServe, The Source, Dow Jones and the New York Times—for detailed information on almost any topic. Consult the World Book and other encyclopedias electronically. Send and receive personal messages and other information through computer bulletin boards. Even receive newspapers on your TV or monitor.

- **Education**—Learn a language and improve your spelling. Help your children learn math, science, English, music and other subjects, at both elementary and advanced levels. Use light pens, drawing tablets and speech synthesizers. Visit the stars through your own planetarium. Learn how to program. And note that with the C64, YOU control the pace of learning, so you can go as fast or as slow as you like.
- **Entertainment**—Play hundreds of action games and mind games. Draw pictures, make music, pursue evil villains and save fair damsels. Do all this in the comfort of your home—you never have to wait in line or pay to park the car.
- **Sound/Color/Graphics**—Control a versatile 3-voice, 9-octave sound synthesizer. Mix and match 16 colors. Create animated figures and displays. Incorporate all these features in your own programs.
- **Programming Languages**—Learn to use the powerful BASIC programming language built into the C64. Use other programming languages such as C, COBOL, COMAL, FORTH, FORTRAN, LOGO, PILOT and PASCAL, as well as machine language.
- **Interfacing with Other Equipment**—Interface with printers, disk drives, tape recorders, communications modems, video monitors, television sets, stereo equipment, video recorders, joysticks, paddle controllers, telephones, light pens, drawing tablets, numeric keypads, music synthesizers and many other types of equipment—including a robot servant!

In doing all these things, you can use the C64 in two ways:

- You can select from many prepackaged programs (software) available on cartridge, disk or tape.
- Or you can create and run your own programs.

Whatever your level of expertise, this User's Guide is designed to help you begin using your C64 quickly and easily.

• HOW TO USE THIS GUIDE

To start using your Commodore 64, follow this procedure:

- Read Section 1, UNPACKING AND SETUP. Then unpack all the equipment and set it up. Follow the directions given in Section 1 and in the overall interconnection diagram found just inside the front cover of this Guide.
- Read Section 2, THE C64 KEYBOARD. This section introduces you to the C64 keyboard, including special keys and functions.
- Read Section 3, USING SOFTWARE. This section tells you how to use software that is packaged in any of the three standard formats—cartridge, tape and disk. When you have completed this section, you will be ready to use almost any of the many commercially available software packages. In other words, you can start computing right away—**even without reading the rest of this Guide.**
- If you are interested in programming the Commodore 64 yourself, you will want to read Section 4, BASIC—A PROGRAMMING LANGUAGE, Section 5, PROGRAMMING IN BASIC, and Section 6, ADVANCED CONCEPTS. These sections describe the Commodore 64's computational capabilities and introduce the advanced BASIC programming language that is built into the Commodore 64. The sophisticated color, graphics, sound and music capabilities of the Commodore 64 are described in Sections 7 and 8. For complete details on all of these features, consult the Commodore 64 Programmer's Guide, available from your dealer and at most bookstores.
- Refer to the Appendices for a wide range of information on the C64, including a list of error messages and recommended responses; detailed technical data on color, graphics and sound; definitions of all C64 BASIC commands, statements and functions; a glossary of computing terms; a list of software available for the C64, and a list of publications on various aspects of the C64.

You can begin using your Commodore 64 as soon as you have set up and connected the equipment. How far and how fast you progress after this beginning is up to you. YOU are in control.



UNPACKING AND SETTING UP

This section tells how to unpack and set up your Commodore 64 Computer and any accessory Commodore equipment you may have purchased with your computer

Unpacking _____	3
Checking What You Received _____	3
Setting Up _____	4
Picking a Spot For Your Computer _____	5
Connecting Your Computer to a Television Set or Monitor _____	5
—Connection to a Television Set _____	5
—Connection to a Monitor _____	7
Connecting Your Computer to Electrical Power _____	7
Turning On The Computer For The First Time _____	7
If You Have a Problem . . . _____	8
Connecting Accessory Equipment _____	11
—Cassette Recorder _____	11
—Disk Drives _____	11
—Printers _____	12
—Chaining Disk Drives and Printers _____	13
—Modems _____	13
—Controllers _____	13
Typical Arrangement of Accessory Equipment and the C64 _____	14
About RAM and ROM _____	14

• UNPACKING

Since you are reading this Guide, you have probably already opened the box containing your Commodore 64 computer.

IMPORTANT!—Don't try to connect one piece of equipment to another, and don't plug anything into an electrical outlet until you have read the instructions in this section.

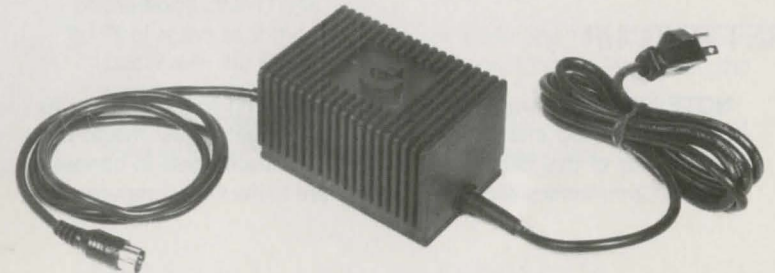
• CHECKING WHAT YOU RECEIVED

The first thing you should do is **MAKE SURE** that you received the following items in the computer box (in addition to this Guide):

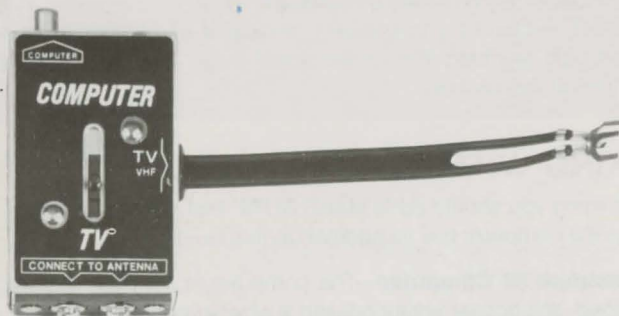
The Commodore 64 Computer—The computer is light tan in color, about 18 inches long, and comes equipped with a typewriter-style keyboard. On the back and right side of the computer there are several types of sockets and plugs that you use to connect other equipment.



Power Supply—This is a black piece of equipment about 6 inches long, 3 inches wide and 3 inches high. There are two cords connected to the power supply. One cord goes to a 3-prong electrical outlet. The other cord is a special cable that plugs into the side of the computer.



TV Switch Box—This is a small black and silver piece of equipment that is about the size of a pack of cigarettes. There is a short strip of flat TV antenna wire attached to the side of the switch box.



TV Connector Cable—This is a black cord about 10 feet long, with a male-type RCA phono plug on each end.



Warranty Card—This is a prepaid, preaddressed postcard. You should fill out and mail this card promptly to ensure that your computer is protected under the terms of the Commodore warranty.

NOTE: If any of the items listed above is missing, contact your dealer immediately. Save the boxes your equipment comes in. They will provide protection for the equipment if you move it or return it for service.

• SETTING UP

NOTE: When you are setting up and connecting your C64, refer to the large, folded interconnection diagram located just inside the front cover of this Guide. This diagram illustrates how to connect typical Commodore accessory equipment to the C64 computer.

PICKING A SPOT FOR YOUR COMPUTER

When you set up your computer equipment, pick a spot away from heat, dust, smoke or electrical interference. If possible, plug your equipment into its own separate circuit.

CONNECTING YOUR COMPUTER TO A TELEVISION SET OR MONITOR

You can connect either a standard television set (color or black and white) or a video monitor to display your computer information. (A Commodore monitor gives a sharper picture.)

Follow the illustrations in the large interconnection diagram and the instructions in the following paragraphs in making these connections.

CAUTION: Before making any connections, turn off the computer and the television set or monitor.

Connection To A Television Set

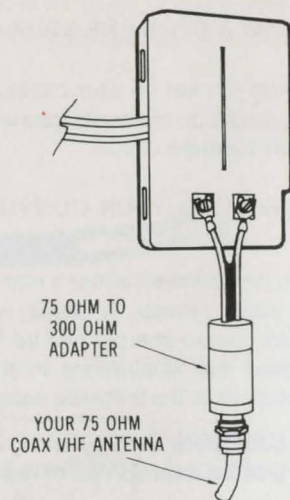
How you connect the C64 to your TV depends on what type of antenna connection your TV has. There are two basic types of antenna connections:

- Flat 300-ohm wire
- Round 75-ohm coaxial cable

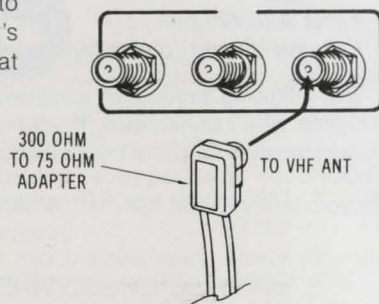
The following procedure assumes that your television set uses the flat 300-ohm wire connections. If your set uses 75-ohm connections, see the Notes following Step 6 of the procedure.

1. Disconnect the VHF antenna wires from the VHF terminal of the TV.
2. Insert these wires under the screws at the bottom of the switch box, marked CONNECT TO ANTENNA.
3. Connect the short wire, coming from the side of the switch box and marked TV VHF, to the VHF terminal of the TV.
4. Slide the selector switch on the switch box to the position marked COMPUTER.
5. Plug one end of the TV connector cable (the 10-foot long black cable with the phono plug at each end) into the TV jack on the back of the C64.
6. Plug the other end of the TV connector cable into the jack marked COMPUTER at the top of the switch box.

NOTE—Step 2: If your antenna cable is the round 75-ohm type, you will need to use a 75-ohm to 300-ohm adapter (not supplied) to attach your VHF antenna cable to the screws at the bottom of the switch box. See diagram at right.

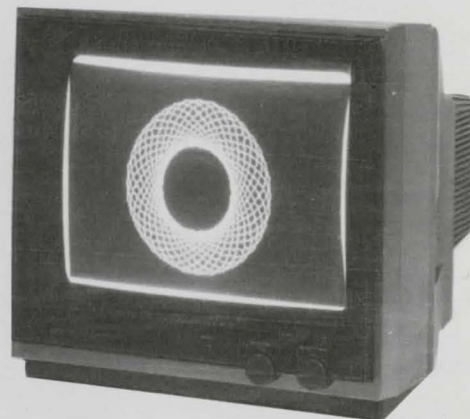


NOTE—Step 3: If your television set's antenna terminals are also round 75-ohm types, you will need to use a 300-ohm to 75-ohm adapter to attach the switch box to your set's VHF terminals. See the diagram at right.



Connection To A Monitor

You connect a monitor to your C64 through the audio/video connector on the back of the computer. The kind of cable you use depends on the type of monitor you have and the type of audio/video connector your C64 has. The interconnection diagram at the front of the Guide shows a Commodore monitor connected to the C64. If you have a monitor, consult your user's manual for full installation and operation instructions.



CONNECTING YOUR COMPUTER TO ELECTRICAL POWER

To connect your computer to electrical power, plug the end of the round power supply cable into the computer power socket (the back connection on the right side of the computer). Make sure the computer power switch (located on the right side of the computer, next to the power socket) is set to OFF.

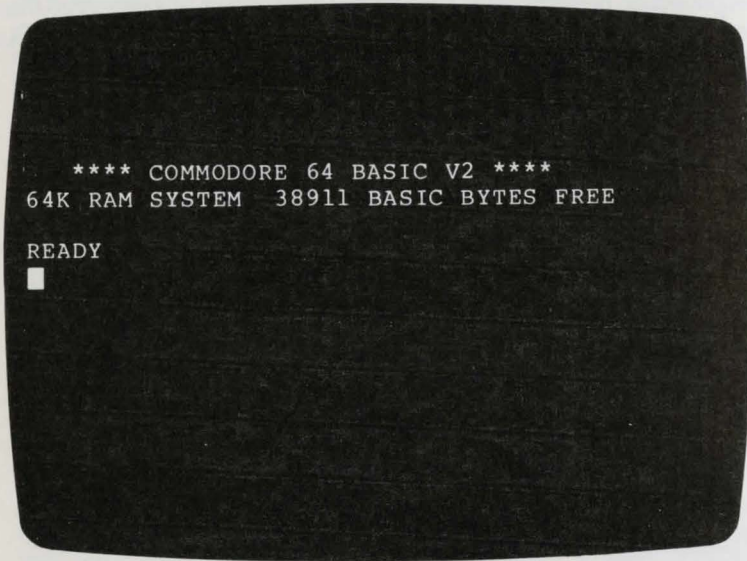
Next, insert the 3-prong plug from the power supply into a 3-hole electrical outlet.

TURNING ON THE COMPUTER FOR THE FIRST TIME

Make sure that you have connected the computer properly to a television set or monitor. Turn on the television set or monitor. If you are using a television set, set the channel selector switch on the back of the computer to either channel 3 or channel 4—whichever is not used in your area. (If you are using a monitor, you don't need to set this switch, since it does not affect the monitor.) Then set the computer power switch (located on the right side of the computer) to ON. The small red light on the top right side of the computer should come on.

Initial Screen Display

Shortly after you turn on your computer, you should see a display like this on your television set or monitor:



The screen has a 25-line display and up to 40 characters can be displayed on each screen line.

The Cursor

Notice the small flashing rectangle at the upper left part of the screen, just below the word READY. This rectangle is called the cursor. The cursor marks your position on the screen. When you type in something or when the computer responds to something you have typed in, the cursor moves accordingly.

IF YOU HAVE A PROBLEM...

If the screen display is not clear, adjust the controls on your television set or monitor. If you don't get a picture at all, check all your connections. Use the troubleshooting chart on the next page as a guide.

Symptom	Cause	Remedy
Indicator Light not "On"	Computer not "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 (3 or 4)
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
Picture with poor color	Bad color adjustment on TV	Adjust color/hue/brightness controls on TV
Sound with excess background noise	TV volume too high	Adjust volume of TV

Symptom	Cause	Remedy
Picture OK, but no sound	TV volume too low	Adjust volume of TV
	Aux. output not properly connected	Connect sound jack to aux. input on amplifier and select aux. input
Computer stuck; cursor not flashing	Computer inadvertently received instructions to disable keyboard; or the printer, cassette or disk drive is in listening mode	While depressing the RUN/STOP key press RESTORE key twice; or reset the accessories by turning off and on; or reset the computer off and on.
Computer displays garbled symbols on the screen	Overheating	Pull plug on power supply when not using computer for extended periods (overnight).

• CONNECTING ACCESSORY EQUIPMENT

In addition to the television set or video monitor used for the display, you can connect various types of accessory equipment (known as peripheral equipment, or simply peripherals) to your C64. Some commonly used peripherals are described in the following paragraphs.

NOTE: Refer to the large interconnection drawing for illustrations of how to connect peripheral equipment.

CASSETTE RECORDER

A special Commodore cassette recorder called a Datassette provides an easy and inexpensive way to save information entered in the computer, or to supply information to the computer. In the interconnection diagram, notice that you connect the Datassette recorder to the C64 through the cassette port on the back of the computer. (Note: To avoid conditions that could adversely affect Datassette performance, always make sure that the recorder is at least two feet from the television set or monitor, or any other equipment—such as stereo components and speakers—that can generate electromagnetic interference.)

If you have a Datassette, consult your user's manual for full installation and operation instructions.



DISK DRIVES

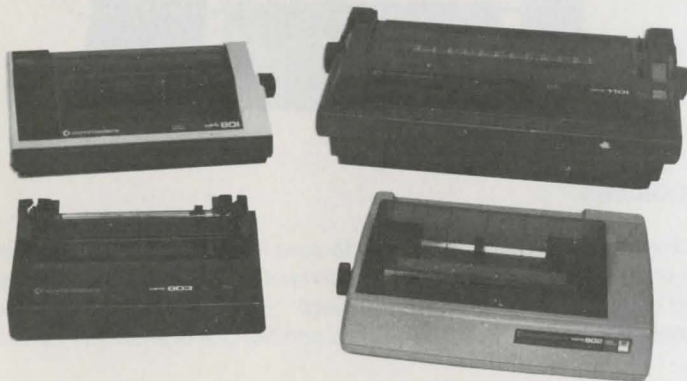
A disk drive is another, faster way to save information entered in the computer or to supply information to the computer. The information is saved or stored on 5-inch disks (sometimes called "floppies"). A typical connection between a Commodore 1541 disk drive and the C64 is shown in the intercon-

nection diagram. Note that you connect the disk drive to the C64 through the serial port on the back of the computer. If you have a disk drive, consult your user's manual for full installation and operation instructions.



PRINTERS

A printer can provide a printed copy ("hard copy") of information that is in the computer or stored on disks or tapes. A connection between a Commodore MPS-802 printer and the C64 computer is shown in the interconnection diagram. Note that in this example the printer is connected to the C64 through the serial port on the back of the disk drive. This type of multiple connection, called "daisy chaining," is described in the next paragraph. You can also connect a printer directly to the C64 by using the serial port on the back of the computer console. If you have a printer, consult your user's manual for full installation and operation instructions.



CHAINING DISK DRIVES AND PRINTERS

You can use the "daisy chaining" (or simply "chaining") technique to connect up to five disk drives or printers to the C64 computer at one time. In this technique, you connect a cable from one serial port of a printer or disk drive to a serial port of another disk drive or printer. Note that a disk drive must be the item of equipment that is directly connected to the C64's serial port.

MODEMS

A modem allows you to use your C64 computer to communicate over telephone lines with other computer users, as well as with information services and computerized bulletin boards. A connection between a Commodore modem and the Commodore 64 computer is shown in the interconnection diagram. Note that you connect the modem to the C64 through the parallel user port in the back of the computer. If you have a modem, consult your user's manual for full installation and operation instructions.



CONTROLLERS

Controllers are devices that allow you to direct computer activities by hand. Controllers include joysticks, paddles and trackballs. Although joysticks are generally associated with computer game activities, they are being used more and more in business and especially educational software programs. Controllers are connected to the C64 through the two game ports on the right side of the computer. Check your software instructions to see which port you should be using.

• TYPICAL ARRANGEMENT OF ACCESSORY EQUIPMENT AND THE C64

Shown connected to the C64 in the accompanying diagram is a grouping of some of the popular Commodore peripheral equipment. This equipment includes a Commodore color monitor, a single disk drive, a Datasette recorder, an AUTOMODEM, a serial printer and a pair of joysticks.



• ABOUT RAM AND ROM

You have probably read or heard the terms RAM (Random Access Memory) and ROM (Read Only Memory). These are the two types of memory used by a computer in processing information. The C64 has 64 kilobytes of RAM. This means that the C64 can hold about 64,000 characters (bytes) of information in its memory. About 39,000 bytes are directly available for use by you. The rest of RAM is used by the C64 in carrying out your instructions and running your programs.

The Commodore 64 also has 20 kilobytes of ROM (Read Only Memory). ROM can only be used by the computer itself to perform and control its internal activities. This memory cannot be changed by you, although there are methods that you can use to see what values are in ROM.

The computer keeps track of how much RAM you have used and how much you have left. The computer also keeps track of the contents and sta-

tus of ROM. So, unless you are interested in creating your own programs, you generally need not be concerned about RAM and ROM.


You should now be able to set up, plug in and turn on all your equipment. Check the equipment connections you make against the overall interconnection diagram in the front of this Guide. This diagram shows a typical setup for an all-Commodore equipment package. You should also refer to the manuals included with any peripheral equipment you may have purchased.

The next section of this Guide describes the Commodore keyboard, telling you how to use some special keys that make the C64's keyboard slightly different—and far more powerful—than that of a standard typewriter.



THE C64 KEYBOARD

This section tells how to use the Commodore 64's keyboard to enter information and to perform special functions

What the Keyboard Is Used For _____	19
Keyboard Modes _____	20
Functions of the Special Keys _____	20
—RETURN _____	20
—SHIFT _____	20
—SHIFT LOCK _____	21
—CRSR _____	21
—INST/DEL _____	21
—CTRL _____	22
—RUN/STOP _____	22
—RESTORE _____	23
—CLR HOME _____	23
—COMMODORE KEY () _____	23
Printing Graphic Characters _____	24
Programmable Function Keys _____	24

• WHAT THE KEYBOARD IS USED FOR

Using the keyboard to enter information is probably the most important method of communicating with your Commodore 64 computer. You use the keys to tell the computer what you want it to do, and to reply to any messages or questions the computer displays on the screen. (These messages and questions are sometimes called "screen prompts" or simply "prompts".)

Most of the letter, number and punctuation keys on the Commodore 64's keyboard look and work like the corresponding key on a standard typewriter. In addition, many of these keys can produce special graphic symbols, which are indicated on the front of the keys. There are also special keys that let the Commodore 64 computer do much more than a typewriter can do. The keyboard illustration shown below points out these special keys. The following paragraphs tell how to use the keys.


Feel free to experiment at the keyboard. There is little chance that anything you do at the keyboard can cause harm and you will benefit from the "hands on" experience.



• KEYBOARD MODES

The C64 keyboard has two typing modes:

- Upper case/graphic mode
- Upper/lower case mode

When you turn on the C64, the keyboard is in the upper case/graphic mode, which means that everything you type is in capital letters. To switch back and forth between modes, you must press the SHIFT Key and the  Key (the COMMODORE key) at the same time.

You do not have to be an accomplished typist to use the C64 effectively. You only need to know the general keyboard layout, including the location and function of the special keys described in this section.

• FUNCTIONS OF THE SPECIAL KEYS

RETURN

Pressing the RETURN key sends what you type into the Commodore 64 computer's memory. Pressing the RETURN key also moves the cursor (the small flashing rectangle that marks where you are on the screen) to the next line.

SHIFT

The SHIFT key works like the shift key on a regular typewriter: that is, when the SHIFT key is held down, it lets you print capital letters or the top characters on double character keys. The SHIFT key is also used with some other special function keys.

When the keyboard is in the upper case/graphic mode, you can use the SHIFT key to print the graphic symbols or characters that appear on the fronts of certain keys. To do this, you hold down the SHIFT key while you press the key with the graphic character you want to print or display. (Note: You can only print the graphic character on the right side of the key when you are in the upper case/graphic mode.)

When you are using the four large keys at the right side of the keyboard (marked f1, f3, f5 and f7 on the top), you must hold down the SHIFT key to activate the functions associated with the markings on the front of the keys (f2, f4, f6 and f8).

SHIFT LOCK

This key locks the SHIFT key in the ON position.

CRSR

There are two keys that let you move the cursor:

↑
CRSR moves the cursor up and down

↓
←
CRSR moves the cursor left and right
→

You don't have to keep tapping a CRSR key to get it to move more than one space. Just hold the CRSR key down and the cursor will continue to move until it reaches the position you want it to be in. Remember that you also must hold down the SHIFT key at the same time if you are moving up or to the left.

INST/DEL

This is a dual purpose key. INST stands for INSerT, and DEL stands for DELete.

Inserting Characters

You must use the SHIFT key with the INST/DEL key when you want to insert characters in a line. If you've left some characters out of a line, use the CRSR keys to move the cursor back to the error, like this:

```
WHILE U WERE OUT  
WHILE ■ WERE OUT
```

Then, while you hold down the SHIFT key, press the INST/DEL key until you have enough space to add the missing characters. INST doesn't move the cursor; it adds space between the cursor and the character to its right, like this:

```
WHILE ■ U WERE OUT  
WHILE YOU WERE OUT
```


Deleting Characters

When you press the DEL key, the cursor moves back a space and erases the character that is there, like this:

```
PRINT "ERROR" # ■  
PRINT "ERROR" ■
```

When you DELETE in the middle of a line, move the cursor just to the right of the character you want to DELETE, like this:

```
FIX IT AGAIN ■ SAM  
FIX IT AGAIN ■ SAM
```

Then press the DEL key. The characters to the right automatically move over one space to the left to close up the space and you get the correct wording, like this:

```
FIX IT AGAIN, SAM
```

Using INSerT and DELeTe Together

You can use the INSerT and DELeTe functions together to fix wrong characters. Just move the cursor to the incorrect characters and press the INST/DEL key by itself to delete the characters. Then press the SHIFT key and INST/DEL key together to add any necessary space. Then type in the corrections.

CTRL

The ConTRoL key is used with other keys to let you do special tasks called control functions. To perform a control function, you hold down the CTRL key while you press some other key. Control functions are commonly used in prepackaged software such as a word processing system.

One control function that is used often is setting colors. To set a color, you hold down the CTRL key while you press the numbered key (1 through 8) that controls the color you want. (You can get additional colors by using the **C** key in the same way.)

RUN/STOP

This is another dual purpose key.

Under certain conditions you can halt a program that is running or a print-out that is in progress by pressing the RUN/STOP key. In using the STOP function, you need only press the RUN/STOP key by itself. Most prepackaged software programs disable the STOP Function of the RUN/STOP key. This

avoids the problem of a program being stopped accidentally, with a possible loss of much valuable data.

When you want to use the RUN function of the RUN/STOP key, you must also use the SHIFT key. For instance, you can RUN a program automatically from a cassette recorder by pressing the RUN/STOP and SHIFT keys at the same time.

RESTORE

The RESTORE key is used with the RUN/STOP key to return the computer to its normal conditions (also known as the default conditions). For example, the normal or default screen color is blue. Suppose you have colored letters on the screen (which you can get by using the SHIFT or **C** keys and the number keys from 1 through 8). If you then press the RESTORE and RUN/STOP keys together, the screen is cleared and returned to its normal blue color, and the READY message is displayed.

Most prepackaged software programs also disable the RESTORE key along with the RUN/STOP key. Again, this eliminates the problem of a program being accidentally stopped and then perhaps restarted in such a way as to garble or destroy important information.

CLR HOME

CLR stands for CLear. HOME refers to the upper left corner of the screen, which is called the HOME position. When you use the SHIFT key with the CLR HOME key, the screen CLears and the cursor returns to the HOME position. When you use CLR HOME key by itself, the cursor returns to the HOME position, but the screen is not CLearRed.

COMMODORE KEY (**C**)

The **C** key (known as the COMMODORE key) has two functions:

1. It lets you switch back and forth between the upper/lower case display mode (the letters and characters on the tops of the keys) and the upper case/graphic display mode (capital letters and the graphics on the fronts of the keys). To switch modes, press the **C** key and the SHIFT key at the same time.
2. The **C** key also lets you use a second set of eight colors. To get these colors, hold down the **C** key while you press one of the number keys, (1-8) in the top row for the color you want.

• PRINTING GRAPHIC CHARACTERS

To print the graphic symbol on the right side of a key, hold down the SHIFT key while you press the key that has the graphic character you want to print. Remember that you can only print the right side graphic characters when you are in the upper case/graphic mode.

To print the graphic character on the left side of a key, hold down the **C-** key while you press the key that has the graphic character you want to print. You can print the left side graphic in either mode.

• PROGRAMMABLE FUNCTION KEYS

The four large keys on the right side of the keyboard marked f1, f3, f5 and f7 on the tops and f2, f4, f6 and f8 on the fronts, are function keys that can be programmed to perform a variety of tasks. See the discussion of the GET command in Sections 5 and 6 for details.

Now that you have successfully connected your C64 equipment and can find your way around the keyboard, you are probably ready and anxious to begin using your computer. The next section tells how to use various types of peripheral equipment to load and run prepackaged software programs.



USING SOFTWARE

This section tells what software is and how to use it with your Commodore 64 Computer

What Software Is _____	27
Software Package Formats _____	27
—Cartridge _____	27
—Tape _____	27
—Disk _____	27
What's in a Software Package _____	27
Loading and Running Software _____	28
—Loading Cartridge Software _____	28
—Loading Prepackaged Cassette Tape Software _____	28
—Loading Your Own Tape Programs _____	29
—Disks and Disk Drives _____	30
—Loading Disk Software _____	30
Hints on Selecting Software _____	32
For More Information _____	32

• WHAT SOFTWARE IS

Software is a set of instructions (also called a program) that tells your computer just what you want it to do.

There are many thousands of prepackaged or "canned" software programs available to you today. This software is what lets you do all those processing activities mentioned in the INTRODUCTION to this Guide.

Most software comes from commercial software companies. There are also many software programs available in computer magazines or from computer user groups. You can even create your own software by using a programming language like BASIC, as described in Sections 4, 5 and 6 of this Guide.

• SOFTWARE PACKAGE FORMATS

Software is packaged in three formats:

Cartridge—This is a package about the size of a deck of cards. The cartridge format is used for many games and for a considerable amount of business and educational software. The cartridge is easily inserted into a special cartridge slot on the computer.

Tape—This is a standard size audio cassette using either standard audio tape or special computer digital tape. The cassette is used with a special cassette recorder.

Disk—This is a 5-1/4 inch disk resembling a 45 rpm record, and enclosed in a square protective envelope. The disk is inserted in a device called a disk drive.

• WHAT'S IN A SOFTWARE PACKAGE

A typical software package consists of the computer program, contained on a cartridge, tape or disk, along with printed instructions that tell you such things as what the program does, how to load and run it, how to enter information, and what displays, reports or other output the program produces. The amount of instructions supplied with the software package depends on the complexity of the program. These instructions can range from a few pages to a complete manual.

Most commercial software is "protected" by special techniques to prevent unauthorized copying.

• LOADING AND RUNNING SOFTWARE

LOADING CARTRIDGE SOFTWARE

A cartridge is easily inserted into the cartridge port in the back of the C64. This port has a special slotted arrangement that accepts a cartridge only one way—with the title up. Insert the cartridge firmly but do not force it. The cartridge should click into place when properly inserted.

Follow these steps to load cartridges:

1. Turn OFF your computer.

YOU MUST TURN OFF YOUR C64 COMPUTER BEFORE YOU INSERT OR REMOVE CARTRIDGES. IF YOU DON'T, YOU MAY DAMAGE THE CARTRIDGE AND THE COMPUTER.

2. Insert the cartridge in the slot on the back of your computer.
3. Turn on your computer.
(Your cartridge will load automatically at this point.)
4. Follow the directions given on the screen or in the printed instructions for the cartridge.

LOADING PREPACKAGED CASSETTE TAPE SOFTWARE

NOTE: If you are using a Datassette tape recorder, remember to keep it at least two feet away from any equipment that could cause electrical interference.

Follow these steps to load prepackaged cassette tapes:

1. Make sure that the Datassette is plugged into the cassette port on the back of the C64.
2. Insert the tape cassette into the Datassette and close the tape compartment door.
3. Rewind the tape to the beginning of the first side, if necessary.
4. Type:

LOAD

The computer responds by telling you to:

PRESS PLAY ON TAPE

5. At this point, the screen goes blank until the computer finds the program. When the program is found, the computer displays this message:

FOUND PROGRAM NAME

6. Press the **C** key. Your prepackaged program is LOADED into the computer.
(If for some reason you want to stop the program from LOADING, press the RUN/STOP key.)
7. The program will either start to run by itself, or you will be instructed to type RUN and press RETURN to start program operation.

NOTE: Many prepackaged cassette programs may take 10 to 15 minutes to load. You will know that loading is complete when you see either a blinking cursor or program instructions on the screen.

LOADING YOUR OWN TAPE PROGRAMS

The procedure for loading tape programs that you have saved on tape yourself is essentially the same as the procedure for loading prepackaged tape software. The major difference is that you may have to specify the name of your tape program. You do this by entering:

LOAD "PROGRAM NAME"

Here, PROGRAM NAME is the name of your program. Notice that you must enclose your program name in quotation marks.

The computer searches the tape for the program named. When the program is located, the computer screen displays the message:

FOUND PROGRAM NAME
LOADING

When loading is completed, the screen displays the message:

READY.

AT the cursor position, you type:

RUN

and press RETURN. The C64 then runs your program.

NOTE: If the entire tape runs to the end without the FOUND message being displayed, rewind the tape and try again.

DISKS AND DISK DRIVES

Disks (also known as diskettes, floppy disks or simply "floppies") are fast, easy-to-use data storage devices. Disks must be inserted into a device known as a disk drive in order to store or provide information.

When using a disk drive, you should make sure that the drive's power cord is plugged into an electrical socket, and that the cable connecting the disk drive to the C64 is plugged into the serial port on the back of the computer. As with the Datassette tape recorder, you should keep the disk drive at least two feet away from any possible sources of electrical interference.

There are two small indicator lights on the front of the disk drive.

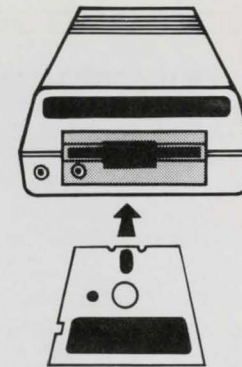
- The green light is the power light, indicating whether the disk drive is on or off.
- The red light tells you two things:
 1. When a program is being **LOAD**ed or **SAVE**d, the red light is lit while the disk is spinning in the drive. When the red light goes off, the **LOAD** or **SAVE** is complete.
 2. If there is a problem with the disk or drive, the red light flashes on and off, even after the disk stops spinning in the drive.

You can also use disks to store programs that you write yourself, and these disks can then be used to load the program back into your C64 whenever you want to run them. See Section 4 for additional commands that you can use if you plan to create, save and run your own programs.

LOADING PREPACKAGED DISK SOFTWARE

Whether you are loading preprogrammed disks or disks that you have programmed yourself, the steps are basically as follows:

1. Insert a disk into your disk drive, as shown in the diagram. Make sure the label on the disk is facing up. Put the disk in so that the labeled end goes in last. Look for a little notch on the disk (it might be covered with a little piece of tape). This notch must be on the left side as you put in the disk, assuming that you're facing your computer. Be sure the disk is all the way in.



2. Close the protective gate on the disk drive after you insert the disk. Just push down the lever.
3. Type the following:

```
LOAD "PROGRAM NAME",8
```

Here, the number 8 tells the computer that you're loading a disk.

NOTE: You can **LOAD** the first program on the disk by using the * sign in place of the program name, as follows:

```
LOAD"*",8
```

4. Press the **RETURN** key. The disk will spin and your screen will display this message:

```
SEARCHING FOR PROGRAM NAME  
LOADING
```
5. Type **RUN** when the screen says **READY** and the cursor appears. Your software is ready to use.

In some cases, prepackaged software may have its own special format for the **LOAD** command. Also, some commercial software may have an automatic **RUN** function built into the program. Check the software instructions carefully, especially if you have any problems.



BASIC— A PROGRAMMING LANGUAGE

This section introduces the BASIC language through some elementary commands and some simple programs

The Basic Programming Language	35
Typing Basic Programs	35
The PRINT Command	36
Order of Operations	37
Printing Text	38
Direct Mode Functions	39
Mathematical Functions	40
Program Mode	41
Constants, Variables and Strings	42
IF - THEN	45
Editing Tips	46
Storing and Reusing Your Program	47
Formatting a Disk	47
The SAVE Command	47
The LOAD Command	48
Displaying Your Program	49
Other Disk-Related Commands	49
Sample Program	50

• HINTS ON SELECTING SOFTWARE

Because of the great number of software products available, and because of the variety of claims made for those products, selecting the right software can be a difficult task. Here are a few tips on choosing software:

- Analyze your needs. Try to find software that meets those needs. Avoid frills.
- If possible, talk to someone who has used the software you are considering.
- Purchase the software from a reputable dealer.
- Try out the package on an equipment setup identical or similar to your own.

One way to ensure compatibility between your C64 and the software you buy is to select from the wide variety of software produced by Commodore. A list of currently available Commodore software is given in Appendix B.

• FOR MORE INFORMATION . . .

The information supplied to this point in this Guide will allow you to load and run prepackaged software in cartridges, tape and disk formats. However, if you would like more information on LOADing, SAVEing and RUNning software packages, refer to the instructions contained in the individual software packages and to the information in your equipment manuals. You should also read the next section of this Guide and consult the Commodore 64 Programmer's Reference Guide, available from your dealer or at most bookstores.

You should now be able to run a wide variety of prepackaged software, using your C64 computer and peripheral equipment. You will find that you can derive great benefit and enjoyment from using these products.

The remaining sections of this GUIDE are addressed to those newcomers to computing who are interested in learning to create their own programs, and to those experienced programmers who want to explore further the considerable capabilities of the Commodore 64 personal computer.

Your Commodore 64 computer is a powerful tool, with many capabilities. It can perform computations in a fraction of a second. It has the ability to make decisions and repeat commands according to your instructions. It can print text on a screen or printer. Up to 1000 characters can be printed on a single screen, which is 40 characters wide by 25 lines high.

• THE BASIC PROGRAMMING LANGUAGE

BASIC is a language with its own vocabulary (called commands, statements and functions) and its own rules of structure (called syntax). A set of instructions in BASIC is called a program. Each instruction in a program is identified by a line number. You can use the BASIC programming language to instruct your computer to perform many functions. Using BASIC, you can communicate with your computer in two ways: in the DIRECT mode, or in the indirect or PROGRAM mode.

Your Commodore 64 is ready to accept BASIC commands in DIRECT mode as soon as you turn it on. In the DIRECT mode, you type commands on the keyboard and enter them into the computer by pressing the RETURN key. The computer executes all commands in DIRECT mode immediately after you press the RETURN key. As you will see, your Commodore 64 can be used as a sophisticated calculator in DIRECT mode.

• TYPING BASIC PROGRAMS

You can type and use BASIC programs without knowing BASIC. However, you must type carefully, because a typing error may cause the computer to reject your information. The following hints will help minimize typing errors when typing or copying a program listing:

1. Spacing between words is not critical; e.g., typing FORT = 1T010 is the same as typing FOR T = 1 TO 10. However, a BASIC word itself must not be broken up by spaces. (See the BASIC Encyclopedia in the back of this Guide for a list of BASIC words.)
2. Any characters can be typed in quotes. Reverse graphic characters within quotes have special control functions.
3. Be careful with punctuation marks. Commas, colons and semicolons also have special functions.
4. Always press the RETURN key (indicated in this Guide by the symbol **RETURN**) after completing a line. Never exceed 80 characters in a line (two screen lines).
5. Distinguish clearly between I and 1 and between 0 and O.

- The computer ignores anything following the letters REM on a program line. REM stands for REMARK. You can use the REM statement to put descriptive comments in your program.

Concentrate on typing short programs until you are familiar with BASIC.

• THE PRINT COMMAND

The PRINT command tells the computer to display information on the screen. You can print numbers or letters, but the grammatical rules are different for each. To print numbers, simply use the PRINT command, followed by the number(s) you want to print. Try typing this on your computer:

```
PRINT 5 RETURN
PRINT 5,6 RETURN
```

Notice the numbers did not display on your screen until you pressed RETURN. The comma in the second PRINT command tells the computer you want to print more than one number. When the computer encounters a comma in a string of numbers in a PRINT statement, the computer prints each succeeding number (in this case, the 6) ten spaces to the right of the preceding number (in this case, the 5). If you don't want all these extra spaces, use a semicolon (;) instead. The semicolon causes the computer to print numbers in a PRINT command three spaces apart. You can print as many numbers as you can fit into two lines on your screen (that is, up to 80 characters). Try typing:

```
PRINT 5;6 RETURN
PRINT 100;200;300;400;500 RETURN
```

You can use the PRINT command the same way to perform calculations like addition and subtraction. Just type the calculation after the PRINT command. Try some of these:

```
PRINT 6 + 4 RETURN
PRINT 50 - 20 RETURN
PRINT 10 + 15 - 5 RETURN
PRINT 75 - 100 RETURN
PRINT 30 + 40, 55 - 25 RETURN
PRINT 30 + 40; 55 - 25 RETURN
```

Notice the fourth calculation resulted in a negative number. Also notice that you can tell the computer to make more than one calculation with a single PRINT command. And as was noted previously, you can use either a

comma or a semicolon in your command, depending on whether or not you want your results spread out.

Find the asterisk key (*) on the right side of your keyboard. This is the symbol for multiplication. Use the (/) located next to the right SHIFT key for division. Type.

```
PRINT 5 * 3 RETURN
PRINT 100 / 2 RETURN
```

The up arrow key (↑) located next to the asterisk key on your keyboard is used to indicate exponentiation. If you want to raise a number to a power, use the PRINT command followed by a number, the up arrow, and the exponent indicating the power, in that order. For example, to find out what 3 squared is, type:

```
PRINT 3 ↑ 2 RETURN
```

TIP: You can use a question mark (?) instead of typing the word PRINT. The remaining examples in this section use the question mark symbol in place of the word PRINT. (Symbols and abbreviations for all BASIC commands can be found in the appendix of this manual.)

• ORDER OF OPERATIONS

You have seen how you can combine addition and subtraction in the same PRINT command. If you combine multiplication or division with other operations, you may not get the result you expect. For example, type:

```
? 4 + 6 / 2 RETURN
```

If you assumed you were dividing 10 by 2, you were probably surprised when the computer responded with 7. You got this result because multiplication and division are performed by the computer before addition or subtraction, no matter in what order you type the command. Multiplication and division are said to take precedence over addition and subtraction. Exponentiation takes precedence over all of the other four operations. If you type:

```
? 16 / 4 ↑ 2 RETURN
```

The computer responds with 1 because it squared the 4 before it divided 16 by it.

You can tell the computer which operations you want performed first by using parentheses () in the PRINT command. For instance, in the first exam-

ple above, if you want to tell the computer to add before dividing, you would type:

```
? (4 + 6)/2 RETURN
```

This gives you the correct answer, 5.

If you want the computer to divide before squaring in the second example, you would type:

```
? (16/4)↑2 RETURN
```

Now you have the correct answer, 16.

If you don't use parentheses, the computer will perform the calculations according to the basic precedence rules. When all operations in a calculation have equal precedence, they are performed from left to right. For example:

```
? 4*5/10*6 RETURN
```

The operations are performed in order from left to right, so the result is 12. If you want to divide $4*5$ by $10*6$ you would type:

```
? (4*5)/(10*6) RETURN
```

The answer is now 0.333333333.

• PRINTING TEXT

Now that you know how to print numbers and make calculations, it's time to learn how to print text. It's actually very simple—there are far fewer rules for printing text than making calculations. You just type the PRINT command followed by whatever you want put on the screen in quotation marks (" "). You can get the quotes by pressing the SHIFT key and the numeral "2" key. Try the following examples.

```
? "COMMODORE 64" RETURN
```

```
? "4*5" RETURN
```

Notice that when you press RETURN, what was in the quotes is displayed on the screen exactly as you typed it. Also note the difference between the second example and

```
? 4*5 RETURN
```

You can PRINT anything you want on the screen by using the PRINT command. You can combine text and calculations in a PRINT command.

```
? "4*5 = "4*5 RETURN
```

See how the computer PRINTS what you put in quotes, makes the calculation and PRINTS the result. It doesn't matter whether the text or calculation comes first. In fact, you can use both several times in one PRINT command.

```
? 4*(2 + 3) "is the same as "4*5 RETURN
```

Notice that even the spaces inside the quotation marks are printed on the screen. Type

```
? " OVER HERE" RETURN
```

There are special keys on your keyboard that you can use in PRINT commands to tell the computer where to put the information on the screen. They are the cursor control keys (CRSR) located below the RETURN key. The one on the right is the cursor right/left key, the other one is the cursor down/up key. Press these keys and note how the cursor moves on the screen. To move the cursor up, press SHIFT while pressing the down/up key. To move the cursor left, press SHIFT while pressing the right/left key.

When you type the cursor keys inside quotation marks, graphic characters are shown on the screen to represent the keys. These characters will NOT be printed on the screen when you press RETURN. Retype the example above, using the cursor down key instead of the spaces inside the quotes. It should look like this:

```
? "○○○○○○○○○○OVER HERE" RETURN
```

You can tell the computer to print anywhere on your screen by using the cursor control keys inside quotation marks.

• DIRECT MODE FUNCTIONS

There are two BASIC functions that tell the computer where to print information on the screen. If you've used a typewriter, you are familiar with the TAB function. It tells the computer to print your information starting in the specified column. For example, if you want to print something starting in the 10th column and something else in the 20th column, you would use the TAB function like this:

```
? TAB(9)"HERE"TAB(19)"HERE" RETURN
```

The computer moves the cursor nine spaces to the right and then begins to print the first "HERE" in the 10th column. If you want to print two pieces of information with exactly 10 spaces between the end of the first and the beginning of the second, how could you do it? If you knew how many letters were in the first item, you could figure out how many spaces to TAB from the

left side to PRINT the second item, but that isn't necessary. There's another function that makes this task much easier, the SPC (space) function. Now the PRINT command would look like this:

```
? "HERE"SPC(10)"HERE" RETURN
```

The second piece of information is printed on the screen 10 spaces to the right of the end of the first piece, no matter how long the first piece is.

• MATHEMATICAL FUNCTIONS

Here are three other important functions that can be very helpful if you want to do more advanced mathematical procedures like rounding off numbers and finding square roots.

The first function is the square root function: SQR. If you want to find the square root of 50, just type:

```
? SQR(50) RETURN
```

You can find the square root of any positive number in this way.

The second function is rounding off a number to the nearest integer. First, use the INT (integer) function. The INT function takes away everything after the decimal point. Type:

```
? INT(SQR(50)) RETURN
```

```
? INT(4.25) RETURN
```

```
? INT(4.75) RETURN
```

If you want to round off to the nearest whole number, then the last example should return a value of 5. In fact, you want to round off to the next highest number any number with a decimal above 0.5. To do this, you have to add 0.5 to the number when using the INT function, so numbers with decimals above 0.5 will be increased by 1 before being rounded down by the INT function. Try this example:

```
? INT(4.75 + 0.5) RETURN
```

In this case the computer adds 0.5 to 4.75 before it executes the INT function, so that 5.25 is rounded down to 5 for the result. If you want to round off the result of a division calculation, you do this:

```
? INT((X/Y) + 0.5) RETURN
```

In this case you can substitute any values for the characters X and Y in the inner parentheses.

What if you want to round off numbers to the nearest 0.01—such as dollars and cents? First, instead of adding 0.5 to your number, add 0.005 and then multiply by 100. Let's say you want to round 2.876 to the nearest 0.01. Using this method, you start with:

```
? (2.876 + 0.005)*100 RETURN
```

Now using the INT function to get rid of everything after the decimal point (which moves two places to the right when you multiply by 100) so you are left with:

```
? INT(((2.876 + 0.005)*100) RETURN
```

which gives you 288. All that's left to do is divide by 100 to give you 2.88—which is what you want! You can round off calculations to the nearest 0.01 like this one:

```
?INT(((2.876 + 1.29 + 16.1-9.534) + .005)*100)/100 RETURN
```

There is one more function which may come in handy when dealing with negative numbers. It is the absolute value function: ABS. Using this function gives you the positive value of the number regardless of whether it is a positive or negative number.

```
? ABS(-10) RETURN
```

```
? ABS(5);"IS EQUAL TO" ;ABS (-5) RETURN
```

• PROGRAM MODE

Now that you can communicate with your Commodore 64 in DIRECT mode by typing BASIC commands, it's time to begin the next step: Writing a program.

A program is just a series of BASIC commands that tell the computer what to do. The commands are numbered so the computer will know in what order you want them executed. In a BASIC program, the commands are called statements or lines. Many of the commands you learned in DIRECT mode can be easily made into program statements. For example, type:

```
10?"COMMODORE 64" RETURN
```

The first thing you should notice is that the computer did not print COMMODORE 64 when you hit RETURN. That is because the 10 before the PRINT command tells the computer that you are writing a BASIC program that begins at line number 10. Now type RUN and press RETURN.

Congratulations! You have just written and RUN your first BASIC program.

The line numbers in a program serve another purpose besides ordering your commands for the computer. They serve as a reference for the computer in case you want to execute the command on that line later in your program. We use the GOTO command to tell the computer to go to a line and execute the command(s) in it. Now type:

```
20 GOTO 10
```

When you press RETURN after typing that line, you add it to your program in the computer's memory. It is common practice to number your program lines in increments of 10 in case you want to go back and add lines later on. Type RUN **RETURN** and watch the PRINT message scroll down your screen. When you have finished watching, press the RUN/STOP key on the left side of your keyboard to stop the program. This is a simple two-line program that repeats itself endlessly because the second line keeps referring the computer back to the first line. It would go on forever unless you stopped it with the STOP key. Now type LIST **RETURN**. The screen should say:

```
10 PRINT"COMMODORE 64"  
20 GOTO 10  
READY
```

Your program is still in memory. You can RUN it again if you want to. This is an important difference between PROGRAM mode and DIRECT mode. Once a command is executed in DIRECT mode, it is no longer in the computer's memory.

Notice that if you used the ? symbol in the PRINT statement, the computer has converted it into the full command. This happens when you LIST any command you have abbreviated in a program.

You can use any whole number from 0 to 63,999 for a line number. Don't be afraid to experiment with your computer, writing programs or just making calculations in DIRECT mode. Nothing you type can damage the computer permanently. Anytime you want to start again or erase a BASIC program in the computer's memory, just type NEW **RETURN**. This command clears out the computer's BASIC memory, the area where programs are stored.

• CONSTANTS, VARIABLES AND STRINGS

CONSTANTS

There is a part of the computer's BASIC memory reserved for the numbers and letters you use in your program. Think of it as a number of boxes in

the computer that store information about your program: Type in this short program:

```
10 X = 5  
20 ?X
```

Now RUN the program and see how the computer prints a 5 on your screen. You told the computer in line 10 that the letter X will represent the number 5 for the remainder of the program. We call this an assignment statement because now there is a box labeled X in the computer's memory, and the number 5 has been assigned to it. The = sign tells the computer that whatever comes to the right of it will be assigned to a box (a memory location) labeled with the letter(s) to the left of it. The box name on the left of the = sign can be either one or two letters, or one letter and one number (the letter MUST come first). The names can be longer but the computer only looks at the first two characters so the names PA and PART would refer to the same box. Also, you cannot use BASIC commands (LOAD, RUN, LIST, etc.) or keywords (INT, ABS, SQR, etc.) as names. Refer to the BASIC encyclopedia in the back of this manual if you have any doubt about what is and what is not a BASIC keyword.

In this case, X is called a constant because it always represents the number 5. You can put calculations to the right of the = sign to assign the result to a box. You can mix text with constants in a print statement to identify them. Type NEW **RETURN**, then try this program:

```
10 A = 3 * 100  
20 B = 3 * 200  
30 ?"A IS EQUAL TO "A  
40 ?"B IS EQUAL TO "B
```

Now there are two boxes labeled A and B in the computer's memory, containing the numbers 300 and 600 respectively. If, later in the program, you want to change the value of a constant, just put another assignment statement in the program. Add these lines to the program above and RUN it again.

```
50 A = SQR(121)  
60 B = ABS(-15)  
70 GOTO 30
```

Once again, you'll have to press the STOP key to break from the program. LIST the program and trace the steps taken by the computer. First, it assigns the value to the right of the = sign in line 10 to the letter A. It does the same thing in line 20 for the letter B. Next, it prints the messages in lines 30 and 40 that give you the values of A and B. Finally, it assigns new values to A and B

in lines 50 and 60. The old values are destroyed and cannot be returned unless the computer executes lines 10 and 20 again (which it does not in this program). When the computer is sent to line 30 to begin printing the values of A and B again, it prints the new values. Lines 50 and 60 reassign the same values to A and B (this does no harm) and line 70 sends the computer back to line 30. This is called an endless loop. It is not good programming practice. Other methods of looping are discussed later in this and the following two chapters.

VARIABLES

A variable is a value assigned to a box that changes during the course of the program. Sometimes the value of a variable is not known at the beginning of the program (i.e., its value will be the result of a calculation to be made in the program). Clear your computer's memory by typing NEW **RETURN** and type in the program below:

```
10 A = INT(100/9)
20 B = (3*4)12
30 C = A*B
40 ?A "TIMES "B" EQUALS "C
```

Note that A and B are constants—they are expressed only in numeric terms in the program. C, on the other hand, is expressed in terms of A and B, making it a variable. The value in a variable box in memory often changes during a program.

STRINGS

A string is a character or group of characters enclosed in quotes. These characters are stored in the computer's memory in much the same way numeric variables are. You can also use names to represent strings as you use them to represent numbers. Use the \$ after the string name to tell the computer it is a string variable and not a numeric variable. Clear your computer's memory and type in the program below:

```
10 A$ = "COMMODORE"
20 X = (200/25)12
30 B$ = "COMPUTER"
40 Y = INT(4*0.35)
50 ?"THE "A$;X;B$" IS NUMBER "Y
```

See how you can print numeric and string variables in the same statement? Try experimenting with variables in your own short programs. Notice

that you can print the value of a variable from a program in DIRECT mode once the program has been RUN. Type ?A\$;X;B;\$ after running the program above and see that those three boxes are still in the computer's memory. If you want to clear this area of BASIC memory but still leave your program intact, use the CLR (clear) command. Just type CLR **RETURN** and all constants, variables and strings are wiped out. But when you type LIST, you can see that the program is still in memory. The NEW command discussed earlier erases both the program *and* the variables in it.

• IF-THEN

Now that you can change the values of variables easily, the next step is to have the computer make decisions based on these updated values. We do this with the IF-THEN statement. We tell the computer to execute a command only IF a condition is true (i.e., IF X = 5). The command we want the computer to execute when the condition is true comes after the THEN part in the statement. Clear your computer's memory and type in this program.

```
10 J = 0
20 ?J, "COMMODORE 64"
30 J = J + 1
40 IF J = 5 THEN GOTO 60
50 GOTO 20
60 END
```

At last, we no longer have to press the STOP key to break out of a looping program. The IF-THEN statement tells the computer to keep printing "COMMODORE 64" and incrementing J until (J = 5) is true. When the IF condition is false, the computer just jumps to the next line of the program no matter what comes after the THEN. Notice the END command in line 60. It is good practice to put an END statement at the end of your programs. It tells the computer where to stop executing statements.

Following is a list of symbols that may be used in the IF condition, and their meanings:

SYMBOL	MEANING
=	EQUALS
>	GREATER THAN
<	LESS THAN
<>	NOT EQUAL TO
>=	GREATER THAN OR EQUAL TO
<=	LESS THAN OR EQUAL TO

• EDITING TIPS

At this point, you know enough to begin programming in BASIC. You have learned how to make calculations in both DIRECT and PROGRAM modes. You have seen how to print text or characters on the screen. You also know how to tell the computer to repeat commands and make decisions. There are more advanced ways of telling the computer to do these things (which you'll see later in this Guide), but you have all you need to get started. The following paragraphs provide some tips on typing in your programs and making corrections and additions to them.

To begin with, type in the program from the IF-THEN example just discussed. Now type 10 `RETURN`. You just erased line 10 from the program. LIST your program and see for yourself. If the old line 10 is still on the screen, move the cursor up so that it is blinking anywhere on that line. Now, if you press RETURN, line 10 is back in the computer's memory. Hold down the SHIFT and press the CLR/HOME key on the upper right of your keyboard to clear your screen. Now LIST your program and notice that line 10 is back again. Move the cursor up again so it is blinking on the 0 in 10. Now type a 5 and press RETURN. You have just duplicated line 10 with a new line at 15. Clear your screen and list the program. You can cursor up and make any changes to any line of the program you want to. Just remember to press RETURN after you make changes, or the computer will not recognize them. Also, you can retype a whole line and press RETURN—the old version of the statement will be erased when you press RETURN.

Now clear your memory and type:

```
10 ?"I ADORE MY 64"
```

Let's say you want to add a word in the middle of that string (for example, the word COMMODORE). Just move the cursor so it is blinking on the letter that is one space *after* the spot where you want to insert the word. In this case, the cursor should be blinking on the 6. Hold down the SHIFT while pressing the INST/DEL key in the upper right corner of your keyboard and watch the 64" move to the right until you release the INST/DEL key. Then type COMMODORE and move the cursor to the 6. In you held down the INSERT key for a long time, then you have some blank spaces to get rid of. Hold down the DEL key until the 64" is in the correct position. Now press RETURN and LIST your program to see if your change is registered in memory.

• STORING AND REUSING YOUR PROGRAM

Once you have edited your program, you may want to store it permanently so that you will be able to recall and use it later. To do this you'll need either the Commodore Datassette recorder or a Commodore disk drive.

• FORMATTING A DISK

To store programs on a new (or blank) disk, you must first prepare the disk to receive data. This is called "formatting" the disk. Make sure that you turn on the disk drive before inserting any disk.

To format a blank disk, type this command:

```
OPEN 15,8,15: PRINT# 15, "N:A$,B$" RETURN
```

In place of A\$, type a disk name of your choice; you can use up to 16 characters to identify the disk. In place of B\$, type a two-character code of your choice (such as W2).

The cursor disappears for a second or so. When the cursor blinks again, seal the disk with the following command:

```
CLOSE 15 RETURN
```

The entire formatting process takes about 80 seconds.

• THE SAVE COMMAND

You use the SAVE command to store your program on cassette tape or disk.

SAVEing ON CASSETTE TAPE

If you are using a Datassette to store your program, insert a blank tape in the recorder, rewind the tape (if necessary) and type:

```
SAVE "PROGRAM NAME" RETURN
```

The PROGRAM NAME can be anything you want it to be. You can use letters, numbers and/or symbols—up to 16 characters in all. Note that you must enclose the PROGRAM NAME in quotation marks. The screen on your computer goes blank while the program is being SAVED, but it returns to normal when the process is completed.

SAVEing ON A DISK

If you have a Commodore single disk drive, you can store your program on disk by typing:

```
SAVE "PROGRAM NAME", 8 RETURN
```

The 8 indicates to the computer that you are using a disk drive to store your program.

The same rules apply for the PROGRAM NAME whether you are using tape or disk. Note that you cannot put two programs with the same name onto the same disk.

• THE LOAD COMMAND

Once a program has been SAVEd, you can LOAD it back into the computer's memory and RUN it anytime you wish.

LOADing FROM CASSETTE TAPE

To LOAD your program from cassette tape, type:

```
LOAD "PROGRAM NAME" RETURN
```

If you do not know the name of the program, you can type:

```
LOAD RETURN
```

and the next program on the tape will be retrieved.

You can use the counter on the Datassette to identify the starting position of programs. Then, when you want to retrieve a program, simply wind the tape forward from 000 to the programs's start location, and type:

```
LOAD RETURN
```

In this case you don't have to specify the PROGRAM NAME; your program will load automatically because it is the next program on the tape.

NOTE: During the LOAD process, programs are not erased from the tape; they are simply copied into the computer. However, LOADing a program automatically erases any BASIC program that may have been in the computer's memory.

LOADing FROM A DISK

To load your program from a disk, type:

```
LOAD "PROGRAM NAME", 8 RETURN
```

Again, the 8 indicates to the computer that you are working with a disk drive.

• DISPLAYING YOUR PROGRAM

To see your program once it has been loaded from either tape or disk, type:

```
LIST RETURN
```

Your entire program will then be displayed.

• OTHER DISK-RELATED COMMANDS

REPLACING A PROGRAM

To replace a program with a corrected copy, type:

```
SAVE"@0:PROGRAM NAME", 8 RETURN
```

VERIFYING A PROGRAM

To verify that a program has been correctly saved or loaded, type:

```
VERIFY"PROGRAM NAME", 8 RETURN
```

If the program in the computer is identical to the one on the disk, the screen display will respond with the letters "OK".

DISPLAYING YOUR DISK DIRECTORY

To see a list of the programs on your disk, type:

```
LOAD"$", 8 RETURN
```

The cursor disappears during this process. When the cursor reappears, type:

```
LIST RETURN
```

A list of the programs on your disk will then be displayed.

DELETING A PROGRAM

To delete a program from the disk (also known as "scratching" a program), type:

```
OPEN 1,8,15, "S0:PROGRAM NAME" RETURN  
CLOSE 1 RETURN
```

INITIALIZING A DISK DRIVE

If the disk drive's red light is blinking, it indicates a disk error. You can restore the disk drive to the condition it was in before the error occurred by using a procedure called "initializing." To initialize a drive, type:

```
OPEN 1,8,15, "I":CLOSE 1 RETURN
```

If the red light is still blinking, remove the disk and turn the drive off, then on.

For further information on SAVEing and LOADing your programs, refer to your Datassette or disk drive manual.

• SAMPLE PROGRAM

Here is a sample program incorporating many of the techniques and commands discussed in this section.

This program calculates the average of three variables (X, Y and Z) and prints their values and their average on the screen. You can edit the program and change the calculations in lines 10 - 30 to change the values of the variables. Line 40 adds the variables and divides by 3 to get the average. Note the use of parentheses in line 40 to tell the computer to add the numbers before it divides them.

TIP: Whenever you are using more than one set of parentheses in a statement, it's a good idea to count the number of left parentheses and right parentheses to make sure they are matched.

```
10X = INT (SQR(45))  
20 Y = INT ((25/(7.5))↑2)  
30 Z = ABS(-16-9)  
40 A = (X + Y + Z)/3  
50 ?"THE AVERAGE OF";X;Y;"AND";Z"IS";A  
60 END
```

You now know something about the BASIC language and some elementary programming concepts. The next section builds on these concepts, introducing new commands and functions that let you interact directly with the computer.



PROGRAMMING IN BASIC

This section describes additional BASIC commands and more sophisticated programming techniques

FOR-NEXT _____	53
INPUT _____	54
GET _____	55
STOP and CONT _____	56
Sample Program _____	56

This section gives you additional information on BASIC that will allow you to create and run more sophisticated BASIC programs. New commands and statements like FOR-NEXT, INPUT, GET and STOP are introduced and used in sample programs. The structure and operation of these sample programs are analyzed and illustrated.

• FOR-NEXT

Remember the sample program in the IF-THEN example in Section 4? We got the computer to print COMMODORE 64 five times by telling it to increment the variable J by units of one until it equalled five, then ended the program. There is a simpler way to do this in BASIC. We use a FOR-NEXT loop in the following way:

```
10 FORJ = 1 TO 5
20 ?"COMMODORE 64"
30 NEXT J
40 END
```

RUN this program and compare the result with the result of the IF-THEN program—they are the same! In fact, the steps taken by the computer are almost identical for the two programs. The FOR-NEXT loop is a very powerful set of commands. You can tell the computer to do many things without having to type them all in your program. Let's trace the computer's steps for the program above.

First, the computer assigns a value of 1 to the variable J. The 5 in the FOR statement tells the computer to execute all statements between the FOR statement and the NEXT statement, in this case just the PRINT statement, until J is equal to 5. After the computer assigns a value of 1 to J, it compares 1 to 5 to see if J = 5 is true—much in the same way as the IF statement. Since J = 5 is not true yet, the computer continues with the program. It executes the PRINT statement. The NEXT J statement tells it to go back to the FOR statement, add 1 to J, compare J to 5 and continue if J = 5 is still false. After five executions of this loop, J will equal 5. At this point, the computer drops down to the statement that comes immediately after the NEXT statement and continues from there.

You can tell the computer to increment your counter by units of 10, 0.5 or any other number. You do this with the STEP command in the FOR statement. For example, if you want the computer to count by tens to 100, type:

```
10 FORX = 0 TO 100STEP 10
20 ?X
30 NEXT
```


Notice that you do not need the X in the NEXT statement if you are only executing one loop at a time—this is discussed in the chapter on advanced concepts. Also, you do not have to increment your counter—you can decrement it.

Edit line 10 in the program to read:

```
10 FOR X = 100 TO 0 STEP -10
```

The computer will count backwards in units of 10 from 100 to 0. If you don't use the STEP command, the computer will increment the counter by units of 1. An important thing to remember when you use the FOR-NEXT loop is that you can use a variable in place of any of the numbers in the FOR statement. As we introduce more of the BASIC commands, you will see what a powerful device this can be. Clear the computer's memory and RUN this program.

```
10 K = 10
20 FOR I = 1 TO K
30 ?"COMMODORE 64"
40 NEXT
```

• INPUT

You can change the value of K in line 10 to make the computer execute the loop as many times as you want it to. What if you wanted to be able to tell the computer how many times to execute the loop at the time the program is RUN?

You want to be able to change the value of K each time you run the program without having to change the program itself. We call this the ability to interact with the computer. You can have the computer ask you how many times you want it to execute the loop. To do this, use the INPUT command. Replace line 10 in the program above with:

```
10 INPUT K
```

Now when you RUN the program, the computer responds with a ? to let you know it is waiting for you to tell it what you want the value of K to be. Type 15 and press RETURN. The computer will then execute the loop 15 times. You can make the computer print a message in an INPUT statement to tell you what variable it's waiting for. Replace line 10 with:

```
10 INPUT"PLEASE ENTER A VALUE FOR K";K
```

Remember to enclose your message to be printed (called a prompt) in quotes. Also, you must use a semicolon between the prompt and the K. You

can put anything you want in your prompt, but it cannot be longer than 38 characters.

The INPUT statement can also be used with string variables. The same rules that apply for numeric variables apply for strings—don't forget to use the \$ to identify all your string variables. Clear your computer's memory and type in this program.

```
10 INPUT"WHAT IS YOUR NAME";N$
20 ?"HELLO "N$
```

Don't forget to press RETURN after you type in your name. Once the value of a variable (numeric or string) has been INPUT in a program, you can refer to it by its variable name any time in the program. Type ?N\$ RETURN—your computer remembers your name!

• GET

There is another BASIC command you can use in your program to interact with the computer. It is the GET command and is similar to INPUT. To see how the GET statement works, clear the computer's memory and type in this program.

```
10 GET A$
20 IF A$ = "" THEN GOTO 10
30 ?A$
40 END
```

When you type RUN RETURN, the computer doesn't appear to do anything. It is waiting for you to press a key on the keyboard. The GET command tells the computer to fetch a character from the keyboard. The computer is perfectly satisfied with a null character—which is what it gets when no key is being pressed. This is the reason for line 20, which tells the computer that if it got a null character, (two double quotes with no space between them), it should go back to line 10 and get another character. This loop continues until you actually press a key. The computer then assigns the character on that key to A\$.

The GET command is very useful in BASIC because you can use it to program a key on your keyboard. The example below programs the Q key to print a message on the screen. Once you type it in and RUN it, press Q and see what happens.

```
10 ?"PRESS Q TO VIEW MESSAGE"
20 GET A$
30 IF A$ = "Q" THEN GOTO 20
```



```

40 IF A$ = "Q" THEN GOTO 60
50 GOTO 20
60 FOR I = 1 TO 25
70 ?"NOW I CAN USE THE GET STATEMENT"
80 NEXT
90 END

```

Notice that if you try to press any key other than the Q, the computer will go back to line 20 to GET another character.

• STOP AND CONT

There is another way to interact with your computer. You can make it stop running the program, then continue executing it when you are ready. The STOP command must be in the program—you can put a STOP statement anywhere you want to in a program. When the computer breaks from the program, you can use direct mode commands to find out exactly what is going on in the program (i.e., the value of a loop counter or other variable). This is a powerful device when you are debugging (fixing) your program. Clear the computer's memory and type in the program below.

```

10 X = INT(SQR(630))
20 Y = (.025*80)↑2
30 Z = INT(X*Y)
40 STOP
50 FOR J = 0 TO Z STEP Y
60 ?"INTERACTION CAN BE FUN"
70 NEXT
80 END

```

Now RUN the program—the computer responds with "BREAK IN 40". At this point, the computer has calculated the values of X, Y and Z. If you want to be able to figure out what the rest of the program is supposed to do, tell the computer to PRINT X;Y;Z. **RETURN** Often, when you are fixing a large program (or a complex small one), you'll want to know the value of a variable at a certain point in the program. When you have all the information you need, type CONT **RETURN**. The computer continues with the program, starting with the statement after the STOP command.

• SAMPLE PROGRAM

Remember the program from the previous section that calculated the average of three numbers? Now that you know how to use the FOR-NEXT

loop and the INPUT command, you can make the program more powerful. Clear the computer's memory and type in the program below. By now, you should recognize the graphic character in line 40 as the CRSR down key inside quotes. Don't worry that line 90 does not fit onto one line of your screen. Remember, a program line can be up to two screen lines (80 characters) long.

```

10 T = 0
20 INPUT"HOW MANY NUMBERS";N
30 FOR J = 1 TO N
40 INPUT"PLEASE ENTER A NUMBER";X
50 T = T + X
60 NEXT
70 A = T/N
80 PRINT
90 ?"YOU HAVE";N" NUMBERS TOTALING";T
100 ?"AVERAGE = ";A
110 END

```

Here is a line-by-line explanation of what the program does.

Line 10 assigns an initial value of 0 to T, which will be the running total of the numbers.

Line 20 lets you determine how many numbers to average.

Line 30 tells the computer to execute a loop that many times.

Line 40 lets you type in the actual numbers to be averaged.

Line 50 adds each number to the running total.

Line 60 tells the computer to go back to line 30, increment the counter (J) and start the loop again.

Line 70 divides the total by the amount of numbers you typed in (N) after the loop has been executed N times to calculate the average.

Line 80 prints a blank line on the screen.

Line 90 prints the message that gives you the amount of numbers and their total.

Line 100 prints the average of the numbers.

Line 110 tells the computer that your program is finished.

Now you have the ability to tell the computer how many numbers you want to average, and you can change the numbers every time you run the program—without having to change the program.

You have extended your grasp of BASIC in this section. The next section shows you how to fine-tune the commands, concepts and techniques you've learned so far.

6

ADVANCED CONCEPTS

This section defines and shows how to use even more powerful BASIC commands, statements, functions and programming techniques

READ-DATA _____	61
RESTORE _____	62
Using Loops _____	64
Using the Colon _____	65
Dimensioning Arrays _____	65
GOSUB-RETURN _____	67
ON GOTO/GOSUB _____	68
RAM/ROM Access _____	69
ASC and CHR\$ _____	70
Function Keys _____	70
Converting Strings and Numbers _____	71
Random Numbers _____	72

This section introduces still more BASIC commands, statements, functions and programming techniques. New commands and functions include READ-DATA, RESTORE, DIMENSION, GOSUB-RETURN, ON GOTO/GOSUB, PEEK, POKE, ASC, CHR\$ and STR\$.

These commands and functions allow you to create repeated actions through techniques called looping and nesting; handle tables of values; branch or jump to another section of a program and also return from that section; assign varying values to a quantity—and more. The use of special function keys (the four large keys to the right of the main keyboard) is also explained. Again, sample programs are developed and analyzed to illustrate just how these BASIC concepts work and interact.

• READ-DATA

There is another way to tell the computer what numbers or characters to use in your program. You can use the READ statement in your program to tell the computer to get a number or character(s) from the DATA statement. For example, if you want the computer to find the average of five numbers, you can use the READ and DATA statements this way:

```
10 T = 0
20 FOR J = 1 TO 5
30 READ X
40 T = T + X
50 NEXT
60 A = T/5
70 ?"AVERAGE = ";A
80 END
90 DATA 5,12,1,34,18
```

When you run the program, the computer will print AVERAGE = 14 on your screen. Let's trace the steps taken by the computer to arrive at this number:

Line 10 assigns a value of zero to T—the running total.

Line 20 tells the computer to execute a loop five times.

Line 30 tells the computer to go to the DATA statement (line 90), get a value and assign it to the variable X.

Line 40 adds the value of X to the running total.

Line 50 tells the computer to execute the loop again.

Line 60 divides the total by five when the loop is completed for the fifth time.

Line 70 prints the average on the screen.

Line 80 tells the computer that it is done.

Line 90 is where the computer goes to get the values for X—this is discussed further below.

You can put any number you want in a DATA statement, but you can't put calculations in a DATA statement. The DATA statement can be anywhere you want in the program—even after the END statement. This is because the computer never really executes the DATA statement, it just refers to it.

If you have more than one DATA statement in your program, the computer will refer to the one that is closest to the read statement being executed at the time. Be sure to separate your data items with commas. The computer uses a pointer to remind itself which piece of data it read last. After the computer reads the first number in the DATA statement, the pointer points to the second number. When the computer comes to the READ statement again, it assigns the value the pointer indicates to the variable name in the READ statement. You can use as many READ and DATA statements as you need in a program, but make sure there is enough data in the DATA statements for the computer to read. Remove one of the numbers from the DATA statement in the last program and run it again. The computer responds with ?OUT OF DATA ERROR IN 30. When the computer executes the loop for the fifth time, there is no data for it to read. That is what the error message is telling you.

• RESTORE

You can use the RESTORE command in a program to reset the data pointer to the first piece of data if you need to. Replace the END statement (line 80) in the program above with:

```
80 RESTORE
```

and add:

```
85 GOTO 10
```

Now the program can run continuously using the same DATA statement. Putting too much into the DATA statement doesn't bother the computer at all. If the computer gives you an OUT OF DATA ERROR message, it is because you forgot to replace the number you removed from the DATA statement.

As mentioned earlier, you can put characters in a DATA statement to assign values to string variables. The same rules apply as for numeric data. Clear the computer's memory and type in the following program:

```
10 FOR J = 1 TO 3
20 READ A$
30 ?A$
40 NEXT
50 END
60 DATA COMMODORE,64,COMPUTER
```

You can use numbers or characters in a DATA statement when you are reading strings. You cannot use characters in a DATA statement when you are reading numbers. When A\$ is a number, you can print it—but you cannot use it in mathematical calculations.

What if you want the computer to remember all the data in the DATA statement instead of replacing the value of a variable with the new data? What if you want to be able to recall the third number or the second string of characters?

Each time you assign a new value to a variable, (as the READ statement does) the computer erases the old value in the variable's box in memory and stores the new value in its place. You can tell the computer to reserve a row of boxes in memory and store every value that you assign to that variable in your program. This row of boxes is called an array.

If the array contains all of the values assigned to the variable X in the READ-DATA example, it would be called the X array. The first value assigned to X in the program is named X(1), the second value would be X(2), and so on. These are called subscripted variables. The numbers in the parentheses are called subscripts. You can use a variable or a calculation as a subscript. Below is an updated version of the average program using subscripted variables.

```
5 DIM X(5)
10 T = 0
20 FOR J = 1 TO 5
30 READ X(J)
40 T = T + X(J)
50 NEXT
60 A = T/5
70 ?"AVERAGE = ";A
80 END
90 DATA 5,12,1,34,18
```

Notice there are not many changes. Line 5 is the only new statement. It tells the computer to set aside five boxes in memory for the X array. Line 30 has been changed so that each time the computer executes the loop, it

assigns a value from the DATA statement to the position in the X array that corresponds to loop counter (J). Line 40 does the same thing it did before, but you must use a subscripted variable to do it.

After you run the program, if you want to recall the third number, type ?X(3) **RETURN**. The computer remembers every number in the array X. You can create string arrays to store the characters in string variables the same way. Try updating the COMMODORE 64 COMPUTER program so the computer will remember the elements in the A\$ array. It should look like this:

```
5 DIM A$(3)
10 FOR J = 1 TO 3
20 READ A$(J)
30 ?A$(J)
40 NEXT
50 END
60 DATA COMMODORE,64,COMPUTER
```

TIP: You do not need the DIM statement in your program unless the array you use has more than 10 elements. See DIMENSIONING ARRAYS.

• USING LOOPS

Before you proceed any further, you'll need to understand more about loops and the ways they are used to get the computer to do what you want. You can use a loop to slow down the computer—by now you have witnessed the amazing speed with which the computer executes commands. See if you can predict what this program will do before you run it.

```
10 A$ = "COMMODORE 64"
20 FOR J = 1 TO 10
30 PRINT
40 FOR K = 1 TO 1500
50 NEXT K
60 PRINT A$
70 NEXT J
80 END
```

Did you get what you expected? The loop contained in lines 40–50 tells the computer to count to 1500 before executing the remainder of the program. This is known as a delay loop and may be useful to you in your programs. Because it is inside the main loop of the program, it is called a nested loop. We will come back to nested loops later in this chapter.

• USING THE COLON

Another useful tool in programming is the colon (:). You can use the colon to separate two (or more) BASIC statements on the same line. This may save time and space in your programs, and it may save some memory. Update the above program by combining the statements in the delay loop on one line. This is how it should now appear:

```
10 A$ = "COMMODORE 64"
20 FOR J = 1 TO 20
30 PRINT
40 FOR K = 1 TO 1500:NEXT
50 PRINT A$
60 NEXT
70 END
```

Notice the program is one line shorter than before. Often the statements in a delay loop are combined on one line, because they work together and are executed in succession.

Statements after a colon on a line are executed in order, from left to right. You can put as many statements as you can fit onto two screen lines (80 characters) in one line of your program. This provides an excellent opportunity to take advantage of the THEN part of the IF-THEN statement. You can tell the computer to execute several commands when your IF condition is true. Clear the computer's memory and type in the following program:

```
10 INPUT "ENTER ANY NUMBER";N
20 IF N < 5 THEN PRINT "LESS THAN 5":GOTO 10
30 ? "GREATER THAN OR EQUAL TO 5"
40 END
```

See how you can tell the computer to execute more than one statement when (N < 5) is true. You can put any statement(s) you want after the THEN command, but none of them will be executed unless the IF condition is true.

• DIMENSIONING ARRAYS

Now that you know how to use a nested loop, you can have the computer handle data in a more advanced way. What if you had a large table of numbers on which you wanted to perform calculations? Picture a chart with 10 rows and 5 numbers in each row. Suppose you wanted to find the average of the five numbers in each row (see chart next page). You could create 10

arrays and have the computer calculate the average of the five numbers in each one. This is not necessary. You can put all of the numbers in a two-dimensional array. This array would have the same dimensions as the chart of numbers you want to work with—10 rows by 5 columns. The DIM statement for this array (we will call it array X) should be:

```
10 DIM X(10,5)
```

This tells the computer to reserve space in its memory for a two-dimensional array named X. The computer reserves enough space for 50 numbers. You do not have to fill an array with as many numbers as you DIMensioned it for, but the computer still reserves space for all of the positions in the array.

Now it becomes very easy to refer to any number in the chart by its column and row position. Refer to the chart of numbers below. Find the third element in the tenth row (1500). You would refer to this number as X(10,3) in your program. The program on the following page reads the numbers from the chart into a two-dimensional array (X) and calculates the average of the numbers in each row.

	Column				
Row	1	2	3	4	5
1	1	3	5	7	9
2	2	4	6	8	10
3	5	10	15	20	25
4	10	20	30	40	50
5	20	40	60	80	100
6	30	60	90	120	150
7	40	80	120	160	200
8	50	100	150	200	250
9	100	200	300	400	500
10	500	1000	1500	2000	2500

```
10 DIM X(10,5),A(10)
20 FOR R = 1 TO 10
30 T = 0
40 FOR C = 1 TO 5
50 READ X(R,C)
60 T = T + X(R,C)
70 NEXT C
80 A(R) = T/5
90 NEXT R
100 FOR R = 1 TO 10
110 PRINT "ROW #";R
120 FOR C = 1 TO 5
130 PRINT X(R,C);NEXT C
140 PRINT "AVERAGE = ";A(R)
150 FOR D = 1 TO 1000:NEXT
160 NEXT R
170 DATA 1,3,5,7,9
180 DATA 2,4,6,8,10
190 DATA 5,10,15,20,25
200 DATA 10,20,30,40,50
210 DATA 20,40,60,80,100
220 DATA 30,60,90,120,150
230 DATA 40,80,120,160,200
240 DATA 50,100,150,200,250
250 DATA 100,200,300,400,500
260 DATA 500,1000,1500,2000,2500
```

• GOSUB-RETURN

Until now, the only way you know to tell the computer to jump to another part of your program is the GOTO statement. What if, at a certain point in the program, you want the computer to jump to another part of the program, execute the statements in it, then return to the point it left off and continue executing the program from there? The part of the program that the computer jumps to and executes is called a subroutine. Clear your computer's memory and enter the program below.

```
10 A$ = "SUBROUTINE":B$ = "PROGRAM"
20 FOR J = 1 TO 5
30 INPUT "ENTER A NUMBER";X
40 GOSUB 100
50 PRINT B$:PRINT
```



```

60 NEXT
70 END
100 PRINT A$:PRINT
110 Z = X^2:PRINT Z
120 RETURN

```

This program takes the five numbers you type in, squares them, and prints the result. The other print messages tell you when the computer is executing the subroutine or the main program. Line 40 tells the computer to jump to line 100, execute it and the statements following it until it sees a RETURN statement. The RETURN statement tells the computer to go back in the program to the line immediately following the GOSUB statement and continue executing. The subroutine can be anywhere in the program—including after the END statement. Also, remember that the GOSUB and RETURN commands must always be used together in a program (like FOR and NEXT & IF and THEN), otherwise the computer will give you an error message.

• ON GOTO/GOSUB

There is an even more powerful way to make the computer jump to another section of your program (we call that branching). By using the ON statement, you can have the computer decide what part of the program to branch to based on a calculation or keyboard input. The ON statement is used with either the GOTO or GOSUB-RETURN commands, depending on what you need the program to do. A variable or calculation should be after the ON command. After the GOTO or GOSUB command, there should be a list of line numbers. Type in the program below to see how the ON command works.

```

10 ?" ENTER A NUMBER BETWEEN ONE AND FIVE"
20 INPUT X
30 ON X GOSUB 100,200,300,400,500
40 END
100 ?"YOUR NUMBER WAS 1":RETURN
200 ?"YOUR NUMBER WAS 2":RETURN
300 ?"YOUR NUMBER WAS 3":RETURN
400 ?"YOUR NUMBER WAS 4":RETURN
500 ?"YOUR NUMBER WAS 5":RETURN

```

When the value of X is 1, the computer branches to the first line number in the list (100). When X is 2, the computer branches to the second number in the list (200), and so on.

• RAM/ROM ACCESS—PEEK AND POKE

The Commodore 64's memory is composed of RAM (Random Access Memory) and ROM (Read Only Memory). In all, there are over 64,000 memory locations in the computer. Each area of the computer's memory has a special function. For instance, there is a very large area to store your programs and the variables associated with them. This is the part of memory that gets cleared when you use the NEW command.

Other areas are not as large, but they have very specialized functions. For instance, there is an area of memory that controls the music features of the computer. There are some memory locations that have special functions of their own. There are two BASIC commands you can use to access and manipulate the computer's memory.

This can be a powerful programming device because the contents of the computer's memory locations determine exactly what the computer should be doing at the time. The PEEK command can be used to make the computer tell you what value is being stored in a memory location (a memory location can store any value between 0 and 255). You can PEEK the value of any memory location (RAM or ROM) in direct or program mode. Type:

```

P = PEEK(650) RETURN
?P RETURN

```

The computer assigns the value in memory location 650 to the variable P when you press RETURN after the first line. Then it prints the value when you press RETURN after the PRINT command. Memory location 650 determines whether or not keys like the SPACEBAR and CRSR repeat when you hold them down. A 0 in location 650 tells the computer to repeat these keys when you hold them down. Hold down the SPACEBAR and watch the cursor move across the screen.

To change the value stored in a RAM location, use the POKE command. Type:

```

POKE 650,96 RETURN

```

The computer stores the value after the comma (96) in the memory location before the comma (650). A 96 in memory location 650 tells the computer not to repeat keys like the SPACEBAR and CRSR keys when you hold them down. Now hold down the SPACEBAR and watch the cursor—almost nothing happens! The cursor moves one position to the right, but it does not repeat. To return to its normal state, type:

```

POKE 650,0 RETURN

```


You cannot alter the value of all the memory locations in the computer—the locations in ROM (57344–65535) can be read, but not changed.

NOTE: As mentioned before, there are many memory locations (65,536) in the Commodore 64. Refer to the Commodore 64 Programmer's Reference Guide for a complete memory map of the computer.

• ASC AND CHR\$

Every character the Commodore 64 can print (including graphic characters) has a number assigned to it. This number is called a character string code (CHR\$) and there are 255 of them in the Commodore 64. There are two functions associated with this concept that should prove to be very useful. The first is the ASC function. Type:

```
?ASC("Q") RETURN
```

The computer responds with 81. 81 is the character string code for the Q key. Substitute any key for Q in the command above to find out the code number for any key.

The second function is the CHR\$ function. Type:

```
?CHR$(81) RETURN
```

The computer responds with Q, of course! CHR\$ is the opposite of ASC. They both refer to the table of character string codes in the computer's memory. See the appendix of this guide for a full listing of ASC and CHR\$ codes.

• FUNCTION KEYS

By this time, you have probably noticed there are four large keys on the far right side of your keyboard. These are your function keys. They are no different than any other key on your keyboard except they do not have a printed character assigned to them. They do, however, have CHR\$ codes. In fact, each of them has two CHR\$ codes—one for when you press the key, and one for when you press the key while holding down the SHIFT key. The CHR\$ codes for the F1–F8 keys are 133–140. However, they are not numbered in order. The list below shows the keys and their corresponding CHR\$ codes.

F1	CHR\$(133)
F2	CHR\$(137)
F3	CHR\$(134)
F4	CHR\$(138)

F5	CHR\$(135)
F6	CHR\$(139)
F7	CHR\$(136)
F8	CHR\$(140)

To get the even-numbered function keys, hold down the SHIFT key while pressing the key. For example, to get F2, hold down SHIFT and press F1.

You can use the function keys in your programs in many ways. To do this, you'll need to use the GET statement. Refer to Section 5 if you need a refresher course on GET. The program below prepares the F1 key to print a message on the screen.

```
10 ?"PRESS F1 TO CONTINUE"  
20 GET A$:IF A$ = "" THEN 20  
30 IF A$ <> CHR$(133) THEN 20  
40 ?"YOU HAVE PRESSED F1"
```

Lines 20 and 30 do most of the work in this program. Line 20 makes the computer wait until a key is pressed before executing any more of the program. Note that when the command immediately after THEN is a GOTO command, only the line number is necessary. Also note that a GOTO command can GOTO the same line it is on. Line 30 tells the computer to go back and wait for another key to be pressed unless the F1 key has been pressed.

• CONVERTING STRINGS AND NUMBERS

Sometimes you may have the need to perform calculations on numeric characters that are stored as string variables in your program. Other times, you may want to perform string operations on numbers. There are two BASIC functions you can use to convert your variables between numeric and string type. The VAL function returns a numeric value for a string variable. Type in the short program below.

```
10 A$ = "64"  
20 A = VAL(A$)  
30 ? "THE VALUE OF"; A$ " IS"; A
```

At this point you should know the program does not have to be ended with an END statement.

The STR\$ function converts numeric variables into string variables. Clear the computer's memory and type in this program.

```
10 A = 64  
20 A$ = STR$(A)  
30 ? A$ " IS THE VALUE OF"; A$
```


• RANDOM NUMBERS

There is one final function before you learn to apply the concepts presented in Sections 4, 5 and 6 to things like graphics and music. The RND function tells the computer to generate a random number. All generated numbers are nine digits in decimal form between 0.000000001 and 0.999999999. Type:

```
?RND(0) RETURN
```

That number is generated by a free-running time clock inside the computer! You can add some calculations to this function to make the computer generate random numbers between one and six—as if a die was being rolled. Type:

```
?INT(RND(0)*6) + 1 RETURN
```

Without the + 1 at the end of this command, the computer generates a random number between zero and five. To simulate a pair of dice rolling, type:

```
?(INT(RND(0)*6) + 1) + (INT(RND(0)*6) + 1)
```

This section and the preceding two sections have been designed to familiarize you with the BASIC computer programming language and its capabilities. In the next two sections you will enter a world of advanced programming in graphics and sound. Remember that more information on every command discussed in this Guide can be found in the Commodore 64 Programmer's Reference Guide.



GRAPHICS, COLOR AND SPRITES

This section introduces the Commodore 64's exceptional graphics, color and animation capabilities

Printing in Different Colors _____	75
Color Character String Codes (CHR\$) _____	76
Color Registers—Changing Screen, Border and Character Colors _____	77
Screen Memory _____	80
Color Memory _____	82
Animation _____	83
Sprite Graphics _____	87
—Sprite Concepts _____	87
—Designing a Sprite Image _____	88
—Converting Your Sprite Image Into Data _____	88
—Controlling Sprites _____	94
—Animating Your Sprites _____	97
—Tying Your Sprite Program Together _____	100
Graphics Modes _____	102

Your Commodore 64 gives you exceptional graphics capabilities. The Commodore 64 offers sixteen colors, five graphics modes and programmable animated objects called sprites. This section elaborates on the several powerful graphics features built into the Commodore 64 and how they are used.

• PRINTING IN DIFFERENT COLORS

The Commodore 64 is capable of displaying 16 different colors on the screen. When you first turn on your Commodore 64, the screen background color is dark blue and the letters in the foreground are light blue. You can change those colors easily. All you do is hold down the CTRL key and press a numbered key between zero and eight. Notice that the cursor changes color according to the numbered key you pressed. All the succeeding characters are displayed in the color you selected. Hold down the Commodore key **⌘** and press a numbered key between zero and eight, and eight additional colors are displayed on the screen. You can use this method of changing colors only in direct mode—that is, outside of a program.

To select color within a program, the same principle applies. For a program, you must include the color selection information within a PRINT statement. For example, type the following and leave spaces between each letter:

```
10 PRINT " S P E C T R U M " RETURN
```

Now type line 10 again but this time hold down the CTRL key and press the 1 key directly after the open quotation mark. Release the CTRL key and type the "S". Now hold down the CTRL again and press the 2 key. Release the CTRL key and type the "P". Next hold down the CTRL key again and press the 3 key. Continue this process until you have typed all the letters in the word SPECTRUM and selected a color between each letter. Type a closed quotation mark and press the RETURN key. Now type RUN and press the RETURN key. The C64 displays the word SPECTRUM with each letter in a different color. Now type LIST and press the RETURN key. Notice the graphic characters that appear in the PRINT statement in line 10. These characters tell the C64 what color you want for each printed letter. Note that these graphic characters do not appear when the Commodore 64 PRINTs the word SPECTRUM in different colors.

The color selection characters, known as control characters, in the PRINT statement in line 10 tell the Commodore 64 to change colors. The computer then prints the characters that follow in the new color until another color selection character is encountered. While characters enclosed in quotation marks are usually PRINTed exactly as they appear, control characters are only displayed within a program LISTing.

Table 7-1 lists the colors available on the C64. The table also shows the key used to specify a given color, and the resulting control character that appears in quotes in a PRINT statement.

Table 7-1. C64 Colors

KEYBOARD	COLOR	DISPLAY	KEYBOARD	COLOR	DISPLAY
CTRL 1	BLACK	■	⌘ 1	ORANGE	◻
CTRL 2	WHITE	E	⌘ 2	BROWN	◻
CTRL 3	RED	E	⌘ 3	LT. RED	⊗
CTRL 4	CYAN	◻	⌘ 4	GRAY 1	◻
CTRL 5	PURPLE	■	⌘ 5	GRAY 2	◻
CTRL 6	GREEN	I	⌘ 6	LT. GREEN	◻
CTRL 7	BLUE	—	⌘ 7	LT. BLUE	◻
CTRL 8	YELLOW	π	⌘ 8	GRAY 3	◻

• COLOR CHARACTER STRING CODES (CHR\$)

Each character on the Commodore 64 keyboard has a number associated with it. When you press a key, the computer scans the keyboard and understands exactly which character is typed. A character code value is entered into memory each time a key is pressed. These codes are referred to as character string codes. Appendix E lists all the character string codes the Commodore 64 understands.

Within a program, you can select colors using character string codes instead of holding down the CTRL key and pressing a numbered key. For instance, enter the following sample program:

```
10 PRINT CHR$(5) RETURN
20 PRINT "WHITE" RETURN
```

NOTE: In the remainder of this section, the RETURN symbol is shown only after DIRECT mode statements, not after program lines.

When you RUN this program, the character color changes from blue to white and the word "WHITE" is displayed. The other 15 colors also have a character string code assigned to them. The following is a list of all the colors available on the Commodore 64 and the corresponding character string codes:

Color	CHR\$ Code	Color	CHR\$ Code
White	CHR\$(5)	Dk. Gray	CHR\$(151)
Red	CHR\$(28)	Gray	CHR\$(152)
Green	CHR\$(30)	Lt. Green	CHR\$(153)
Blue	CHR\$(31)	Lt. Blue	CHR\$(154)
Orange	CHR\$(129)	Lt. Gray	CHR\$(155)
Black	CHR\$(144)	Purple	CHR\$(156)
Brown	CHR\$(149)	Yellow	CHR\$(158)
Lt. Red	CHR\$(150)	Cyan	CHR\$(159)

To select any of the Commodore 64 colors, PRINT the above character string codes according to the colors you want to display on the screen. The following program illustrates how to select colors within a program.

```
10 PRINTCHR$(5)
15 PRINT"WHITE"
20 PRINTCHR$(28)
25 PRINT"RED"
30 PRINTCHR$(30)
35 PRINT"GREEN"
40 PRINTCHR$(31)
45 PRINT"BLUE"
47 PRINTCHR$(129)
48 PRINT"ORANGE"
50 PRINTCHR$(144)
55 PRINT"BLACK"
60 PRINTCHR$(149)
65 PRINT"BROWN"
70 PRINTCHR$(150)
75 PRINT"LT. RED"
80 PRINTCHR$(151)
85 PRINT"DK. GRAY"
90 PRINTCHR$(152)
95 PRINT"GRAY"
100 PRINTCHR$(153)
110 PRINT"LT. GREEN"
120 PRINTCHR$(154)
130 PRINT"LT. BLUE"
140 PRINTCHR$(155)
150 PRINT"LT. GRAY"
200 PRINTCHR$(156)
210 PRINT"PURPLE"
220 PRINTCHR$(158)
230 PRINT"YELLOW"
240 PRINTCHR$(159)
250 PRINT"CYAN"
```


• COLOR REGISTERS—CHANGING SCREEN, BORDER AND CHARACTER COLORS

Your Commodore 64 computer has 64K of memory. This means the C64 holds 64 times 1024 (65536) bytes of information. Think of the internal structure of your computer as 65536 storage compartments piled one on top of the other. They are labeled starting from the bottom at location zero (0) and continue upward to location 65535 on top. You can also refer to each byte as a register, so your Commodore 64 has 65536 registers.

Each byte inside your computer is used for a specific purpose. For instance, you have 38911 bytes available to program in BASIC. Your Commodore 64 tells you this as soon as you turn on the computer and read the opening screen. You may ask, what are all the rest of the bytes used for? They control the computer's brain, known as the operating system. The operating system registers control all the features of your Commodore 64.

A portion of the operating system controls graphics and color. You can select different colors by changing the contents of the Commodore 64 color registers. There are three color registers which control the colors of the border, the background and the characters. When you first turn on your Commodore 64, the background color is dark blue and the character and border colors are light blue. You can change the background, border and character color registers with the BASIC POKE statement.

The POKE command modifies the contents of the specified location and places the newly specified value in that location. The format of the POKE command is:

POKE memory location, value

For example, type the following POKE command:

```
POKE 53280,0 RETURN
```

Did you notice what happened? The border color changed from light blue to black. Location 53280 is the border color register. Location 53281 is the background color register and location 646 is the character color register. Now change the background color from dark blue to black with the following command:

```
POKE 53281,0 RETURN
```

Now all you need to know is how to change the character color with a POKE command. You already learned the two other methods to change the character color in the last section, first with the CTRL key and second with

character string codes (CHR\$). The following POKE changes the character color from light blue to white:

```
POKE 646,1 RETURN
```

Note that the character color changes to white, but the characters already on the screen remain the same color as before. All the characters you type from now on are displayed in white unless you change the character color again.

You're probably wondering what the values that are POKEd into the color registers mean. These values are the color information codes for the 16 colors available on the Commodore 64. The following list contains all the Commodore 64 colors and the corresponding color codes:

0	Black	8	Orange
1	White	9	Brown
2	Red	10	Light Red
3	Cyan	11	Dark Gray
4	Purple	12	Gray
5	Green	13	Light Green
6	Blue	14	Light Blue
7	Yellow	15	Light Gray

Try the following program. It uses FOR...NEXT loops, which you learned in the last chapter.

```
5 PRINT "♥": REM Use shifted CLR/HOME key to produce heart symbol
   shown in parentheses
10 FOR I=0 TO 15
15 POKE 53280, I
16 FOR J=1 TO 500: NEXT
18 NEXT
19 POKE 53280, 0
20 FOR I=0 TO 15
25 POKE 53281, I
26 FOR J=1 TO 500: NEXT
28 NEXT
29 POKE 53281, 0
30 FOR I=0 TO 15
35 POKE 646, I
36 PRINT "COLOR"
37 FOR J=1 TO 500: NEXT
38 NEXT
39 POKE 646, 14
50 POKE 53280, 14: POKE 646, 14: POKE 53281, 6
```


This program changes the color code value of each of the color registers using a FOR . . . NEXT loop. Lines 10 through 18 POKE each color value from 0 (black) to 15 (light gray) into the border color register and displays each border color on the screen. Lines 20 through 28 POKE each color value into the background color register and display each background color on the screen. Lines 30 through 38 POKE each color value into the character color register and display each character color on the screen.

Lines 16, 26 and 37 are FOR . . . NEXT loops that slow down the program. They are empty FOR . . . NEXT loops that delay program execution so you can notice the color changes on the screen. Try the program without the delay loops and see how fast the Commodore 64 runs. Line 40 restores the original border, screen and character color registers.

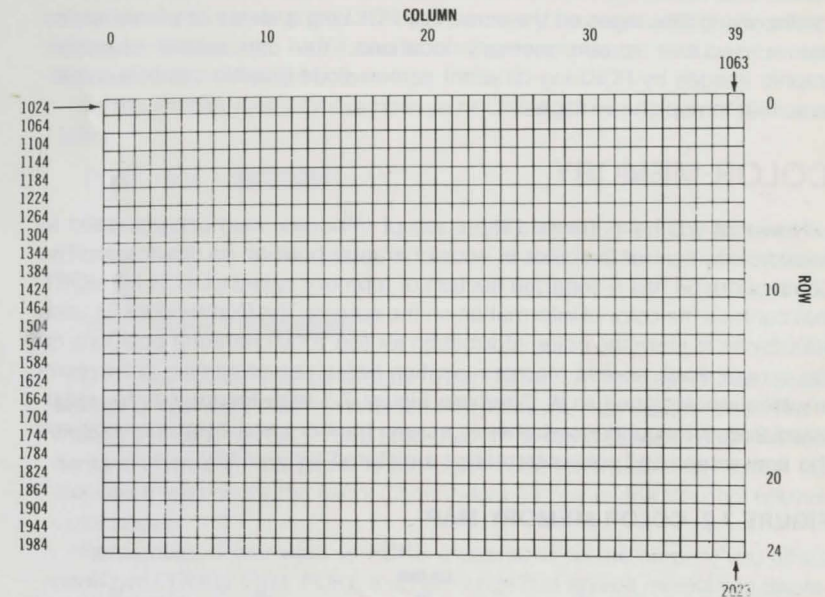
• SCREEN MEMORY

Since graphics is one of the Commodore 64's strongest features, the screen is an important part of the computer. The Commodore 64 screen has 1000 character positions—40 columns by 25 rows. Each character position uses one byte of memory, so the Commodore 64 needs 1000 bytes to store the information you see on the screen.

In the Color Register section, we referred to the memory of the Commodore 64 as 65536 storage compartments piled one on top of the other. Screen memory uses part of those storage compartments starting at location 1024 and ending at location 2023. The screen appears as a grid having 40 X (horizontal) positions and 25 Y (vertical) positions. In memory, however the character positions are actually stored sequentially.

The top left character position on the screen, referred to as the HOME position, is stored at location 1024. The character position directly to the right of HOME is stored at location 1025 and so on. The character position at the top right corner of the screen is stored at location 1063, 40 locations past the beginning of screen memory. The last character position, located at the bottom right corner of the screen, is stored at location 2023, the end of screen memory. Examine Figure 7-1 to understand the correspondence between the way the screen looks and the way information is sequentially stored in memory.

FIGURE 7-1. SCREEN MEMORY MAP



Remember when you learned about character string codes in the Color Character String Code section? The Commodore 64 has a separate set of codes used only by screen memory to display characters on the screen. Instead of outputting characters to the screen in PRINT statements, you POKE a screen code value directly into a specific screen memory location. For example, enter the following line:

```
POKE 1024,1 RETURN
```

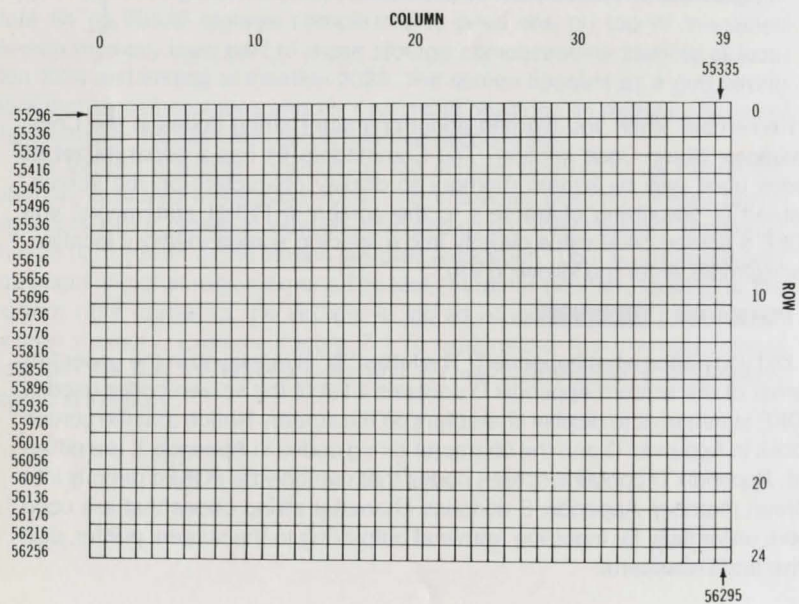
Did you notice what happened? The letter "A" is displayed in the upper left corner of the screen. Appendix D contains a list of the screen codes used in POKE statements to display characters on the screen. Notice that the screen codes in Appendix D and the character string codes in Appendix E are different. Appendix D contains screen codes that can only be POKEd directly into screen memory. Appendix E contains character string codes that are used more universally for inputting from and outputting to the screen, printer, disk drive and Datassette.

You can POKE any of the values in Appendix D into any of the screen locations between 1024 and 2023. Experiment with different characters and try displaying messages on the screen by POKEing a series of screen codes into consecutive screen memory locations. You can create character graphic images by POKEing different screen code graphic symbols in patterns that form picture images.

• COLOR MEMORY

Now that you have grasped the concept of screen memory, you need to know how to control the color of each character position on the screen. The Commodore 64 has a separate section of memory called COLOR MEMORY, that controls the color of information on the screen. The Commodore 64 uses 1000 bytes to store the color information for the 1000 character positions on the screen. Each screen memory location has a corresponding color memory location assigned to it. Compare Figure 7-1 with Figure 7-2, to understand the correspondence between screen memory and color memory and the way screen and color information are stored sequentially.

FIGURE 7-2. COLOR MEMORY MAP



Location 1024 in screen memory corresponds to location 55296 in color memory. Location 1063 corresponds to location 55335. Screen memory location 2023 corresponds to color memory location 56295. Remember, each screen location has a one to one correspondence to a color memory location that controls its color.

In the screen memory example you POKEd a 1 into location 1024 as follows:

```
POKE 1024,1 RETURN
```

This places the letter "A" in the HOME position on the screen. Now change the color of the letter "A" in the HOME position with the following POKE:

```
POKE 55296,1 RETURN
```

Did you notice the difference? The letter "A" in the HOME position changed from light blue to white. At this point you may wonder what the "1" means in POKE 55296,1. This time the "1" is not a screen code that represents a character. Instead it is the color code for white. Refer to the Color Registers section for the list of Commodore 64 colors and the corresponding color codes.

Remember, if you want to POKE a character to the screen, you actually need two POKES. First, POKE a screen code into screen memory to display a character. Second, POKE a color code into color memory to display the color of the character.

• ANIMATION

Your Commodore 64 is capable of animating objects on the screen. The idea behind computer animation is to display an image on the screen and simulate its motion through computer instructions.

Remember when you POKEd a character into screen memory and it was displayed on the screen? That's what you are going to do to animate a graphic character. To animate a graphic character on the screen, POKE its screen code into a screen memory location. Next, POKE the screen code for a blank (32) into the same screen location. Then POKE the graphic character screen code into a screen location next to the original one. Repeat the process with a series of adjacent screen memory locations. Since the computer is displaying and blanking out the graphic character in successive screen locations so quickly, the image appears to be moving. For example, type in the following program and RUN it.


```

10 PRINT"□"
20 FOR I=1024 TO 2023 STEP41
30 POKEI,81
35 POKE54272+I,7
40 FOR J=1TO45:NEXT
45 POKEI,32
50 NEXT
100 FOR I=2009TO1450 STEP-39
110 POKEI,81
120 POKE54272+I,7
130 FOR J=1TO45 :NEXT
140 POKEI,32
150 NEXT
160 GOTO20

```

This is your first taste of animation. You have just made a yellow ball bounce on the screen. Although the bouncing ball program is a simple example of animation, you are now on your way to programming sophisticated, animated graphics.

Here's an explanation of the program:

- Line 10 clears the screen. Loop 1, lines 10 through 50, displays and moves the ball from the top of the screen to the bottom. Line 20 begins a loop at the start of screen memory. Notice the FOR . . . NEXT statement has the words STEP 41. This tells the computer to increment the index variable I, by 41 locations at a time, starting at location 1024 and ending at location 2023. When STEP is not specified in a FOR . . . NEXT loop, your computer cycles through each index variable one at a time.
- Line 30 POKES screen code value 81 into the screen location according to the index variable I. The value 81 represents the screen code for the ball character that bounces on the screen. The first cycle of the loop POKES screen code 81 into location 1024. The second cycle POKES screen code 81 into screen location 1065 (1024 + 41). The third cycle POKES screen code 81 into screen location 1106 (1065 + 41) and so on. Each cycle through the loop skips 40 screen locations and POKES the ball 41 locations past the previous screen location.

- Line 35 POKES color code 7 (yellow) into the color memory location corresponding to the screen location that is POKED with the ball character. Remember, when you POKE a screen code value into screen memory, you also have to POKE a color code value into the corresponding color memory location. See Figure 7-1 and 7-2 to understand how each screen memory location corresponds to its own color memory location.
- In line 35, location $54272 + 1$ is the beginning of color memory during the first cycle of the loop ($54727 + 1024 = 55296$). The loop increments the color memory location the same way as screen memory. The second cycle of the loop increments the index variable I, so the POKE statement in line 35 POKES the color code value into location 55337 ($55296 + 41$). Color location 55337 corresponds to screen location 1065. As you can see, the loop takes care of POKING the screen location and corresponding color location so that the ball is always displayed correctly in yellow.
- Line 40 is an empty FOR . . . NEXT loop. It acts as a time delay to slow down the program so the animation appears smooth. Try the program without line 40. You'll notice the program becomes choppy.
- Line 45 POKES screen code value 32, the blank character, into the same screen location that was POKED with screen code 81 in line 30. This turns off the ball character. The ball character is turned on and off so quickly, it looks as though the ball is always on the screen.
- Line 50 is a NEXT statement. It updates the index variable I. The loop then cycles until the index variable equals 2023. At that point the program executes loop 2.
- Loop 2 bounces the ball upward and off the right side of the screen. Loops 1 and 2 both have the same statements, except different screen memory locations are decremented in line 100 instead of incremented as in line 20. The GOTO statement in line 60 tells the computer to go back to line 20 and execute everything again. The GOTO statement gives you a way to RUN your programs continuously. Stop the program by pressing the RUN/STOP key.

Here's another animation program that bounces the yellow ball off all four "walls" of the screen. This program is based on program three, but it has five loops instead of three. Each of the five loops is just like the two loops in the preceding program, except that the last three loops use different screen locations to control the three additional bounces of the ball.


```

10 PRINT"█"
20 FOR I=1024 TO 2023 STEP41
30 POKEI,81
35 POKE54272+I,7
40 FOR J=1TO45:NEXT
45 POKEI,32
50 NEXT
100 FOR I=2009TO1450 STEP-39
110 POKEI,81
120 POKE54272+I,7
130 FOR J=1TO45 :NEXT
140 POKEI,32
150 NEXT
200 FOR I=1423TO1044 STEP-41
210 POKEI,81
220 POKE54272+I,7
230 FOR J=1TO45 :NEXT
240 POKEI,32
250 NEXT
300 FOR I=1050TO1554 STEP38
310 POKEI,81
320 POKE54272+I,7
330 FOR J=1TO45 :NEXT
340 POKEI,32
350 NEXT
400 FOR I=1544TO2009 STEP42
410 POKEI,81
420 POKE54272+I,7
430 FOR J=1TO45 :NEXT
440 POKEI,32
450 NEXT
490 GOTO100

```

Now that you can animate a simple graphic character, it's time to learn a much more sophisticated method called sprite animation.

• SPRITE GRAPHICS

SPRITE CONCEPTS

You've learned how to control color with the CTRL key, with PRINT statements, and with character string codes. You now know how to PRINT alphanumeric and graphic characters on the screen within quotes, as character strings, and by POKEing values directly into screen memory. Animating existing character images, as described in the last section, has certain limitations. For true graphic animation, you need a way to create your own images, color those images and control their movement on the screen. That's where sprites come in.

Sprites are programmable movable objects. They are animated, high resolution images you can create into any shape. You can move these images anywhere on the screen and color them in 16 colors. The Commodore 64 has a set of sprite registers that control the color, movement and shape of the sprite. Sprites on the C64 provide you with true animation and sophisticated graphics capabilities like no other home computer. You'll soon amaze yourself once you program sprites and control their movement on the screen.

A special chip inside the Commodore 64, called the VIC (Video Interface Controller (6566)) chip, controls graphics modes and sprites. Border and screen color registers as well as the sprite registers are all part of the VIC chip. The VIC chip normally can control 8 sprites at once. Through advanced programming you can control more than eight sprites. The VIC chip can even determine if a sprite has moved in front of or behind another sprite. The size of each sprite can also be expanded both vertically and horizontally. You can use sprites in any mode: standard character, multi-color, standard and multi-color bit map and extended color modes. See the discussion of Graphics Modes later in this section for more information.

Let's begin by examining the properties of characters first, and then relate them to sprites. A character on the screen is an 8 by 8 dot grid. Since there are 40 columns by 25 lines on the screen, the entire screen has 320 (40 x 8 dots per character width) dots across times 200 (25 lines x 8 dots per character height) tall, which equals 64,000 total dots.

Each character pattern requires 8 bytes of storage in character memory. Each of the eight rows of dots in the 8 by 8 character grid require a byte of memory storage. In other words, each screen dot requires a bit of memory, so an 8 by 8 dot grid consists of 64 square dots and requires 64 bits (8 bytes) of memory.

Each dot on the screen is called a pixel. Pixel is a computer term for picture element. A sprite is made up of a 24 by 21 pixel grid, compared to a

character which is an 8 by 8 pixel grid. The width of a sprite is 24 pixels, which is equal to the width of three screen characters (bytes). Since a sprite is 21 rows of three bytes wide, a sprite requires 63 bytes (21 rows x 3 bytes per row) of storage. Figure 7-3 illustrates the layout and storage requirements of a sprite.

DESIGNING A SPRITE IMAGE

The first step in programming a sprite is designing the sprite image. For a beginner, the best way to design a sprite is on a piece of graph paper. Draw a box 24 blocks across by 21 blocks tall, just like Figure 7-3. The box you have just drawn is 504 (21 x 24) square blocks. Each block represents a bit in memory. If you divide 504 by 8 bits per byte, you'll see that the sprite uses up 63 bytes of memory.

You can now start designing your sprite image. Keep in mind that each block within the box you have drawn represents one bit in the Commodore 64's memory. As you probably know by now, a bit can take on one of two values, zero or one. That is why a bit is called a binary digit, since the root "bi" means two. A zero (0) means that a bit is "off" and a one (1) means that a bit is turned "on".

When you are designing your sprite on a piece of graph paper, darken a block if you want that bit to be on, and leave a block blank if you want that bit off. The combination of darkened blocks and blank blocks forms your sprite image. That is, if you want to turn on a dot in the sprite image, you must turn on a corresponding bit in memory where the sprite DATA is stored.

Refer to Figure 7-4 as an example of designing a sprite on a piece of graph paper. Remember, the darkened blocks are "on" bits and the blank blocks are "off" bits. The sprite image in Figure 7-4 represents a smiling face. Use the blank sprite-making grid in Figure 7-5 to create your own sprite images.

CONVERTING YOUR SPRITE IMAGE INTO DATA

The next step in programming a sprite is coding the sprite image into data the computer can understand. On your sheet of graph paper, label the top of each column the same as in Figure 7-6.

Label the first eight columns as follows: 128, 64, 32, 16, 8, 4, 2, 1. Label the second and third set of eight columns the same way.

You now have three sub-sets (bytes) of eight columns (bits) per row, each labeled from 128 on the left to 1 on the right. Each 8 column sub-set represents 8 pixels that correspond to a byte of memory. Again, since there are 21 rows with three bytes each, the total amount of memory the sprite requires is 63 bytes.

FIGURE 7-3. SPRITE GRID

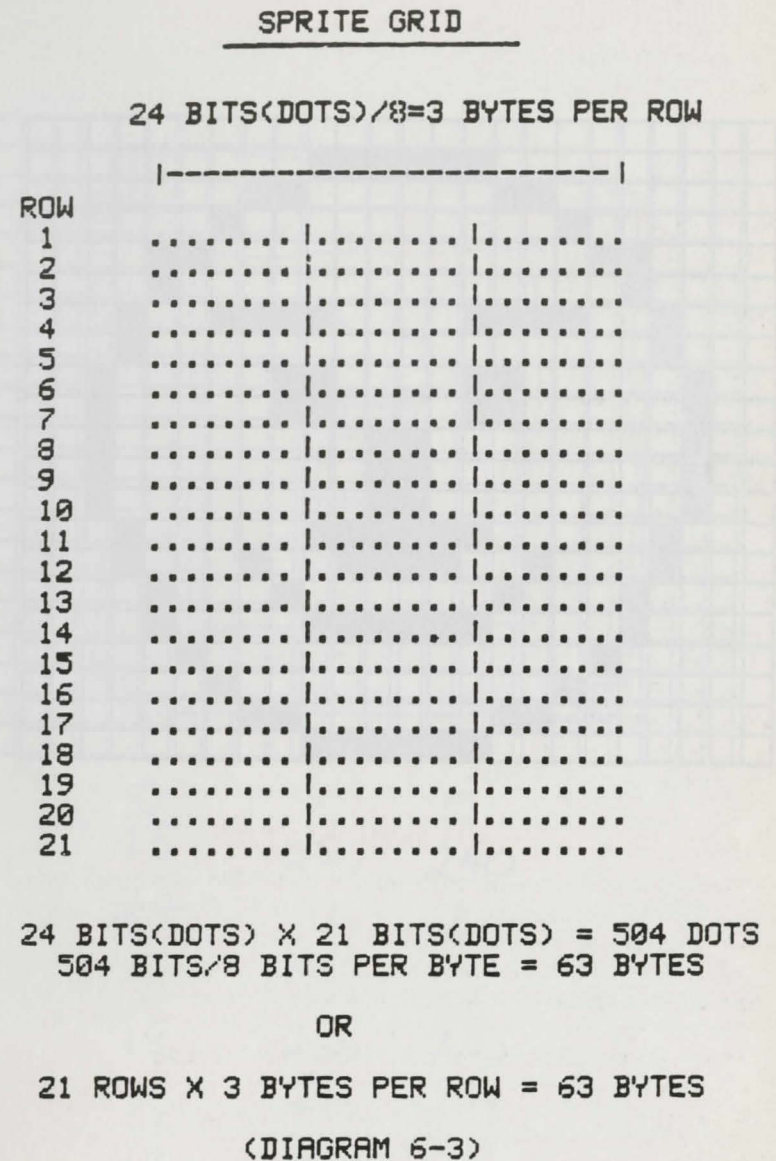


FIGURE 7-4. SPRITE-MAKING GRID

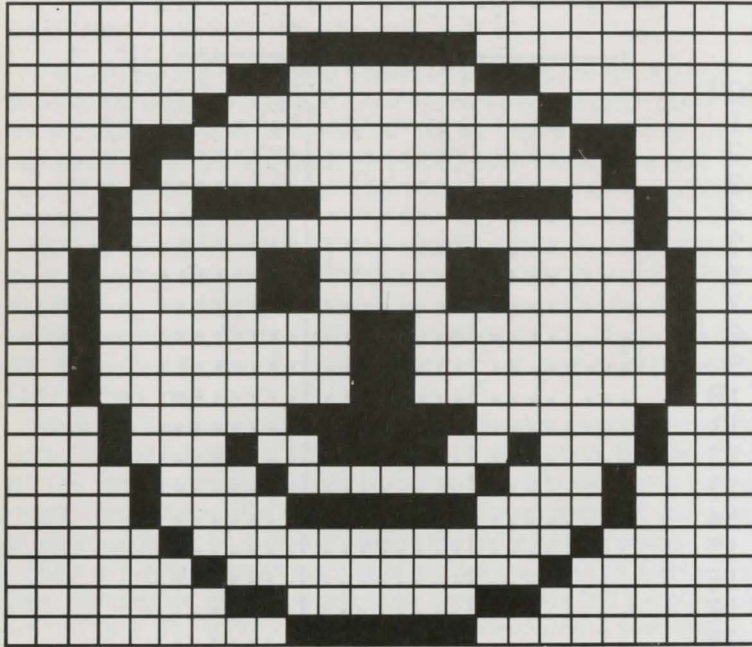


FIGURE 7-5. SPRITE-MAKING GRID

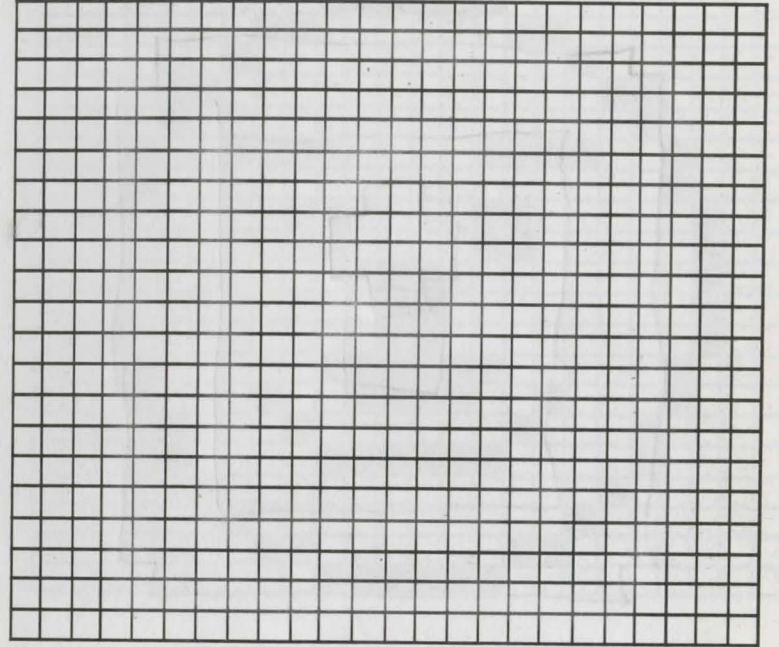
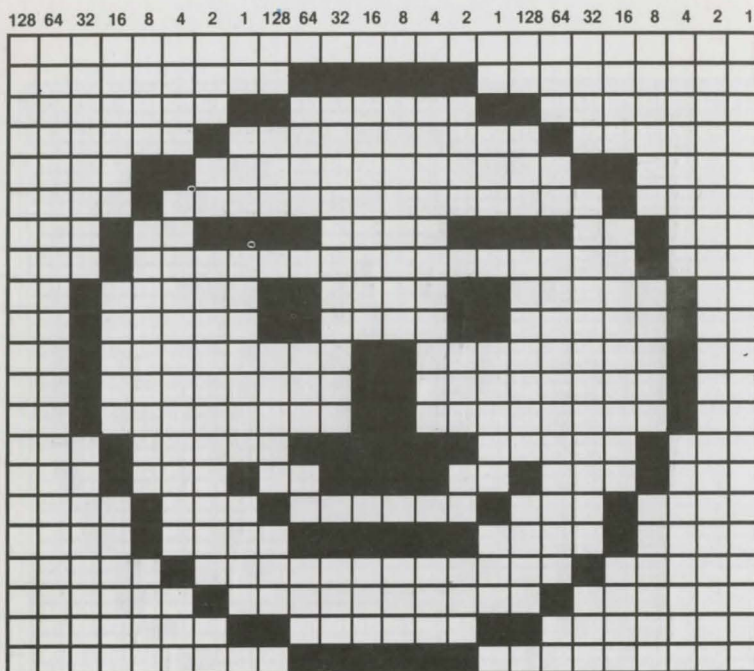
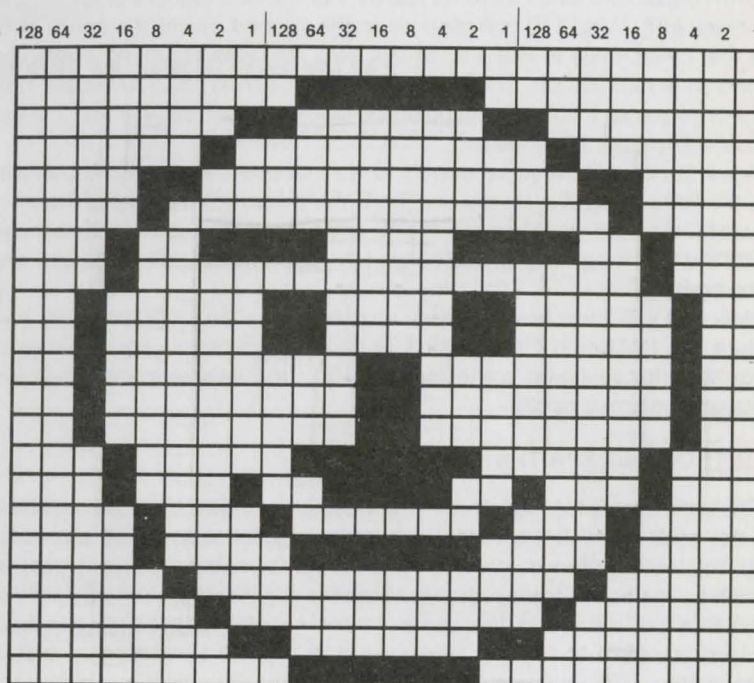


FIGURE 7-6. SPRITE-MAKING GRID.



Now you have a way to convert the graph paper image to computer data. For each darkened square within an eight column sub-set (byte) add up the number at the top of the column. Do this for each of the three 8 column sub-sets per row or a total of 63 times. Do not add column values in which individual squares are blank since these represent "off" pixels. Only add up the column values for the darkened squares. Once you calculate all the byte values for each eight column sub-set, you have 63 pieces of data to define your sprite. These values must be READ by the C64 and stored in DATA statements within a program. Study Figure 7-7 to grasp the concept of converting a sprite picture on graph paper to data used by the C64.

FIGURE 7-7. SPRITE-MAKING GRID



```

100 DATA 0,0,0
110 DATA 0,126,0
120 DATA 1,129,128
130 DATA 2,0,64
140 DATA 12,0,48
150 DATA 8,0,16
160 DATA 19,197,200
170 DATA 16,0,8
180 DATA 32,195,4
190 DATA 32,195,4
200 DATA 32,24,4
210 DATA 32,24,4
220 DATA 32,24,4
230 DATA 16,126,8
240 DATA 17,60,136
250 DATA 8,129,16
260 DATA 8,126,16
270 DATA 4,0,32
280 DATA 2,0,64
290 DATA 1,129,128
300 DATA 0,126,0
    
```


In the program shown in Figure 7-7, the DATA values in line 100 correspond to the three sub-sets of the first row of the sprite grid. All three pieces of DATA equal zero since all three sub-sets of the first row of the sprite grid are blank (off). Line 110 corresponds to the second row of the sprite grid. The first DATA value in line 110 equals zero, because again, no pixels are turned on in that sub-set. The second piece of DATA in line 110 equals 126, since the squares in the column positions labeled 64, 32, 16, 8, 4 and 2 in the middle sub-set are all turned on.

Again the third DATA value in line 110 is zero because none of the pixels in that 8 column sub-set is turned on. The DATA in line 120 represents the pixel values for the third row of the sprite grid, line 130 represents the values in the fourth row of the sprite grid, and so on. Line 300 corresponds to the last row of the sprite grid.

Now that you know how to design a sprite on a sheet of graph paper and code it into DATA that the Commodore 64 can understand, you are almost ready to write your first sprite program. But first let's examine the sprite registers and how they work.

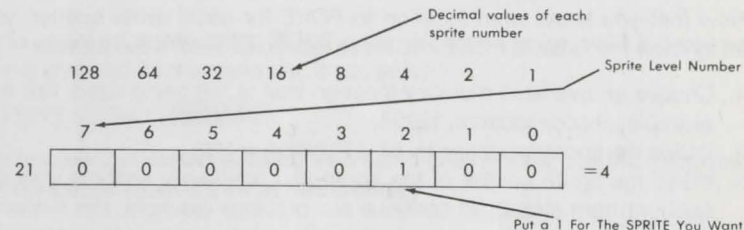
CONTROLLING SPRITES

Special memory locations within the video chip, known as sprite registers, are set aside to control sprites. Each sprite register is assigned a specific task. The first register you need to set is the sprite enable register at location 53269. As the name implies, the sprite enable register turns on a sprite. You must POKE a value into the sprite enable register, depending on which sprite(s) you want to turn on. Here's a list of the POKE values that enable each sprite:

Sprite No.	POKE Value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

You may have noticed the POKE value for each sprite is equal to two, raised to the sprite number. For example, the POKE value for sprite seven is two raised to the seventh power, which equals 128. Figure 7-8 illustrates this concept.

FIGURE 7-8. SPRITE POKE VALUES



The POKE command to turn on sprite 7 is:

POKE 53269,128

If you want to enable more than one sprite, add the POKE values of the sprites you want to turn on, and POKE the sum into the sprite enable register.

Now you have to store the sprite DATA somewhere in the Commodore 64's memory. Although you already converted your sprite picture into DATA as in lines 100 through 300 in Figure 7-7, you still have to READ that DATA and POKE it into memory. Before you can do that, you must tell the C64 where to store the DATA.

You point out where the DATA is stored using a sprite pointer. Each of the eight sprites has its own pointer. The following is a list of the sprite pointer memory locations:

Sprite No.	Memory Location
0	2040
1	2041
2	2042
3	2043
4	2044
5	2045
6	2046
7	2047

Now that you know what location to POKE for each sprite pointer, you need to know the value to POKE into these locations. Here's the formula:

1. Choose an available memory location that is not being used. For this example, choose location 12288.
2. Divide the chosen location by 64: $12288/64 = 192$
3. POKE the sprite pointer of the sprite you previously enabled with the quotient from step 2. To continue our previous example, the following POKE command uses the seventh sprite pointer to point to sprite DATA starting at location 12288:

```
POKE 2047, 192 RETURN
```

To determine other locations to store sprite DATA, consult The Commodore 64 Programmers Reference Guide.

As mentioned before, the sprite DATA must be READ and then POKEd into memory once the sprite pointers tell the Commodore 64 where to store the DATA. The sprite pointer was set with the previous POKE command. Now you can READ the sprite DATA you converted from your sprite image and POKE it into memory starting at location 12288. POKEing the DATA into memory actually creates the sprite. The following program segment READs the DATA and POKEs it into memory starting at location 12288.

```
50 FOR N = 0 to 62
60 READ 0
70 POKE 12288 + N,Q
80 NEXT
```

So far you have enabled the sprite, set the sprite pointer to tell the Commodore 64 where to store the sprite DATA and POKEd the sprite into memory. All you need to do now is to assign a sprite color and control the sprite's movement on the screen, and your sprite program will be finished.

Each sprite has its own sprite color register. The following is a list of sprite color register locations:

Sprite No.	Memory Location
0	53287
1	53288
2	53289
3	53290
4	53291
5	53292
6	53293
7	53294

To assign a sprite color, POKE a sprite color register with a color code between 0 and 15. For example, if you enter:

```
POKE 53294,7 RETURN
```

sprite seven is colored yellow. (For a list of color codes, see the Color Register discussion given earlier in this section.)

ANIMATING YOUR SPRITES

Animation is the last step before your program can RUN. The key behind animation is motion. Each of the eight sprites has two registers that control movement on the screen. One register is the sprite X position, which controls the horizontal sprite movement. The other is the sprite Y position, which controls the sprite's vertical movement. The following is a list of the sprite X and Y position registers for each sprite:

Sprite No.	Memory Location
0 - X pos	53248
0 - Y pos	53249
1 - X pos	53250
1 - Y pos	53251
2 - X pos	53252
2 - Y pos	53253
3 - X pos	53254
3 - Y pos	53255
4 - X pos	53256
4 - Y pos	53257
5 - X pos	53258
5 - Y pos	53259
6 - X pos	53260
6 - Y pos	53261
7 - X pos	53262
7 - Y pos	53263

The easiest way to control the vertical and horizontal coordinate values is within a FOR . . . NEXT loop. Set up a loop and POKE the index variable from the loop into the vertical and horizontal sprite position registers. For example, to move sprite 7 diagonally on the screen, use the following program segment:

```
85 FOR Z = 0 TO 200: REM Set up loop; index variable = z
90 POKE 53262,Z : REM Poke sprite 7 x pos. with index variable z
95 POKE-53263,Z : REM Poke sprite 7 y pos. with index variable z
98 NEXT : REM Update index variable position
```

Notice that the FOR . . . NEXT loop moves sprite 7 the maximum number of vertical values (200), but only moves horizontally 200 out of the 320 possible positions. That was done to keep the example program simple.

The sprite Y position register can store any of the 200 possible vertical position values. The sprite X position register cannot store all of the 320 horizontal position values because the sprite position register, like all other memory locations in the Commodore 64, can only represent a value up to 255.

How do you position a sprite past the 255th horizontal screen position? The answer is, you have to borrow a bit from another register in order to represent a value greater than 255.

An extra bit is already set aside in the Commodore 64's memory in case you want to move a sprite past the 255th horizontal location. Location 53264 controls sprite movement past position 255. Each of the 8 bits in 53264 controls a sprite. Bit 0 controls sprite 0, bit 1 controls sprite 1 and so on. For example, if bit 7 is on, sprite 7 can move past the 255th horizontal position.

Each time you want a sprite to move across the entire screen, turn on the borrowed bit in location 53264 when the sprite reaches horizontal position 255. Once the sprite moves off the right edge of the screen, turn off the borrowed bit so the sprite can move back onto the left edge of the screen. The following POKE command allows sprite seven to move past the 255th horizontal position:

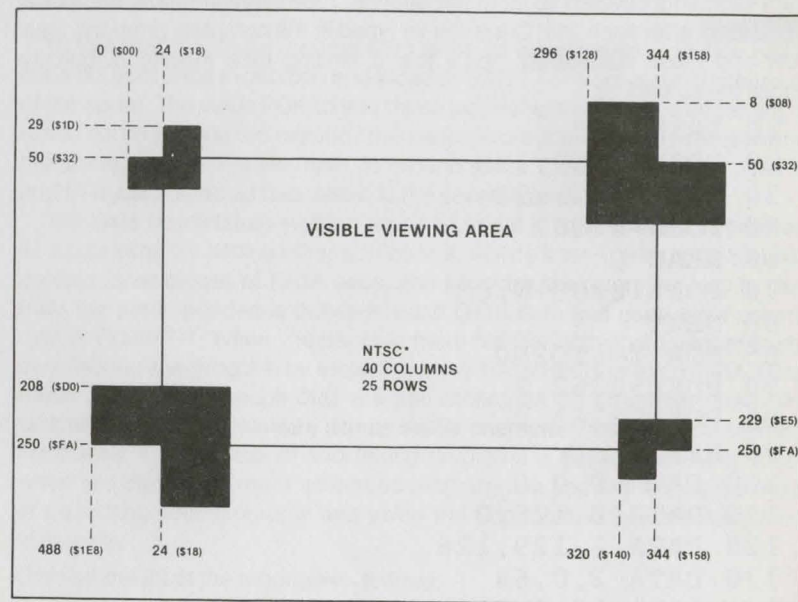
```
POKE 53264,128 RETURN
```

The number 128 is the resulting value from turning on bit 7. You arrive at this value by raising two to the seventh power. If you want to enable bit 5, raise two to the fifth power, which of course equals 32. The general rule is to raise two to the power of the sprite number that you want to move past the 255th horizontal screen position. Now you can borrow the extra bit you need to move a sprite all the way across the screen. To allow the sprite to reappear on the left side of the screen, turn off bit seven again, as follows:

```
POKE 53264,0 RETURN
```

Not all of the horizontal (X) and vertical (Y) positions are visible on the screen. Only vertical positions 50 through 249 and horizontal positions 24 through 342 are visible. In the example, when you moved sprite 7 on the screen, you started the sprite moving at horizontal location zero and vertical position zero. Location 0,0 is off the screen as is any horizontal location less than 24 and greater than 343. Any vertical location less than 50 and greater than 249 is also off the screen. The Commodore 64 OFF-SCREEN locations are set aside so that an animated image can move smoothly onto and off of the screen. Study Figure 7-9 to understand the layout of the visible horizontal and vertical sprite positions.

FIGURE 7-9. VISIBLE SPRITE POSITIONS



*North American television transmission standards for your home TV

TYING YOUR SPRITE PROGRAM TOGETHER

Now you are ready to tie all the sprite concepts together into a sprite program. Let's review the entire procedure. In order to program a sprite, you must:

1. Create the sprite image on a sheet of graph paper.
2. Convert the sprite image into DATA values the Commodore 64 can understand.
3. Enable the sprite.
4. Use a pointer to tell the Commodore 64 where to store the sprite DATA.
5. READ the sprite DATA and POKE it into memory, starting at the location indicated by the sprite pointer.
6. Color the sprite.
7. Control the sprite's movement on the screen.

The following program combines all the concepts, statements and program segments covered so far in this section. Type in the program, and press **RETURN** after each line. Once you've typed in the complete program, type RUN and press **RETURN**. You'll see a smiling face moving diagonally across the screen.

```
10 PRINT "☺"  
20 POKE53269,128  
30 POKE2047,192  
50 FORN=0TO62  
60 READ Q  
70 POKE12288+N,Q  
80 NEXT  
85 FOR Z=1TO200  
90 POKE53262,Z  
95 POKE53263,Z  
98 NEXT  
100 DATA 0,0,0  
110 DATA 0,126,0  
120 DATA 1,129,128  
130 DATA 2,0,64  
140 DATA 12,0,48  
150 DATA 8,0,16  
160 DATA 19,197,200  
170 DATA 16,0,8  
180 DATA 32,195,4
```

```
190 DATA 32,195,4  
200 DATA 32,24,4  
210 DATA 32,24,4  
220 DATA 32,24,4  
230 DATA 16,126,8  
240 DATA 17,60,136  
250 DATA 8,129,16  
260 DATA 8,126,16  
270 DATA 4,0,32  
280 DATA 2,0,64  
290 DATA 1,129,128  
300 DATA 0,126,0
```

Now add the following lines and RUN the program again.

```
55 POKE 53271,128  
57 POKE 53277,128
```

Notice that the sprite now appears twice its original size. Location 53277 controls horizontal expansion and location 53271 controls vertical expansion of the sprite. The value POKEd into these locations is calculated according to which sprite you want to expand. The general rule is raise two to the power of the sprite number. For example, to expand sprite 7, the value 128 in lines 55 and 57 is calculated as two raised to the seventh power, or 128.

You have successfully written your first sprite program. Use this program as a basis and try adding other sprites to it. Notice lines 100 through 300 only contain three pieces of DATA each. The program is written this way to illustrate the correspondence between each DATA item and each eight column byte in Figure 7-7. When you become more familiar with sprite concepts you can shorten the program by including more DATA items in each DATA statement. Lines 100 through 300 are still stored as 80 character lines. The spaces are stored in memory just as visible characters, but they use memory needlessly. The process of shortening programs is called crunching. Later, when you become a more advanced programmer, you will realize the value of crunching your programs and using the Commodore 64's memory more efficiently.

Change line 20 of the program as follows:

```
20 POKE 53269,224 : REM Enable sprites 7, 6 and 5
```

Add the following lines to the program and RUN it again. The REM statements are optional. You don't have to type them in. They document the program so you can follow each program step.

15 POKE 53280,1 : REM Change the border color to white
 17 POKE 53281,1 : REM Change the background color to white
 35 POKE 2046,192 : REM Set sprite 6 data pointer to 12288
 37 POKE 2045,192 : REM Set sprite 5 data pointer to 12288
 43 POKE 53293,6 : REM Color sprite 6 blue (6)
 45 POKE 53292,2 : REM Color sprite 5 red (2)
 92 POKE 53260,Z : REM Set sprite 6 horizontal (X) position
 94 POKE 53258,100 : REM Set sprite 5 horizontal (X) position
 96 POKE 53261,100 : REM Set sprite 6 vertical (Y) position
 97 POKE 53259,Z : REM Set sprite 5 vertical (Y) position
 99 GOTO 85 : REM Put the program into a continuous loop

Two more sprites appear on the screen, one from the left side of the screen and one from the top. Notice in the program, both sprites 5 and 6 use the same sprite DATA as sprite 7. That's why all three sprites look the same. If you want to change the way a sprite looks, design another sprite image on a piece of graph paper just as you did before. Then add another complete set of sprite DATA as in lines 100 through 300. In addition, READ the DATA and POKE it into a section of memory other than locations 12288 through 12351, since the other sprite DATA is already there. Finally, set the sprite DATA pointer to the starting location where the sprite DATA is POKEd into memory.

All three sprites in the above program store their DATA starting at location 12288. That's why lines 30, 35 and 37 POKE the same value into each of the three respective sprite DATA pointers. If all eight sprites were enabled, each one could use the same DATA and you would have eight identical sprites on the screen.

Lines 43 and 45 color sprite 6 blue and sprite 5 red. Lines 92 through 97 control the movement of sprites 5 and 6. Line 99 puts the program into a continuous loop. If you want to stop it, press the RUN/STOP key. Notice the sprite remains on the screen. To clear the screen completely, hold down the RUN/STOP key and press the RESTORE key.

Up to now, you've programmed three sprites on the screen. Try using all eight. In a relatively short time you should be able to create your own sprites in several colors and animate them on the screen. You can then move on to explore the very sophisticated color, graphics and animation features available on the C64. Consult the Commodore 64 Programmer's Reference Guide for more information on color graphics, sprites and animation.

• GRAPHICS MODES

The Commodore 64 can operate in five different graphics modes. They are divided into two groups known as character display modes and bit map

modes. Character display modes, as the name implies, display an entire 8 x 8 dot character grid at a time. In character display modes, the smallest unit of information you can display is an 8 x 8 pixel grid which equals one character. Bit map modes allow you to display each pixel, one at a time. Bit map mode gives you absolute control over the screen image. Graphics performed in bit map mode are referred to as high resolution graphics.

Both groups of graphics modes can be divided into separate subdivisions. Character display modes are separated into these three subdivisions:

1. Standard Character Mode
2. Multi-Color Character Mode
3. Extended Background Color Mode

Bit map modes are separated into these two subdivisions:

1. Standard Bit Map Mode
2. Multi-Color Bit Map Mode

Each of the character display modes get character information from one of two places in the Commodore 64's memory. Normally, character information is taken from character memory stored in a separate chip called a ROM (Read Only Memory). However, the Commodore 64 gives you the option of designing your own characters and replacing the original Commodore 64 characters with your own. Your own programmable characters are stored in a portion of the 64K of RAM (Random Access Memory) available to you in the C64.

The Commodore 64 normally operates in standard character mode. When you first turn on the Commodore 64, you are automatically in standard character mode. When you write programs, the C64 is also in standard character mode. Standard character mode displays characters in one of 16 colors on a background of one of 16 colors. All the information contained in this chapter operates in standard character mode except sprites. Sprites are classified separately from character display modes and bit map modes.

Multi-color character mode gives you more control over color than the standard graphics modes. Each screen dot within an 8 x 8 character grid can have one of four colors, compared to the standard modes which can only have one of two colors. Multi-color mode uses two additional background color registers. The three background color registers and the character color register together give you a choice of four colors for each dot within an 8 x 8 dot character grid.

Multi-color mode has one disadvantage. Each screen dot in multi-color mode is twice as wide as a dot in standard character mode and standard bit map mode. As a result, multi-color mode has only half the horizontal resolu-

tion (160 x 200) of the standard graphics modes. However, the increased control of color more than compensates for the loss in horizontal resolution.

Extended background color mode allows you to control the background color and foreground color of each character. Extended background color mode uses all four background color registers. In extended color mode, however, you can only use the first 64 characters of the screen code character set. The second set of 64 characters is the same as the first, but they are displayed in the color assigned to background color register 2. The same holds true for the third set of 64 characters and background color register 3, and the fourth set of 64 characters and background color register 4. The character color is controlled by color memory. For example, in extended color mode, you can display a purple character with a yellow background on a black screen.

Standard bit map mode allows you to control each screen dot in one of two colors. This gives you the ability to create detailed graphic images on the screen. Bit mapping is a technique that stores a bit in memory for each dot on the screen. If the bit in memory is turned off, the corresponding dot on the screen becomes the color of the background. If the bit in memory is turned on, the corresponding dot on the screen becomes the color of the foreground image. The series of 64,000 dots on the screen and 64,000 corresponding bits in memory control the image you see on the screen. Most of the finely detailed computer graphics you see in demonstrations and video games are bit mapped high resolution graphics.

Multi-color bit map mode is a combination of standard bit map mode and multi-color character mode. You can display each screen dot in one of four colors within an 8 x 8 character grid. Again, as in multi-color character mode, there is a tradeoff between the horizontal resolution and color control.

This section has described a variety of color and graphics techniques based on advanced programming concepts. The full explanation of these concepts is beyond the scope of this Guide. If you want more details on graphics techniques and graphics programming, refer to the Commodore 64 Programmer's Reference Guide.

The next section completes your introduction to the Commodore 64 computer by outlining the varied sound and music capabilities available to you through the C64.



MUSIC AND SOUND

This section introduces the Commodore 64's versatile music and sound capabilities

The SID Microprocessor	107
Music	107
—Playing From Sheet Music	107
—Obtaining the Data	108
—Writing the Program	110
Sound Effects	112
—Program Notes	114

• THE SID MICROPROCESSOR

A special microprocessor in the C64 known as the SID (Sound Interface Device) provides the C64 with extraordinary capabilities in generating musical tones and sound effects. This section introduces you to these capabilities. For more details, see Appendix G of this book and consult the Programmer's Reference Guide.

• MUSIC

The Commodore 64 is capable of producing musical tones over a large range—a full nine octaves for up to three separate voices (musical instruments) simultaneously. You can teach your C64 to play anything from *Happy Birthday* to Beethoven's Fifth Symphony.

By controlling a series of internal registers in the SID, you can program your C64 to play a variety of complex musical sounds. These sounds or notes have the qualities of a particular musical instrument and vary in pitch and duration.

PLAYING FROM SHEET MUSIC

In a musical score sheet you will find notes indicated by position and appearance. Compare these with Figure 8-1 for the note name and Figure 8-2 for note duration.

FIGURE 8-1. NOTE NAMES

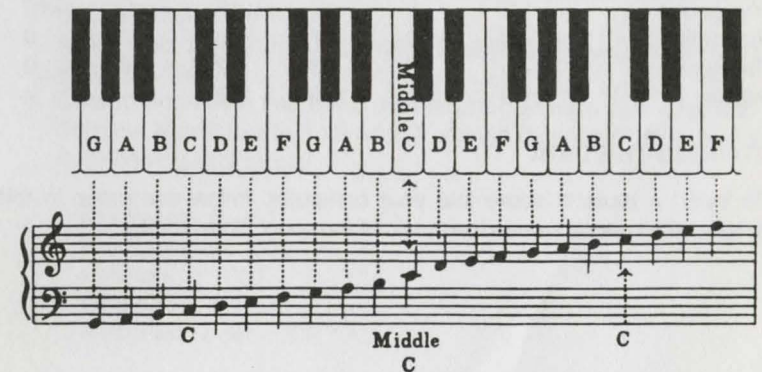


FIGURE 8-2. NOTE DURATION



To create these notes through the speakers of your monitor or TV, you must turn ON several registers in the SID microprocessor. There are seven registers for each of three voices. Each must be filled with a particular value. See Table 8-1 for the values of registers 2 through 6. Registers 0 and 1 are for sound frequency and are adjusted later in the program.

Table 8-1. Sound Register Values

Register number	2	3	4-ON	4-OFF	5	6
Musical instruments:						
Piano	225	0	65	64	9	0
Flute	0	0	17	16	96	0
Harpichord	0	0	33	32	9	0
Xylophone	0	0	17	16	0	240
Accordian	0	0	17	16	102	0
Trumpet	0	0	33	32	96	0
Noise	0	0	129	128	-	-

OBTAINING THE DATA

To insert a musical score into your computer, follow each step in this example, which incorporates the music of the song "Tom Dooley":

CHORUS:

Hang down your head, Tom Doo - ley,

Hang down your head and cry. —

Hang down your head, Tom Doo - ley,

Poor boy, you're bound to die. —

1. Select the musical instrument and determine the register values from Table 8-1.
Piano: register 2 is 255, register 3 is 0; register 4 is 65 for ON and 64 for OFF; register 5 is 9 and register 6 is 0.
2. Determine the name and value of each note; use Figures 8-1 and 8-2 as guides. Tabulate the results.
Notes read: D (eighth), D (eighth), E (quarter), G (quarter), B (half), B (half), etc.
3. Convert each note into the proper register settings called N1 and N2 from the Note Table in Appendix G and the duration (DR), based upon the following note values:

Eighth note = 250
 Quarter note = 500
 Half note = 1000
 Whole note = 2000
 A note with a dot = DR * 1.5

Tabulated Data				
Note	Value	N1	N2	DR
D	1/8	18	104	250
D	1/4	18	104	500
D	1/8	18	104	250
E	1/4	20	169	500
G	1/4	24	146	500
B	1/2	30	245	1000
B	1/2	30	245	1000
etc.				

- Write the program.

NOTE: Registers 2, 3, 4, 5 and 6 are set based on the musical instrument. Registers 0 and 1 are based upon each note and will vary. There is a register 24. It is the volume for all instruments and is always set to 15. The volume from your speaker is controlled by the monitor's volume control.

WRITING THE PROGRAM

Playing music requires turning on the appropriate registers, reading the notes and turning the sound on and off. All the registers can be turned on early in the program except register 4, which is turned on only when the music is needed.

Selecting a register is done by the BASIC term POKE, followed by the register number plus 54272, a comma and the proper value.

- Set all the registers to zero:

```
10 S = 54272:FOR SW = S TO S + 24: POKE SW,0:NEXT SW
```

- Set the volume to the maximum of 15:

```
20 POKE S + 24,15
```

- Turn on registers 2, 3, 5 and 6, based upon the instrument you are using (in this case, the piano):

```
30 POKE S + 2,255
```

```
40 POKE S + 3,0
```

```
50 POKE S + 5,9
```

```
60 POKE S + 6,0
```

- POKE a note into registers 1 and 0 from the table developed above. Since it will vary, represent the value with variable names N1 and N2.

```
80 POKE S + 1,N1:POKE S,N2
```

- Activate the sound with register 4, using the value for the proper instrument (65 for piano):

```
90 POKE S + 4,65
```

- Keep the sound on for the required time based on the value of DR in your table. Since this value is a variable, it is represented by its variable name, DR:

```
100 FOR Z = 1 TO DR: NEXT Z
```

- Turn off the sound, using the proper value:

```
110 POKE S + 4,64
```

- Keep the sound off for a very short time—about a tenth of a second.

```
120 FOR T = 1 TO 50: NEXT T
```

- Continue steps 4 through 8 with successive notes by using a READ statement and a loop.

```
70 READ N1, N2, DR
```

```
125 GOTO 70
```

- Store the note information in DATA statements. For simplicity, each DATA statement below represents one note:

```
130 DATA 18,104,250
```

```
132 DATA 18,104,500
```

```
134 DATA 18,104,250
```

```
136 DATA 20,169,500
```

```
138 DATA 24,146,500
```

```
140 DATA 30,245,1000
```

```
142 DATA 30,245,1000
```

```
etc.
```

- Include a means to stop the program:

```
75 IF N1 = 0 THEN END
```

```
200 DATA 0,0,0
```

Your sample program, when completed from sheet music, will look like this:

```
5 REM CHORUS FROM TOM DOOLEY
```

```
10 S = 54272:FOR SW = S TO S + 24:POKE SW,0:NEXT
```

```

20 POKE S + 24,15
30 POKE S + 2,255
40 POKE S + 3,0
50 POKE S + 5,9
60 POKE S + 6,0
70 READ N1,N2,DR
75 IF N1 = 0 THEN END
80 POKE S + 1,N1:POKE S,N2
90 POKE S + 4,65
100 FOR Z = 1 TO DR:NEXT Z
110 POKE S + 4,64
120 FOR T = 1 TO 50:NEXT T
125 GOTO 70
130 DATA 18,104,250,18,104,500,18,104,250,20,169,500,24,146,500
140 DATA 30,245,1000,30,245,1000
150 DATA 18,104,250,18,104,500,18,104,250,20,169,500,24,146,500
160 DATA 27,148,2000
170 DATA 18,104,250,18,104,500,18,104,250,20,169,500,24,146,500
180 DATA 27,148,1000,27,148,1000
190 DATA 27,148,250,27,148,500,30,245,250,24,146,500,
    20,169,500,24,146,1500
200 DATA 0,0,0

```

Be sure to raise the volume on your monitor when you run your program. To double the tempo, change line 100:

```
100 FOR T = 1 TO DR/2: NEXT T
```

To play a different song, change DATA statements to the appropriate values.

Now that you have created your first song, experiment with other instruments by varying the register values. You can also combine several voices to represent chords or other instruments by adding 7 or 14 to each of the register numbers (except register 24). Thus, registers 7 through 13 can control the second voice, and registers 14 through 20 the third voice.

• SOUND EFFECTS

Besides music, you can also create special sound effects by using the noise registers and varying the sound characteristics known as ADSR (Attack, Decay, Sustain and Release). These are combined in registers 5 and 6. A thorough explanation is provided in the Programmer's Reference Guide. Below are register values for sample sound effects.

Sound Effects Register Values

Registers Variable Names	0	1	2	3	4-ON	4-OFF	5	6	*	24
Sound effects:										
Police Siren	85	36	0	0	33	32	136	129	350	15
Crash	251	5	0	0	129	128	129	65	50	0
Rocket Blast-off	100	25	0	0	129	128	9	129	50	0
Machine Gun	75	34	0	0	129	128	8	1	50	15
Wailing	N2	40	0	0	65	64	15	0	1	15
Shooting	200	40	0	0	129	128	15	15	1	0

* Not a register. Part of the timing loop.

The following program, called "Sound Effects", incorporates all these variables and can produce each of these sounds. The technique is identical to creating music, except generally only one note is needed; hence there are no data statements. For details, see the Programmer's Reference Guide.

```

10 CLR: REM ** SOUND EFFECTS **
15 PRINT"WHICH SOUND EFFECT?":PRINT "1. WAILING":PRINT "2. SHOOTING":PRINT "3. ";
16 PRINT"SIREN":PRINT"4. ROCKET":PRINT"5. CRASH":PRINT"6. MACHINE GUN"
17 INPUT X
20 S = 54272:FOR SW = S TO S + 24:POKESW,0:NEXT:
K = - 1:T1$ = "000000"
21 ON X GOTO 23,24,25,26,27,28
23 V = 15:N1 = W1 = 65:W2 = 64:AD = 15:SR = 0:DR = 1:P1 = 9:P2 = 255:Q = 1:
GOTO30:REM WAILING
24 N2 = 200:N1 = 40:W1 = 129:W2 = 128:AD = 15:SR = 15:DR = 1;GOTO30:REM SHOOTING
25 N2 = 85:N1 = 36:W1 = 33:W2 = 32:AD = 136:SR = 129:DR = 350:V = 15:Q = 2:
GOTO30:REM SIREN
26 N2 = 100:N1 = 25:W1 = 129:W2 = 128:AD = 9:SR = 129:DR = 50:K =
25:GOTO30:REM ROCKET
27 N2 = 251 :N1 = 5:W1 = 129:W2 = 128:AD = 129:SR = 65:DR = 50:
GOTO30:REM CRASH
28 N2 = 75:N1 = 34:W1 = 129:W2 = 128:AD = 8:SR = 1:DR = 50:V = 15:REM

```


MACHINE GUN

```
30 POKE S + 2,P2:POKE S + 3,P1:REM PULSE
40 POKE S + 5,AD:POKE S + 6,SR:REM ADSR
50 POKE S + 1,N1:POKE S,N2:REM NOTE
55 IF Q = 2 THEN Q = 3
56 IF Q = 2 THEN POKE S + 1,64:POKE S,188
60 POKE S + 4,W1:REM ON SWITCH
63 IF Q < > 1 GOTO 70
65 FOR N2 = 200 TO 5 STEP -1:POKE S,N2:NEXT N2
68 FOR N2 = 150 TO 5 STEP -1:POKE S,N2:NEXT N2
70 FOR VL = 15 TO V STEP K:POKE S + 24,VL:REM VOLUME
80 FOR T = 1 TO DR:NEXT T:REM DURATION
90 NEXT VL
100 POKE S + 4,W2:REM SOUND OFF
110 IF TI$ > = "000005" THEN 10
115 IF Q = 3 THEN Q = 2:GOTO 56
120 GOTO 50
```

READY.

PROGRAM NOTES

The Sound Effects program contains six sound effects the user can pick from. Lines 10 through 21 clear all the variables and request a selection. The variable K in line 20 is necessary for the rocket sound. TI\$ sets the built-in timer to zero. Lines 23 through 28 establish the values of the register variables for each sound. Lines 30 through 50 enter these values into the proper registers. The variable Q in lines 55, 56 and 115 restricts those lines to the siren. The variable Q in line 63 restricts lines 65 and 68 for wailing only. Line 70 allows for a variable volume; where none was required, V was set to 15. Line 80 allows for a variable note duration; when not required, the variable DR was set to 1. Lines 60 and 100 are the main registers. Line 110 cuts off the sound after five seconds. You can then select another effect.

Although by now you have experienced first hand the versatility and power of the Commodore 64 computer, you probably realize that you have only begun to tap the potential of this extraordinary computer. The appendices to this Guide suggest many additional sources of information that you can use to further explore the fascinating world of computing with the C64.

APPENDICES

A.	Help _____	117
	—Error Messages _____	117
	—Troubleshooting Chart _____	120
B.	Peripherals and Software _____	123
	—Commodore Connections for Peripherals _____	123
	—Commodore Software _____	128
C.	Screen and Color Memory Maps _____	136
D.	Screen Display Codes _____	138
E.	ASCII And CHR\$ Codes _____	141
F.	Sprite Register Map _____	144
G.	Music Note Table _____	147
H.	Sound Control Settings _____	149
I.	Derived Trigonometric Functions _____	151
J.	Abbreviations of BASIC Keywords _____	152
K.	BASIC Conversions _____	155
L.	Recommended Reading List _____	157

APPENDIX A

• HELP

To help you with questions or problems about Commodore hardware, software or services, this appendix provides several sources of information.

• ERROR MESSAGES

MESSAGE	What the Problem Is	What to Do
BAD DATA	String data was received from an open file, but the program was expecting numeric data.	Make sure data was saved with a separator between each.
BAD SUBSCRIPT	The program was trying to reference an element of an array whose number is outside the range specified in the DIM statement.	Verify you have dimensioned the array properly. In direct mode, have the C64 print the value of the subscript as a clue.
BREAK	Program execution was stopped because you hit the STOP key.	Use the CONT command to proceed or reRUN the program.
CAN'T CONTINUE	The CONT command will not work, either because the program was never RUN, there has been an error, or a line has been edited.	You probably made a correction; reRUN the program.
DEVICE NOT PRESENT	The required I/O device not available for an OPEN, CLOSE, CMD, PRINT #, INPUT #, or GET #.	Verify the peripheral you are calling for is on and proper OPEN statement is used.
DIVISION BY ZERO	Division by zero is a mathematical oddity and not allowed.	Command the C64 to print the suspect variables to determine which one became a zero.
EXTRA IGNORED	Too many items of data were typed in response to an INPUT statement. Only the first few items were accepted.	Check your punctuation.
FILE NOT FOUND	No file with that name exists.	Verify you have the correct tape or disk and you spelled the name correctly; note especially spacing and upper-case characters.

MESSAGE	What the Problem Is	What to Do
FILE NOT OPEN	The file specified in a CLOSE, CMD, PRINT #, INPUT #, or GET #, must first be OPENed.	Open file. Verify you used proper file number.
FILE OPEN	An attempt was made to open a file using the number of an already open file.	Close file first or use new file number.
FORMULA TOO COMPLEX	The string expression being evaluated should be split into at least two parts for the system to work with, or a formula has too many parentheses.	Use smaller strings. Reduce the number of parentheses.
ILLEGAL DEVICE NUMBER	Occurs when you try to access a device illegally (e.g., LOADING from keyboard, screen or RS-232C).	Use correct device number.
ILLEGAL DIRECT	The INPUT statement can only be used within a program, and not in direct mode.	Use another command.
ILLEGAL QUANTITY	A number used as the argument of a function or statement is out of the allowable range.	Use direct mode to determine the value of the variables at the moment. Correct negative subscripts. Verify dimensions are large enough.
LOAD	There is a problem with the program on disk.	Reload.
MISSING FILE NAME	LOADs and SAVEs from the serial port (e.g., the disk) require a file name to be supplied.	Key in the file name.
NEXT WITHOUT FOR	This is caused by either incorrectly nesting loops or having a variable name in a NEXT statement that doesn't correspond with one in a FOR statement.	Verify the loop has a starting and ending point. Do not jump into the middle of a loop.
NOT INPUT FILE	An attempt was made to INPUT or GET data from a file which specified to be for output only.	Correct the OPEN statement's secondary address.
NOT OUTPUT FILE	An attempt was made to PRINT data to a file which was specified as input only.	Correct the OPEN statement's secondary address.

MESSAGE	What the Problem Is	What to Do
OUT OF DATA	A READ statement was executed but there is no data left unREAD in a DATA statement.	Verify data was not missed; add more data if necessary.
OUT OF MEMORY	There is no more RAM available for program or variables. This may also occur when too many FOR loops have been nested, or when there are too many GOSUBs in effect.	Reduce the quantity of GOSUBs and FOR NEXT loops operating at once. Reuse loop variables where possible to prevent too many unfinished loops. Clean up the memory using FRE(X) function.
OVERFLOW	The result of a computation is larger than the largest number allowed, which is 1.70141884E + 38.	Check your computation steps.
REDIM'D ARRAY	An array may only be DIMensioned once. If an array variable is used before that array is DIM'd, an automatic DIM operation is performed on that array setting the number of elements to ten, and any subsequent DIMs will cause this error.	If the array was identified early it was automatically dimensioned to 10. Locate the DIM statement before using the variable.
REDO FROM START	Character data was typed in during an INPUT statement when numeric data was expected. Just re-type the entry so that it is correct, and the program will continue by itself.	Provide the proper INPUT response.
RETURN WITHOUT GOSUB	A return statement was encountered, and no GOSUB command has been issued.	Verify the program ends before coming to subroutines tagged at program's end.
STRING TOO LONG	A string can contain up to 255 characters.	Keep strings to 255 characters and any single INPUT to 80 characters.
?SYNTAX ERROR	A statement is unrecognizable by the Commodore 64. A missing or extra parenthesis, misspelled keywords, etc.	Look for spelling or grammar errors or words not in the BASIC vocabulary.
TYPE MISMATCH	This error occurs when a number is used in place of a string, or vice-versa.	Verify \$ signs were typed where they belong.

MESSAGE	What the Problem Is	What to Do
UNDEF'D FUNCTION	A user defined function was referenced, but it has never been defined using the DEFFN statement.	Define the function with DEF within the program.
UNDEF'D STATEMENT	An attempt was made to GOTO or GOSUB or RUN a line number that doesn't exist.	Make sure line numbers exist.
VERIFY	The program on tape or disk does not match the program currently in memory.	Save the program again, under another name.

NOTE: A common error is to type a 41-character line, not hit **RETURN** and type a second line as if it were a new line. **RETURN** will then treat both lines as one. To find this type of error, list your program and continue hitting **RETURN**. Watch the cursor jump to the beginning of each instruction line. A skipped line means it was tagged onto the line above it. Retype these lines.

• TROUBLESHOOTING CHART

For problems which appear to be hardware oriented use this troubleshooting chart first:

Symptom	Cause	Remedy
Indicator Light not "On"	Computer not "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

Symptom	Cause	Remedy
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 (3 or 4)
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
Picture with poor color	Bad color adjustment on TV	Adjust color/hue/brightness controls on TV
Sound with excess background noise	TV volume too high	Adjust volume of TV
Picture OK, but no sound	TV volume too low	Adjust volume of TV
	Aux. output not properly connected	Connect sound jack to aux. input on amplifier and select aux. input
Computer Stuck; cursor not flashing	Computer inadvertently received instructions to disable keyboard; or the printer, cassette or disk drive is in listening mode	While depressing the RUN/STOP key press RESTORE key twice; or reset the accessories by turning off and on; or reset the computer off and on.
Computer displays garbage on the screen	Overheating	Pull plug on power supply when not using computer for extended periods (overnight).

Books

Many books have been published about the Commodore 64. For a sampling see Appendix L.

Magazines

Subscribe to "Commodore Microcomputers" and "Power/Play" for the latest on Commodore hardware and software. For subscription information call 800-345-8112 (In Pennsylvania call 800-662-2444).

User Groups

There are over 1,000 user groups (clubs) dedicated to helping Commodore owners and sharing experiences. Find the address of the nearest user group in the next issue of "Commodore Microcomputers" or "Power/Play." Or for information on how to start a user group in your area write to:

Commodore User Groups
1200 Wilson Drive
West Chester, Pa. 19380

Commodore Information Network

Use your AUTOMODEM or VICMODEM to communicate directly with Commodore or other C64 owners through CompuServe and use the Hotline or the Special Interest Group Bulletin Boards. Information is provided with your modem.

Customer Support Hotline

For assistance by telephone, call the customer support hotline:

215-436-4200

Hours: 9-8 Monday to Friday (Eastern Time)
10-4 Saturday

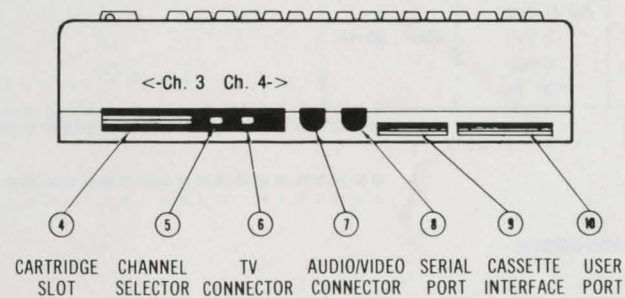
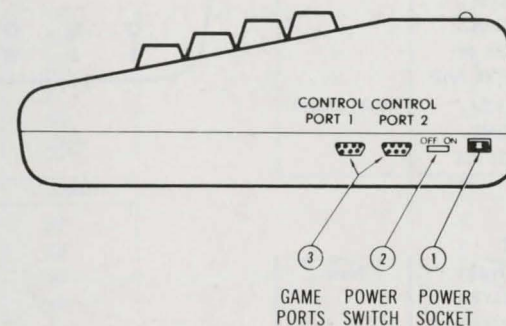
or write to

Commodore Customer Support
1200 Wilson Drive
West Chester, Pa. 19380

APPENDIX B

• PERIPHERALS AND SOFTWARE

• COMMODORE CONNECTIONS FOR PERIPHERALS



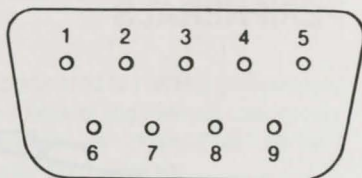
Side Panel Connections

1. Power Socket—The free end of the cable from the power supply is attached here.

- Power Switch—Turns on power from the transformer.
- Game Ports—There are two game ports, numbered 1 and 2. Each game port can accept a joystick or game controller paddle. The light pen can only be plugged into port 1. (Port 1 is the port closest to the front of the computer). Use the ports as instructed with the software.

Control Port 1

Pin	Type	Note
1	JOYA0	
2	JOYA1	
3	JOYA2	
4	JOYA3	
5	POT AY	
6	BUTTON A/LP	
7	+5V	MAX. 50mA
8	GND	
9	POT AX	



Control Port 2

Pin	Type	Note
1	JOYB0	
2	JOYB1	
3	JOYB2	
4	JOYB3	
5	POT BY	
6	BUTTON B	
7	+5V	MAX. 50mA
8	GND	
9	POT BX	

Rear Connections

- Cartridge Slot—This rectangular slot is a parallel port that accepts program or game cartridges, as well as special interfaces.

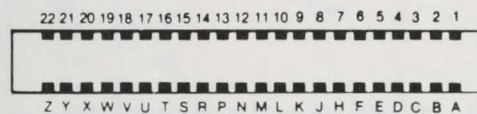
Cartridge Expansion Slot

Pin	Type
12	BA
13	DMA
14	D7
15	D6
16	D5
17	D4
18	D3
19	D2
20	D1
21	D0
22	GND

Pin	Type
N	A9
P	A8
R	A7
S	A6
T	A5
U	A4
V	A3
W	A2
X	A1
Y	A0
Z	GND

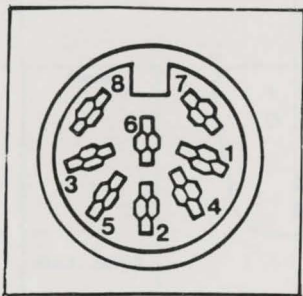
Pin	Type
1	GND
2	+5V
3	+5V
4	IRQ
5	R/W
6	Dot Clock
7	I/O 1
8	GAME
9	EXROM
10	I/O 2
11	ROML

Pin	Type
A	GND
B	ROMH
C	RESET
D	NMI
E	S 02
F	A15
H	A14
J	A13
K	A12
L	A11
M	A10



- Channel Selector—Use this switch to select which TV channel (3 or 4) the computer's picture will be displayed on.

6. TV Connector—This connector supplies both the picture and sound to your television set.
7. Audio/Video Connector—This DIN pin connector supplies direct audio and composite video signals. These can be connected to the Commodore monitor or used with separate components.

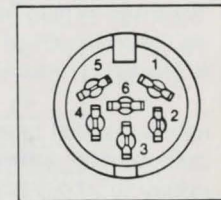


Pin	Type	Note
1	LUM/SYNC	Luminance/SYNC output
2	GND	
3	AUDIO OUT	
4	VIDEO OUT	Composite signal output
5	AUDIO IN	
6	COLOR OUT	Chroma signal output
7	NC	No connection
8	NC	No connection

8. Serial Port—A Commodore serial printer or 1541 single disk drive can be attached directly to the Commodore 64 through this port.

Serial I/O

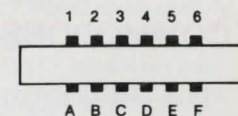
Pin	Type
1	SERIAL SRQIN
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	RESET



9. Cassette Interface—A Datasette recorder can be attached here to store programs and information.

Cassette

Pin	Type
A-1	GND
B-2	+5V
C-3	CASSETTE MOTOR
D-4	CASSETTE READ
E-5	CASSETTE WRITE
F-6	CASSETTE SENSE

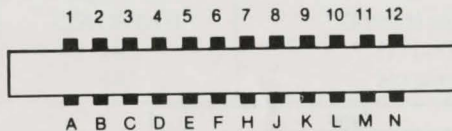


10. User Port—Various interface cartridges can be attached here, including the Commodore AUTOMODEM and RS232 communication cartridge. It is not recommended that this port support RS232-C printers.

User I/O

Pin	Type	Note
1	GND	
2	+5V	MAX. 100 mA
3	RESET	
4	CNT1	
5	SP1	
6	CNT2	
7	SP2	
8	PC2	
9	SER. ATN IN	
10	9 VAC	MAX. 100 mA
11	9 VAC	MAX. 100 mA
12	GND	

Pin	Type	Note
A	GND	
B	FLAG2	
C	PB0	
D	PB1	
E	PB2	
F	PB3	
H	PB4	
J	PB5	
K	PB6	
L	PB7	
M	PA2	
N	GND	



• COMMODORE SOFTWARE

Commodore supports the C64 with a full range of software in cartridges, disk and cassette including educational programs, financial software, productivity software, programming aids, business software, arcade games, Bally Midway games, music and strategy adventure games as well as books from the Commodore Library (See Appendix L).

Here is a list of Commodore software:

Productivity Software

Micro Cookbook Computer age solution to menu planning. Manage your recipes quickly and easily.

Easy Script 64 Our best word processor. Displays 764 lines by 40 characters. Prints more than 130 columns. Global/local search/replace/hunt/find. Super/subscripts. Insert/delete characters, lines, sentences, paragraphs . . . much more. Works with Easy Spell 64.

Easy Spell 64 Don't misspell it . . . Easy Spell It . . . with this automatic spelling checker. Includes 20,000 word Master Dictionary plus your own 10,000 word vocabulary. Requires Easy Script.

Easy Calc 64 Electronic spreadsheet on convenient plug-in cartridge. Color-bar graph feature. Display or print up to 254 rows by 63 columns. Calculates and edits entries automatically. Includes color-graphic capability. A must for budgeting, forecasting and calculation.

The Manager Flexible, multi-purpose database manager lets you design your own computerized reports and files for home or business. Address files, mailing list, project status, investments . . . 4 built-in applications, or design your own. Powerful arithmetic capabilities.

Easy Mail 64 Easy-to-use mailing list program. Use with Commodore printer to create address labels and lists for home or business purposes. Club mailings, membership lists, Christmas lists, direct mail uses, more.

Word/Name Machine Beginner's word processor. Easy-to-follow menus guide you through program operation. Form letters.

Financial Software

Easy Finance I Loan/Mortgage Computerize your loan and mortgage calculations with 12 loan functions including amortization, financial forecasting, Rule of 78's interest, property investment, cost analysis. Bar graphs.

Easy Finance II Basic Investing Calculate investment alternatives and current values of stock/bonds/annuities. 16 investment functions. Net present values, future values. Bar graphs.

Easy Finance III Advanced Investing Calculate weighted cost of capital, accrued interest on bonds, earnings per share, future uneven cash values, rate of return, present value of a tax deduction. 16 functions. Bar graph.

Easy Finance IV Business Management 21 Business calculation functions: Lease/Purchase Analysis, breakeven analysis, compensation, lease/purchase, optimal order quantities, business forecasting, much more. Bar graphs.

Easy Finance V Statistics and Forecasting Assess present/future sales, trends and other business parameters with 9 statistical and forecasting functions. Calculate average growth rate/expected values. Special help menu.

Financial Advisor Designed for the High School or College finance student or the Loan Officer at the local bank. Computes loans/investments/stocks/bonds with amazing speed and accuracy.

Business Software

General Ledger 8 general ledger options. 1500 transactions. 150 chart of-accounts. Posting integrated with other accounting modules. Custom income statement, trial balances, full reports.

Accounts Receivable/billing 11 billing functions. 150 invoices. 75 customers. 40 transactions/file. Billing, credit, receivables. Printed statements.

Accounts Payable/Checkwriting Combines tracking of vendor payables with integrated checkwriting system. Interfaced with other accounting modules.

Payroll For Business with 50 employees or less. 24 different payroll functions. Payroll checks include federal/state/other deductions. Integrated with General Ledger. Prints W2's and 941's.

Inventory Management Computerized tracking of 1000 inventory items. Stock receipts/issues/orders/adjustment with printed reports. Calculates use/reorders/economic order quantities/cost averaging and more. For all types of inventories including personal collections and insurance lists.

Magic Desk I Type & File Now you can type, file and edit personal letters and papers without learning any special commands! All Magic Desk commands are pictures. Just move the animated hand to the feature you want to use and you're ready to go.

Educational Programs

Intro to Basic I 17 programs, a 150-page manual and a flowchart template. This program is an instructional guide that teaches the fundamentals of programming in BASIC while assuming the user has no previous knowledge of programming.

Intro to Basic II Uses the same easy-to-understand approach presented in BASIC I. This package includes a 180 page manual with explanations, practice drills, examples and a disk with 33 programs. Learn more advanced techniques of BASIC programming with this terrific program!

Zortec and the Microchips (Ages 6 to 12) A fun way to teach young people how to program in BASIC. Help Zortek teach the Microchips to program the computer before the Zitrons attack.

Pilot (Ages 12 to Adult) A special language that helps nonprogrammers design computerized quizzes and drills. For teachers, parents and students.

LOGO (Ages 6 to Adult) Terrapin LOGO. The best, most powerful version of LOGO on any home computer. Includes sprite graphics, sound commands, turtle graphics and 400 page tutorial.

Chopper Math (Ages 7 and up) Practice basic math skills. Get the right answer and land the helicopter on its landing pad before it crashes.

Type Right (Ages 12 to Adult) No more hunting and pecking! Learn how to type on your computer with 17 lessons and 4 games.

Speed/Bingo Math (Ages 4 to 10) Two math games help you build math skills and have fun at the same time. One or two players. A Bally/Midway conversion.

Easy Match/Easy Count (Ages 4 to 6) Kinder Concepts Series. Practice identifying shapes and letters and counting objects—important pre-reading and pre-math skills.

What's Next/Letters or Numbers (Ages 4 to 6) Kinder Concepts Series. Practice in identifying correct sequences of numbers and letters.

Letter Sequences/The Long or Short of It (Ages 4 to 6) Kinder Concepts Series. Practice in identifying letter sequences and in recognizing which shape is longest or shortest.

A Letter Match/More or Less (Ages 4 to 6) Kinder Concepts Series. Upper and lower-case letters, more or less relationships, and matching numbers.

Shapes and Patterns/Group It (Ages 4 to 6) Kinder Concepts Series. Brightly colored shapes and sounds give practice in identifying shapes and patterns, grouping and regrouping. Excellent pre-reading and pre-math development drills.

A BEE C's (Ages 3 to 6) The Commodore Bee guides your child in learning the alphabet. Playing games reinforces this skill. Excellent tool for young children. It talks with Magic Voice!

Visible Solar System (Ages 7 to Adult) Authentic, astronomer-tested journey through the solar system. Tour the planets Earth, Mars, Jupiter and Saturn as well as asteroids, comets and meteors. Planet fun for space buffs!

Number Nabber/Shape Grabber (Ages 6 to 12) Two learning games in one. Builds arithmetic and shape identity skills. Lively music and sound effects make this a favorite.

Math Facts (Ages 5 to 10) Give practice in basic math facts. Several levels.

Numbers Galore (Ages 3 to 14) 3 different math programs. Number Match It for preschool, Math Facts Games for elementary, and Number Cruncher for middle school.

Frenzy/Flip Flop (Ages 6 to 14) Milliken EduFun Series Frenzy . . . subtraction and division . . . the hungry gator arrives . . . save the fish . . . play the Bonus game . . . the more you save . . . the more you play! Flip Flop . . . transformational geometry . . . look at the two figures . . . do they need to flip, turn or slide? . . . stand on your head . . . lie on your side . . . you'll flip over this game!

Golf Classic/Compubar (Ages 10 to 14) Milliken EduFun Series. Golf Classic (angle and length estimation). Fore! Play the angles. Choose distances. Multiple players. Sports fun! Compubar (read graphs, construct arithmetic expressions) add this bar; subtract that one. Did you read them correctly?

Gulp!/Arrow Graphics (Ages 6 to 12) Milliken EduFun Series. Gulp! (addition and multiplication drill) the race is on... add... multiply... faster, faster... don't get caught... watch out for those jaws! Arrow Graphics (problem solving and directionality) following the traveling arrow... where did it go?... left or right how many steps?

Alien Counter/Face Flash (Ages 4 to 9) Milliken EduFun Series. Alien Counter (counting) Flying saucers... numbers in the sky... aliens landing on Earth... another perfect encounter? Face Flash (counting, visual memory, and base ten) ready, set, go... now you see them... count fast... they're gone! How many?

Battling Bugs/Concentration (Ages 9 to 12) Milliken EduFun Series. Battling Bugs (positive and negative numbers) columns of bugs, get rid of them all! You might be the master exterminator! Concentration (equivalent fractions)... choose two... are they equal? Two players.

Easy Lesson/Easy Quiz For Teachers. Create your own lessons and quizzes using the power of the computer.

Arcade Games

International Soccer A Gold Medallion Game. As close to real soccer as you can get without putting on cleats! Realistic player and ball movement highlight this stunning version of the most popular sport in the world.

Jack Attack A Gold Medallion Game. Combines cartoon animation with strategic challenge. 64 different screens. A Commodore original rated a "must-buy"—Electronic Games Magazine.

Avengers Destroy attacking aliens with laser cannons, as you dodge their bullets. Classic arcade action. Multi-speed attacks.

Frogmaster Unique sports challenge. Train frogs to play football and rugby. Over 100 variations. Play against computer, friend or yourself.

Jupiter Lander Space landing simulation. Horizontal and vertical thrust. Softland scoring. Joystick control.

Le Mans Multi-obstacle road racing at its best. Arcade action and graphics. Night, water, ice and divided highway hazards.

Pinball Spectacular Real pinball action and thrills. Sound you won't believe. Chutes, lights, bumpers and more.

Radar Rat Race Beat the maze. Eat all the cheese. Beware of deadly cats and rats. Cartoon action for all ages.

Starpost Protect the Star Post from waves of invaders. 3 levels of skill. 99 levels of action.

Star Ranger Fight your way through hordes of space enemies. Avoid asteroids and land safely. Superb graphics combined with intriguing strategy.

Supersmash Racquetball arcade classic. 3 Games in 1. Many skill levels make this game a smash hit.

Tooth Invaders Reviewed by American Dental Association. Arcade action teaches good dental care. Beat Tooth Decay in 9 levels.

Triad One/Two players. Position yourself on tic-tac-toe grid for different attack strategies. Progressive difficulty levels test both reflexes and mind.

Dragonsden Arcade-style excitement in this Commodore original. Battle giant spiders, bats and the dragon in this contest of skill and reflexes. 3 levels.

Bally Midway Games

Blueprint Help J.J. build the "Ammo Machine" and save Loni. Parts are stored in a colorful maze of houses. Multi-skill and difficulty levels.

Clowns Amazing action under the Big Top. Clowns pop balloons for high-scoring, colorful acrobatic fun for all.

Gorf 4 Space action games in 1. Fly your fighter, defeat the Empire. Multi-skill levels. Talks with Magic Voice. "Best home version ever"—Creative Computing.

Kickman Ride the unicycle and catch falling objects. Multi-skill levels. Excellent graphics and superb sound. Watch out! Don't fall!

Lazarian 4 different screens. Multi-skill level space action. Rescue, evade obstacles and destroy the one-eyed leviathan.

Omega Race Fast space race action. Many skill levels. Avoid deadly mines as you eliminate droid forces.

Seawolf The classic two-player sea battle. Torpedo PT Boats and Destroyers. Great graphics and sound.

Wizard of Wor Fight your way through 25 mazes. Defeat the wizard and his pets. Two-player, multi-skill. Talks with Magic Voice. Brilliant conversion of the popular classic.

Solar Fox You're the pilot as you navigate your spaceship over a grid of colorful pulsating entities, using your laser to erase enemy life forms and other surprises. But be careful of the enemy fire or you'll be erased! Fast paced conversion of a Bally Midway original uses an unlimited number of levels and dozens of different patterns.

Strategy Adventures

Deadline Find the murder and solve the mystery in 12 hours. Inspector casebook and evidence included.

Starcross Travel through the mystery ship. Meet aliens, friend and foe. Face the challenge of your destiny. Galaxy Map included.

Suspended Awaken in 500 years. Solve varied and original puzzles to save your planet from total destruction.

Zork I, II, or III Zork Series. Fantasy adventure in a dungeon. Find all the treasure and escape alive. Three continued fantasies.

Music

Music Composer Create, play and save your tunes easily. Simulates up to 9 instruments. Notes appear on screen. Play your keyboard like a piano.

Music Machine Play piano or organ melodies and percussion rhythms together. Music staff shows notes on screen. Vibrato, tempo and pitch controls.

Programming Aids

Assembler 64 For experienced Assembly language programmers. Create, assemble, load and execute 6500 series Assembly language code. Macro assembler. Two machine language monitors. Editor and loaders. Support routines. User manual.

Simon's Basic Expands Commodore BASIC with 114 commands such as RENUMBER and TRACE, plus graphics commands. Programmers and novices love it! A must for the serious Commodore user.

Super Expander 64 Easy graphics and music. Draw points, lines, arcs, circles, ellipses, polygons. Create more sprites. Easy music programming. Combine text and graphics. Adds 21 special commands to BASIC.

• Personal Checklist

Use this checklist to keep track of what you purchase or plan to purchase.

Hardware:

- Printer
 - DPS 1101 DAISYWHEEL
 - MPS 801
 - MPS 802
 - MCS 801
- Commodore Color Monitor
- 1541 Disk Drive
- Datassette
- Magic Voice Speech Module
- Joysticks
- AUTOMODEM
- Paddles
- Light Pen

Software

- Productivity
 - _____
 - _____
 - _____
- Financial
 - _____
 - _____
- Educational
 - _____
 - _____
 - _____
- Business
 - _____
 - _____
- Games
 - _____
 - _____
 - _____
 - _____
- Music
 - _____
- Adventures
 - _____
 - _____
- Programming Aids
 - _____
- Miscellaneous
 - _____

Books: Commodore Library

- Programmer's Reference Guide
- Commodore 64 Adventures
- Business Applications on the Commodore 64
- Graphic Art: Using Turtle Graphics
- Mathematics on the C64
- Advanced Programming Techniques on the C64
- Artificial Intelligence on the C64
- Programming for Education on the C64
- 1541 Disk Companion

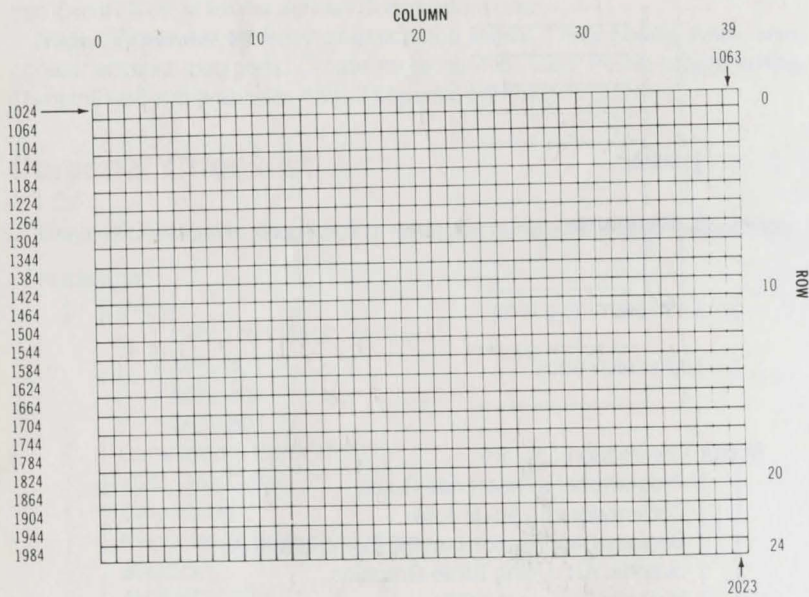
APPENDIX C

• SCREEN AND COLOR MEMORY MAPS

The following maps display the memory locations for identifying the characters on the screen as well as their color. Each map is separately controlled and consists of 1000 positions.

The characters displayed on the maps can be controlled directly with the POKE command.

SCREEN MEMORY MAP

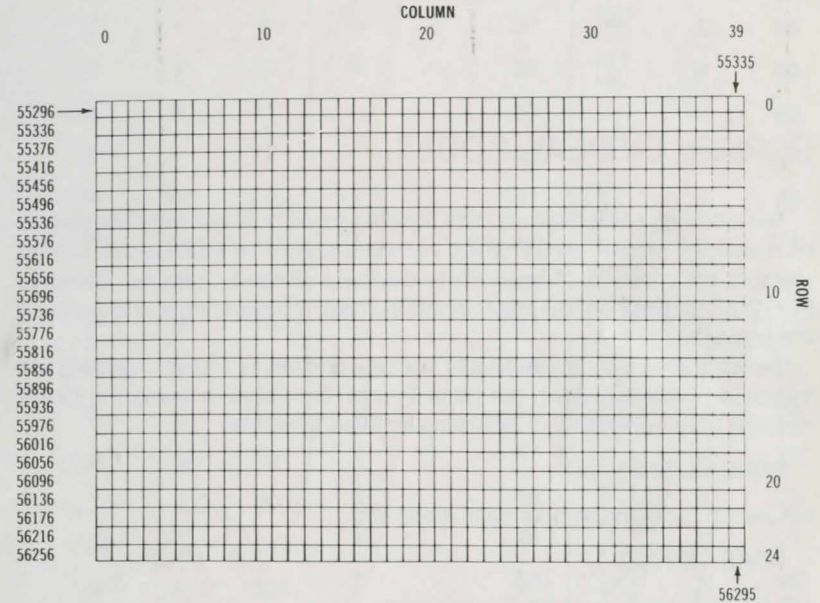


The Screen Map is POKEd with a value from the Screen Code per Appendix D:

POKE 1024,13

will display the letter M in the upper left corner of the monitor.

COLOR MEMORY MAP



The color map is POKEd with a color value; this changes the character's color. Thus

POKE 55296,1

will change the letter M inserted above from light blue to white.

Color Codes

- | | |
|----------|----------------|
| 0 Black | 8 Orange |
| 1 White | 9 Brown |
| 2 Red | 10 Light Red |
| 3 Cyan | 11 Dark Gray |
| 4 Purple | 12 Medium Gray |
| 5 Green | 13 Light Green |
| 6 Blue | 14 Light Blue |
| 7 Yellow | 15 Light Gray |

Border Control Memory 53280

Background Control Memory 53281

APPENDIX D

• SCREEN DISPLAY CODES

The following chart displays all of the characters built into the Commodore 64 character sets. It shows which numbers should be POKEd into screen memory (locations 1024 to 2023) to display a desired character. Likewise PEEKing that position will result in this number representing a character on the screen.

Two character sets are available but only one set at a time. The sets are switched by holding down the [SHIFT] and [C=] keys or using the CHR\$ function from Appendix E or by poking a memory location:

POKE 53272,21

will switch to graphics/upper-case mode and

POKE 53272,23






























will switch to upper-case/lower-case mode.

Any character on the chart may be displayed in reverse video by adding 128 to the values shown.

Example: to display a solid circle at location 1504, POKE the code for the circle (81) into that location:

POKE 1504, 81

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	C	c	3	F	f	6
A	a	1	D	d	4	G	g	7
B	b	2	E	e	5	H	h	8

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
I	i	9	%		37		A	65
J	j	10	&		38		B	66
K	k	11	'		39		C	67
L	l	12	(40		D	68
M	m	13)		41		E	69
N	n	14	*		42		F	70
O	o	15	+		43		G	71
P	p	16	,		44		H	72
Q	q	17	-		45		I	73
R	r	18	.		46		J	74
S	s	19	/		47		K	75
T	t	20	0		48		L	76
U	u	21	1		49		M	77
V	v	22	2		50		N	78
W	w	23	3		51		O	79
X	x	24	4		52		P	80
Y	y	25	5		53		Q	81
Z	z	26	6		54		R	82
[27	7		55		S	83
£		28	8		56		T	84
]		29	9		57		U	85
↑		30	:		58		V	86
←		31	;		59		W	87
SPACE		32	<		60		X	88
!		33	=		61		Y	89
"		34	>		62		Z	90
#		35	?		63			91
\$		36			64			92

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
		93			105			117
		94			106			118
		95			107			119
SPACE		96			108			120
		97			109			121
		98			110		<input checked="" type="checkbox"/>	122
		99			111			123
		100			112			124
		101			113			125
		102			114			126
		103			115			127
		104			116			

Codes from 128-255 are reversed images of codes 0-127.

APPENDIX E

• ASCII AND CHR\$ CODES

Each character has a corresponding ASCII code obtained by typing:

PRINT ASC("x")

where X is any character you select, resulting in the following table. Typing

PRINT CHR\$(n)

where n is the ASCII code number from the table will print the corresponding character. Thus:

PRINT CHR\$(65)

will display the letter A where the cursor happens to be and

PRINT CHR\$(147)

will clear the screen.

PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$
	0		17	"	34	3	51
	1		18	#	35	4	52
	2		19	\$	36	5	53
	3		20	%	37	6	54
	4		21	&	38	7	55
	5		22	.	39	8	56
	6		23	(40	9	57
	7		24)	41	:	58
DISABLES	8		25	*	42	;	59
ENABLES	9		26	+	43		60
	10		27	,	44	=	61

PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$
	11	RED	28	-	45	>	62
	12	CRSR	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
D	68	♠	97	☐	126	Lt. Gray	155
E	69	☐	98	☐	127	PIR	156
F	70	☐	99		128	CRSR	157
G	71	☐	100	Orange	129	YEL	158
H	72	☐	101		130	CYN	159
I	73	☐	102		131	SPACE	160
J	74	☐	103		132	☐	161
K	75	☐	104	f1	133	☐	162
L	76	☐	105	f3	134	☐	163
M	77	☐	106	f5	135	☐	164
N	78	☐	107	f7	136	☐	165
O	79	☐	108	f2	137	☐	166
P	80	☐	109	f4	138	☐	167
Q	81	☐	110	f6	139	☐	168
R	82	☐	111	f8	140	☐	169
S	83	☐	112	SHIFT RETURN	141	☐	170
T	84	☐	113	SWITCH TO UPPER CASE	142	☐	171
U	85	☐	114		143	☐	172
V	86	☐	115	BLK	144	☐	173
W	87	☐	116	CRSR	145	☐	174
X	88	☐	117	RVS OFF	146	☐	175
Y	89	☐	118	CLR HOME	147	☐	176

PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$
Z	90	☐	119	INST DEL	148	☐	177
[91	☐	120	Brown	149	☐	178
£	92	☐	121	Lt. Red	150	☐	179
]	93	☐	122	Dk. Gray	151	☐	180
↑	94	☐	123	Gray	152	☐	181
←	95	☐	124	Lt. Green	153	☐	182
☐	96	☐	125	Lt. Blue	154	☐	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

APPENDIX F

• SPRITE REGISTER MAP

Register # Dec	Hex	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
0	0	S0X7							S0X0	SPRITE 0 X Component
1	1	S0Y7							S0Y0	SPRITE 0 Y Component
2	2	S1X7							S1X0	SPRITE 1 X
3	3	S1Y7							S1Y0	SPRITE 1 Y
4	4	S2X7							S2X0	SPRITE 2 X
5	5	S2Y7							S2Y0	SPRITE 2 Y
6	6	S3X7							S3X0	SPRITE 3 X
7	7	S3Y7							S3Y0	SPRITE 3 Y
8	8	S4X7							S4X0	SPRITE 4 X
9	9	S4Y7							S4Y0	SPRITE 4 Y
10	A	S5X7							S5X0	SPRITE 5 X
11	B	S5Y7							S5Y0	SPRITE 5 Y
12	C	S6X7							S6X0	SPRITE 6 X
13	D	S6Y7							S6Y0	SPRITE 6 Y
14	E	S7X7							S7X0	SPRITE 7 X Component
15	F	S7Y7							S7Y0	SPRITE 7 Y Component
16	10	S7X8	S6X8	S5X8	S4X8	S3X8	S2X8	S1X8	S0X8	MSB of X COORD.
17	11	RC8	ECM	BMM	BLNK	RSEL	YSCL2	YSCL1	YSCL0	Y SCROLL MODE
18	12	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	RASTER
19	13	LPX7							LPX0	LIGHT PEN X
20	14	LPY7							LPY0	LIGHT PEN Y

Register # Dec	Hex	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
21	15	SE7							SE0	SPRITE ENABLE (ON/OFF)
22	16	N.C.	N.C.	RST	MCM	CSEL	XSCCL2	XSCCL1	XSCCL0	X SCROLL MODE
23	17	SEXY7							SEXY0	SPRITE EXPAND Y
24	18	VS13	VS12	VS11	VS10	CB13	CB12	CB11	N.C.	SCREEN Character Memory
25	19	IRQ	N.C.	N.C.	N.C.	LPIRQ	ISSC	ISBC	RIRQ	Interrupt Requests
26	1A	N.C.	N.C.	N.C.	N.C.	MLPI	MISSC	MISBC	MRIRQ	Interrupt Request MASKS
27	1B	BSP7							BSP0	Background-Sprite PRIORITY
28	1C	SCM7							SCM0	MULTICOLOR SPRITE SELECT
29	1D	SEXX7							SEXX0	SPRITE EXPAND X
30	1E	SSC7							SSC0	Sprite-Sprite COLLISION
31	1F	SBC7							SBC0	Sprite-Background COLLISION

Register # Dec	Hex	Color
32	20	BORDER COLOR
33	21	BACKGROUND COLOR 0
34	22	BACKGROUND COLOR 1
35	23	BACKGROUND COLOR 2
36	24	BACKGROUND COLOR 3
37	25	SPRITE MULTICOLOR 0
38	26	SPRITE MULTICOLOR 1

Register # Dec	Hex	Color
39	27	SPRITE 0 COLOR
40	28	SPRITE 1 COLOR
41	29	SPRITE 2 COLOR
42	2A	SPRITE 3 COLOR
43	2B	SPRITE 4 COLOR
44	2C	SPRITE 5 COLOR
45	2D	SPRITE 6 COLOR
46	2E	SPRITE 7 COLOR

APPENDIX G

• MUSIC NOTE TABLE

Note values are POKEd into two memory locations 54272 and 54273, also known as registers or switches 0 and 1 respectively.

POKE the value N1 into Register 1 and the value N2 into Register 0.

The list below covers three octaves of notes for Bass and Treble Clef. For the full list of nine octaves, see the Programmer's Reference Guide.

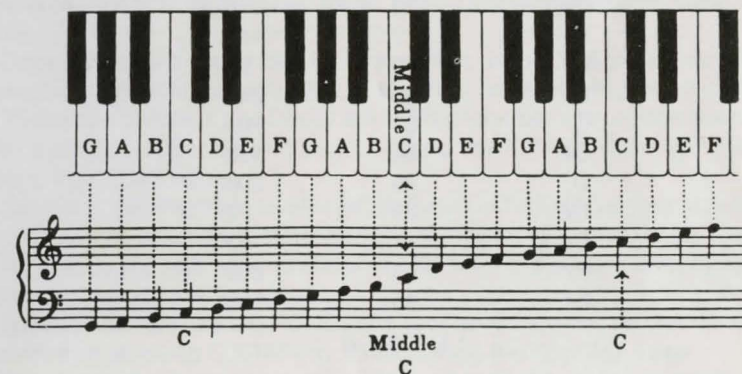


TABLE OF NOTE VALUES

NOTE	N1	N2
G	6	36
G#	6	130
A	6	228
A#	7	77
B	7	189
C	8	50
C#	8	175
D	9	51
D#	9	191
E	10	84

F	10	241
F#	11	152
G	12	73
G#	13	4
A	13	201
A#	14	156
B	15	122
*C	16	101
C#	17	96
D	18	104
D#	19	128
E	20	169
F	21	227
F#	23	49
G	24	146
G#	26	8
A	27	148
A#	29	57
B	30	245
C	32	204
C#	34	192
D	36	208
D#	39	1
E	41	83
F	43	200
F#	46	99

* MIDDLE C

APPENDIX H

• SOUND CONTROL SETTINGS

Each sound parameter is POKEd into a register of the specialized sound generating chip. Each register or switch is a memory location (called byte address) starting with 54272.

Each sound has a characteristic ADSR consisting of the following four parameters: Attack, Decay, Sustain, Release.

Attack is the rate sound rises to maximum volume. It can vary from a 2-millisecond cycle to an 8-second cycle. The corresponding register value is 0 to 15.

Decay is the rate sound falls from maximum volume to sustain level. This varies from a 6-millisecond cycle to 24 seconds, corresponding with 0 to 15.

The values of Attack and Decay are POKEd together into register (switch) 5 by a single number derived by multiplying the ATTACK value by 16 and adding the DECAY value.

Sustain is the amplitude level at which the sound is held, varying from 0% to 100% of maximum level corresponding to register values of 0 to 15.

Release is the rate at which volume falls from the sustain level to zero; similar in timing to the decay rate.

Sustain and Release are POKEd into register 6 together as one number derived by multiplying SUSTAIN by 16 and adding the RELEASE value.

Waveform is the shape of the sound wave produced. The waveforms called Triangle, Sawtooth and Pulse are related to the musical instrument. Noise is a randomized waveform. Only specific register values will activate this characteristic of sound.

Pulse is the tonal quality of the Pulse waveform. Thus, whenever register (switch) 4 is activated with a 65, a value other than zero must be POKEd into either switch 2 or 3 for the Pulse Rate.

Frequency is the vibratory level of sound which distinguishes one note from another. Concert A is 440 cycles per second. Switches 0 and 1 are required to define the frequency. 256 times the value in Register 1 plus the value of Register 0 is the sound generator's oscillator frequency. This is directly proportional to the sound frequency.

Below is a table of values which can be POKEd into these registers. The actual memory location is 54272 plus the register number.

Register			Description	Range of Values
	Voice 1	Voice 2	Voice 3	
0	7	14	frequency	0 to 255
1	8	15	frequency	0 to 255
2	9	16	pulse	0 to 255
3	10	17	pulse	0 to 15
4	11	18	Waveform	16,32,64,128 17,33,65,129
5	12	19	Attack/Decay	0 to 255
6	13	20	Sustain/Release	0-255
	All voices			
	21		Filter-low cutoff	0 to 7
	22		Filter-high cutoff	0 to 255
	23		Resonance	16,32,64,128 or any sum
	23		Filter switch/voice	1,2 or 4
	24		Volume	0 to 15

APPENDIX I

• DERIVED TRIGONOMETRIC 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	$ARCCOS(X) = -ATN(X/SQR(-X*X+1)) + \pi/2$
INVERSE SECANT	$ARCSEC(X) = ATN(X/SQR(X*X-1))$
INVERSE COSECANT	$ARCCSC(X) = ATN(X/SQR(X*X-1)) + (SGN(X) - 1)*\pi/2$
INVERSE COTANGENT	$ARCOT(X) = ATN(X) + \pi/2$
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	$ARCCOTH(X) = LOG((X+1)/(x-1))/2$

APPENDIX J

• ABBREVIATIONS OF BASIC KEYWORDS

As a time-saver when typing programs and commands, you can abbreviate most keywords. The abbreviation for PRINT is a question mark. The abbreviations for other words are made by typing the first one or two letters of the word, followed by the SHIFTed next letter of the word. When used in a program line, the keyword will LIST in the full form.

Com-mand	Abbrevi-ation	Looks like this on screen	Com-mand	Abbrevi-ation	Looks like this on screen
ABS	A SHIFT B	A	END	E SHIFT N	E
AND	A SHIFT N	A	EXP	E SHIFT X	E
ASC	A SHIFT S	A	FN	NONE	FN
ATN	A SHIFT T	A	FOR	F SHIFT O	F
CHR\$	C SHIFT H	C	FRE	F SHIFT R	F
CLOSE	CL SHIFT O	CL	GET	G SHIFT E	G
CLR	C SHIFT L	C	GET#	NONE	GET#
CMD	C SHIFT M	C	GOSUB	GO SHIFT S	GO
CONT	C SHIFT O	C	GOTO	G SHIFT O	G
COS	NONE	COS	IF	NONE	IF
DATA	D SHIFT A	D	INPUT	NONE	INPUT
DEF	D SHIFT E	D	INPUT#	I SHIFT N	I
DIM	D SHIFT I	D	INT	NONE	INT

Com-mand	Abbrevi-ation	Looks like this on screen	Com-mand	Abbrevi-ation	Looks like this on screen
LEFT\$	LE SHIFT F	LE	RIGHT\$	R SHIFT I	R
LEN	NONE	LEN	RND	R SHIFT N	R
LET	L SHIFT E	L	RUN	R SHIFT U	R
LIST	L SHIFT I	L	SAVE	S SHIFT A	S
LOAD	L SHIFT O	L	SGN	S SHIFT G	S
LOG	NONE	LOG	SIN	S SHIFT I	S
MID\$	M SHIFT I	M	SPC(S SHIFT P	S
NEW	NONE	NEW	SQR	S SHIFT Q	S
NEXT	N SHIFT E	N	STATUS	ST	ST
NOT	N SHIFT O	N	STEP	ST SHIFT E	ST
ON	NONE	ON	STOP	S SHIFT T	S
OPEN	O SHIFT P	O	STR\$	ST SHIFT R	ST
OR	NONE	OR	SYS	S SHIFT Y	S
PEEK	P SHIFT E	P	TAB(T SHIFT A	T
POKE	P SHIFT O	P	TAN	NONE	TAN
POS	NONE	POS	THEN	T SHIFT H	T
PRINT	?	?	TIME	TI	TI
PRINT#	P SHIFT R	P	TIMES\$	TI\$	TI\$
READ	R SHIFT E	R	USR	U SHIFT S	U
REM	NONE	REM	VAL	V SHIFT A	V
RESTORE	RE SHIFT S	RE	VERIFY	V SHIFT E	V
RETURN	RE SHIFT T	RE	WAIT	W SHIFT A	W

NOTE: See the BASIC Encyclopedia (page 159) for details on specific commands.

APPENDIX K

• BASIC CONVERSIONS

If you have programs written in a BASIC other than Commodore BASIC, some minor adjustments may be necessary before running them on the C64. Here are some hints to make the conversion easier.

String Dimensions

Modify all statements that are used to declare the dimension of strings. Commodore BASIC does not require a string length dimension—only the quantity of variables. A statement such as DIM A\$(J,K), which dimensions a string array for J elements (single array), each of length K, should be converted to DIM A\$(J).

String Concatenation

Replace the ampersand or comma some BASICs use to concatenate strings with the plus sign.

Substrings

In Commodore 64 BASIC, the MID\$, RIGHT\$ and LEFT\$ functions are used to take substrings of strings. Forms such as A\$(J) to access the Jth character of the nondimensioned string A\$ must be changed to MID\$(A\$,J,1).

Forms such as A\$(J,K) to access the Kth character of the single dimensioned array A\$(J) must be changed to MID\$(A\$(J),K,1).

Slicers in the form A\$(J TO K) which take a substring of the variable from the Jth character to the Kth character must be changed to MID\$(A\$,J,K-J+1).

LET statements

LET statements are accepted by Commodore BASIC and need not be changed.

PAUSE

To create a time delay in Commodore BASIC replace PAUSE statements with a FOR . . . NEXT loop.

PRINT AT

To achieve an equivalent in the C64 BASIC, replace PRINT AT X, Y with:

```
POKE 782,X:POKE 781, Y:SYS 65520
```

Multiple Statements

Some BASICs use a backslash (/) to separate multiple statements on a line. Commodore BASIC requires the colon (:).

Multiple Assignments

To set more than one variable to the same constant, such as LET B = C = 0 must be converted to separate statements with colons or on individual lines: B = 0:C = 0.

MAT functions

Programs using MAT functions available on some BASICs must be rewritten using FOR . . . NEXT loops to execute properly.

Random Numbers

Other BASICs may apply the random function differently. Whereas INT(RND*6)+1 is used to obtain the six numbers of a die, use INT(RND(0)*6) + 1 in Commodore BASIC.

APPENDIX L

• RECOMMENDED READING LIST

Below is a sample list of books available from the major publishing houses as well as the Commodore Library.

Beginning BASIC

Brady	Taking Off with BASIC on the C64
Hayden	Basic Commodore 64 BASIC
	I Speak BASIC to My C64
Prentice Hall	Programming Your Commodore 64 in BASIC
Sams	Learn BASIC Programming in 14 Days on Your C64
	Commodore 64 Starter Book
Sybex	Your First Commodore 64 Program
	The Easy Guide to Your C64

Just for Kids

Creative	Computers for Kids: C64
Datamost	Kids and the Commodore 64
Sams	Commodore 64 for Kids from 8 to 80
QUE	Timlost
Sybex	Power Up! Kids' Guide to the C64
Trillium	Kids Working with Computers: C64

General Applications

Brady	101 Uses for the C64
Commodore	Business Applications on the C64
	Artificial Intelligence on the C64
	Commodore 64 Adventures
Compute	Creating Arcade Games on the C64
Creative	The Working Commodore 64
Hayden	Stimulating Simulations for the C64
Reston	C64 Data Files: A BASIC Tutorial
Sams	Commodore 64 BASIC Programs
TAB	Using and Programming the C64

Education

Byte Commodore	Learning with Commodore LOGO Mathematics on the C64 Programming for Education on the C64
Sybex	Parents, Kids, and the C64

Computer Controllers

Birkhauser Prentice Hall Sybex	Your Computer Butler Easy Interfacing Projects for the C64 The Commodore 64 Connection
--------------------------------------	--

Graphics

Arrays Brady CBS Commodore Prentice Hall Sams Sybex TAB	C64 Color Graphics: An Advanced Guide C64 Graphics: Activities Handbook Color Graphics for the C64 Graphic Art on the C64 Sprite Graphics for the C64 C64 Graphics and Sounds Graphics Guide to the C64 C64 Graphics and Sound Programming
--	---

Music

Birkhauser Prentice Hall	The C64 Music Book Music and Sound for the C64
-----------------------------	---

Telecommunications

Osborne/McGraw	C64 Telecommunications
----------------	------------------------

References

Arrays Brady Commodore	The C64 User's Encyclopedia Introduction to Assembly Language for the C64 Advanced Programming Techniques on the C64 Programmer's Reference Guide 1541 Disk Companion
Compute Computext Reston Osborne/McGraw	Compute's First Book of 64 C64 BASIC Guide Master Memory Map: C64 Your Commodore 64

BASIC 2.0 ENCYCLOPEDIA

INTRODUCTION _____	161
BASIC COMMANDS _____	162
BASIC STATEMENTS _____	168
BASIC FUNCTIONS _____	181
VARIABLES AND OPERATORS _____	186

• INTRODUCTION

In this manual, you've seen an assortment of exercises using the BASIC language that give you a feel for computer programming and some of the vocabulary involved. This encyclopedia gives a list of the rules (syntax) and terms of the BASIC 2.0 language, along with a concise description of each. Experiment with these commands, and remember—you can't damage your Commodore 64 by typing in programs, and the best way to learn computing is by computing.

The encyclopedia provides formats, brief explanations and examples of the BASIC 2.0 commands and statements. It is not intended to teach BASIC. If you are interested in learning BASIC, Appendix L 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 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 can use many statements as commands by using them in direct mode (i.e., without line numbers).

The BASIC Encyclopedia is organized as follows:

- **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 more complete explanation of BASIC 2.0 commands is provided in the Commodore 64 Programmer's Reference Guide, available from your Commodore dealer or your local bookstore.

• BASIC COMMANDS

CONT (Continue)

This command is used to restart the execution of a program which has been stopped by using the STOP key, a STOP statement or an END statement within the program. The program will restart at the exact place it left off.

CONT will not work if you have changed or added lines to the program (or even just moved the cursor), or if the program halted due to an error, or if you caused an error before trying to restart the program. In these cases you will get a CAN'T CONTINUE ERROR.

COPY

On a dual disk drive (4040), COPY a disk file from one drive (the source file) to the other. On a single disk drive (1541), COPY a file on the same disk under a different filename. You must open the disk file before COPYING as follows:

OPEN file number, device number, channel number

Example:

OPEN 15,8,15 : REM file 15, device 8, channel 15

The COPY command format is as follows:

PRINT#15,"COPY[drive number]:new file = [drive number]:old file"

For example:

PRINT#15,"COPY0:NOON = 1:NIGHT"

copies the file named "NIGHT" from drive 1 to drive 0 and renames it "NOON".

PRINT#15,"COPY0:STUFF = 1:STUFF"

copies the file named STUFF from drive 1 to drive 0.

PRINT#15,"COPY0:DOGS = 0:CATS"

copies the file named "CATS" onto the same disk in a single drive and renames it "DOGS"

LIST

LIST [first line] — [last line]

The LIST command displays the lines of a BASIC program in the Commodore 64 memory.

The LIST command has five options:

1. Type the word LIST to display the entire program in memory. Slow down the LISTing by holding down the CTRL key. Stop the LISTing by pressing the RUN/STOP key.

Example: LIST (LISTs the entire program).

2. Type the word LIST and follow it with a line number to display that specified program line.

Example: LIST 10 (LISTs only line 10).

3. Type the word LIST and follow it with a line number and a dash to display the program starting at the specified line number.

Example: LIST 100- (LISTs from line 100 to the end of the program).

4. Type the word LIST and follow it with a dash and a line number to display the program from the beginning to the specified line number.

Example: LIST-100 (LISTs the program from the start up to line 100).

5. Type the word LIST, follow it with a line number, a dash and another line number to display the program from the first specified line number to the second specified line number.

Example: LIST 10-200 (LISTs lines 10 through 200).

LOAD

The LOAD command fills the computer's memory with a program stored on diskette or cassette tape. The format for the load command is as follows:

LOAD "filename",[device number],[secondary address]

The filename is the name of the program you want to load. The device number for cassette is 1, the device number for a disk drive is 8. The secondary address is number 1 and is only specified when you want to LOAD a machine language program into a specific Commodore 64 memory location from which it was SAVED.

You have four options in which to LOAD a program from CASSETTE tape.

1. Type LOAD and press RETURN. The computer responds by displaying:

PRESS PLAY ON TAPE

Press the play button on the Datassette. The screen turns off and the computer searches for the first program on the cassette tape. Press the **C** key to LOAD the program or the spacebar to search for the next program on the cassette.

Example: LOAD **RETURN**

2. Type LOAD followed by a program name within quotation marks. The same sequence of events happens as above (option 1) except this directly LOADs a specified program name from the cassette tape.

Example: LOAD "Program Name" **RETURN**

3. Type LOAD followed by a program name within quotation marks, a comma and the number 1. This is the same as number 2 except you specify the device number 1 as the Datassette. If a device number is not specified, it defaults to device number 1. When using cassette tape, you do not have to specify the device number, it is optional.

Example: LOAD "Program Name",1 **RETURN**

4. Type LOAD followed by a program name within quotes, a comma, the number 1, a second comma and the number 1 again. This LOADs a specified program name from device 1 (Datassette) into the Commodore 64 memory location from which it was SAVEd. If the secondary address 1 is specified, the program name and device number must be specified.

Example: LOAD "Program Name",1,1 **RETURN**

You have two options in which to LOAD a program from DISKETTE:

1. Type LOAD followed by a program name within quotation marks, a comma and the number 8. This LOADs the specified program from diskette.

Example: LOAD "Program Name",8

2. Type LOAD followed by a program name within quotes, a comma, the number 8, a second comma and the number 1. This LOADs the specified machine language program name into a predetermined Commodore 64 memory location from which it was SAVEd.

Example: LOAD "Program Name",8,1 **RETURN**

In these examples, **RETURN** indicates that you must press the RETURN key after the given command.

Arguments appearing within brackets are optional.

NEW

BE CAREFUL WHEN YOU USE THIS COMMAND. This command erases the entire program in memory, and also clears out any variables that may have been used. Unless the program was SAVEd, it is lost.

The NEW command can also be used as a BASIC program statement. When the program reaches this line, the program is erased. This is useful if you want to leave everything neat when the program is done.

RUN

RUN [line number]

The RUN command executes a program in the Commodore 64's memory. If a line number is specified following the RUN command, the computer starts RUNNING the program at the specified line number. RUN may be used within a program.

Examples:

RUN Executes the program from the beginning.

RUN 100 Starts executing the program at line 100.

SAVE

SAVE ["filename" [,device number[,EOTflag]]]

The SAVE command stores a program currently in the computer's memory onto a disk or cassette tape.

You have three options in SAVEing programs on CASSETTE

1. Type SAVE and press RETURN. The Commodore 64 responds by displaying the message:

PRESS RECORD & PLAY ON TAPE

Press the PLAY and RECORD buttons on the Datassette. The Commodore 64 SAVEs the program in memory starting at the current position of the cassette tape. Make sure you do not have an important program at that tape position because the C64 SAVEs the current program on top of the original one, and that one is lost.

Example: SAVE **RETURN**

2. Type SAVE followed by a program name in quotation marks. This SAVES the specified program onto the cassette tape at the current position. The same conditions in option 1 apply.

Example: SAVE "Program Name" **RETURN**

3. Type SAVE followed by a program name in quotes, a comma and the number 1. This is the same as option 2 except you specify the device number for the Datassette.

Example: SAVE "Program Name",1 **RETURN**

To SAVE to DISK, type SAVE followed by a program name within quotes, a comma, and the number 8. This saves the contents of the Commodore 64 memory onto the disk. The diskette must be formatted before you can SAVE programs on it. See the DISK NEW command.

Example: SAVE "Program Name",8 **RETURN**

SCRATCH

Deletes a file from the disk directory. Use this command to erase unwanted files and to create more storage space on the disk. You must first open the disk command channel (secondary address 15) before scratching any files as follows:

OPEN file number, device number, secondary address

Example: OPEN 15,8,15 **RETURN**

The format for the SCRATCH command is as follows:

PRINT# file number,"SCRATCH [drive number]:filename"

Example: PRINT#15,"SCRATCH0:MY BACK" **RETURN**

You can abbreviate the SCRATCH command as follows:

PRINT#15,"S0:MY BACK" **RETURN**

The above examples erase the file named "MY BACK" from the disk in drive 0.

VERIFY

VERIFY "filename",[device#],[secondary address]

The VERIFY command compares the program on tape or disk with the one in memory. If the programs are identical, the Commodore 64 responds with "OK". If the two versions of the program differ, a VERIFY ERROR results. This command makes sure the program is SAVED correctly to tape or disk.

You have three ways to VERIFY a program:

1. Type VERIFY and press RETURN. This verifies the program at the current position of the cassette tape.

Example: VERIFY **RETURN**

2. Type VERIFY followed by a program name within quotation marks. This verifies the specified program name on the cassette tape.

Example: VERIFY "Program Name" **RETURN**

3. Type VERIFY followed by a program name within quotation marks, a comma and a device number. The device number can be either 1 for the Datassette or 8 for the disk drive. This command verifies the specified program name on the designated device (disk drive or tape).

Examples: VERIFY "Program Name",8 **RETURN** (Verifies program on disk)

VERIFY "Program Name",1 **RETURN** (Verifies program on cassette tape)

• BASIC STATEMENTS

CLOSE

CLOSE file number

This statement closes a previously opened file. The number following the word CLOSE is the file number to be closed.

Example: CLOSE 2 **RETURN** (Closes file 2)

CLR

CLR

This statement clears the value of any variables in memory, but leaves the program itself intact.

Example: CLR **RETURN**

CMD

CMD file number [,string]

CMD sends output which normally goes to the screen, to the specified file corresponding to another device. This can be a printer file or a data file on tape or disk. The file must be OPENed first.

Example: OPEN 4,4 **RETURN** (Open file 4 on device 4—the printer)
CMD 4 **RETURN**
LIST **RETURN** (Lists the program in memory on printer)
PRINT#4 **RETURN** (Close channel to printer)
CLOSE 4 **RETURN** (Close file 4)

You can specify an optional string in the CMD command. Any characters in the specified string are output to the device.

Example:

OPEN 4,4, (Open file 4 on device 4)
CMD 4, "Anybody out there?" (Send screen output to printer)
PRINT#4 (Close channel to printer)
CLOSE 4 (Close file 4)

DATA

DATA constant list

This statement is followed by a list of items to be used by READ statements. Items may be numeric values or text strings, and are separated by commas. String items need not be inside quote marks unless they contain a space, a colon, or a comma. If two commas have nothing between them, the value will be READ as a zero for a number, or as an empty string.

DATA 12, 14.5, "HELLO, MOM", 3.14, PART 1

DEF FN

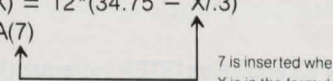
DEF FN function name (numeric variable)

This command allows you to define a complex calculation as a function with a short name. In the case of a long formula that is used many times within the program, this can save time and space.

The function name will be FN and any legal variable name (1 or 2 characters long). First you must define the function using the statement DEF followed by the function name. Following the name is a set of parentheses enclosing a numeric variable. The actual formula that you want to define then follows, with the variable in the proper spot. You can then "call" the formula, substituting any number for the variable.

Example:

```
10 DEF FNA(X) = 12*(34.75 - X/.3)
20 PRINT FNA(7)
```



For this example, the result would be 137.

DIM

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

Before you can use an array, you must first execute a DIM statement to establish the DIMensions of the array. If the array has less than 11 elements you do not need a DIM statement since the Commodore 64 automatically DIMensions each variable to 10 elements.

The DIM statement is followed by the name of the array, which may be any legal variable name. The array name is followed by an integer enclosed in parentheses. The integer specifies the number of elements in each dimension. You may use any number of dimensions, but keep in mind that each

array element uses memory. To figure out the total number of array elements in each array, multiply the number of elements in each dimension of the array.

NOTE: Integer arrays use only 40% of the space of floating point arrays.

Example:

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

Array A\$ has 41 elements

Array B7 has 16 elements

Array CC% has 125 elements

You can dimension more than one array in a DIM statement by separating the arrays with commas. If you execute a DIM statement more than once for each array within a program, a REDIM'D ARRAY ERROR message is displayed. It is good programming practice to place DIM statements near the beginning of the program.

END

When a program encounters an END statement, the program stops running immediately. You may use the CONT statement to re-start the program at the statement following the END statement.

Example: END

FOR... TO... STEP

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

The FOR... TO statement and the NEXT statement are used together to form a program loop—a sequence of instructions that are executed repetitively. The loop variable acts as a counter and is added to or subtracted from during the FOR/NEXT loop. The start value is the beginning count of the loop variable and the end value is the finishing count of the loop variable. The STEP portion of the FOR... TO statement is specified if you want to increment the loop variable by a value greater than 1. You must specify the STEP portion if the start value is greater than the end value and you are counting backwards through a program loop.

The logic of the FOR... TO 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 the loop variable is less than or equal to the end value, the statement immediately

following the FOR... TO statement is executed. If the loop variable is greater than the end of loop value, the statement directly following the NEXT statement is executed. If the loop variable value is negative, the loop is executed until it becomes less than the end value. See the NEXT STATEMENT.

Example:

```
10 FOR L = 1 TO 20
20 PRINT L
30 NEXT L
40 PRINT "BLACKJACK! L = "L
```

This program prints the numbers from one to twenty on the screen, followed by the message BLACKJACK! L = 21.

You can set up loops inside one another. This is known as nesting loops. Nest loops so that the inner FOR... TO statement and the corresponding NEXT statement are both in between the outer FOR... TO statement and corresponding NEXT statement. Remember that in nesting, the last loop to start is the first one to end.

Example: Nested Loops

```
10 FOR L = 1 TO 20
20 PRINT L
30 FOR I = 1 TO 10
40 PRINT I
50 NEXT I
60 NEXT L
```

GET

GET variable list

The GET statement inputs data from the keyboard, one character at a time. When the computer accepts a character from the keyboard, it is assigned to the variable name specified in the GET command. If no character is typed, a null (empty) character is assigned, and the program continues without waiting for a key. For this reason, the GET statement is usually used along with an IF... THEN statement to check for a null character (""),. There is no need to press the RETURN key after you type a character for a GET statement.

The word GET is followed by a variable name, usually a string variable. If a numeric variable is used and a key other than a number is pressed, the program stops and a TYPE MISMATCH ERROR occurs. If a string variable is used, the GET statement accepts any character entered from the keyboard. The GET statement can only be used within a program.

Example:

```
10 GET A$:IF A$ = "" THEN 10: REM Wait for a key to be pressed to
continue.
```

GET#

GET# file number,variable list

The GET# statement inputs data from a previously opened file on a peripheral device, one character at a time. The character is assigned to the specified variable name. This command can only be executed within a program.

Example:

```
10 GET#1,A$
```

GOSUB

GOSUB line number

The GOSUB statement calls a separate and independent program segment called a subroutine. When a GOSUB statement is encountered in a program, the computer jumps to a subroutine, and executes it. When a RETURN statement is encountered in the subroutine, the computer jumps back to the instruction directly following the GOSUB statement in the main program.

Example:

```
10 GOSUB 800: REM Jump to the subroutine at line 800 and execute it.
```

```
:
```

```
800 PRINT "HI THERE"
```

```
810 RETURN
```

GOTO

GOTO line number

The GOTO statement jumps to and executes the instructions starting at the specified line number. When used in direct mode the GOTO statement starts execution of a program at the specified line number.

Example:

```
10 PRINT "REPETITION IS THE MOTHER OF LEARNING"
20 GOTO 10
```

The GOTO in line 20 causes the program to be run continuously, until the RUN/STOP key is pressed.

IF... THEN

IF expression THEN statement(s)

The IF... THEN statement evaluates a condition and executes one of two possible program segments, depending on whether the condition is true or false. If the expression is true, the BASIC statement directly following the word THEN is executed. If the expression is false, the program continues to the program line directly following the line containing the IF statement. The evaluated expression is usually a mathematical expression containing relational or logical operators (=, <, >, <=, >=, <>, AND, OR, NOT). The IF... THEN statement is the computer's way of making a decision.

Example:

```
50 IF X = 0 THEN PRINT "OK"
60 PRINT "REST OF PROGRAM"
```

Evaluates the value of X. If X equals 0, the computer PRINTS "OK" and continues with line 60. If X does not equal 0, the part following the word THEN is skipped and the program continues with line 60.

INPUT

INPUT["prompt string";] variable list

The INPUT statement accepts characters from the keyboard and stores them in the specified variable name. The program pauses, displays a question mark (?) on the screen, and waits for you to type a response and press the RETURN key. The maximum amount of characters you can INPUT is 77.

The word INPUT is followed by a variable name or list of variable names separated by commas. You can include a message enclosed in quotes called a prompt before the list of input variables. If the prompt is present, there must be a semicolon (;) after the closing quote. When more than one variable is INPUT, separate them with commas. If you don't, the computer asks for the remaining input variable values by displaying two question marks (??) on

the screen. 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 "Number of Ice Cream Cones";A$
```

INPUT#

INPUT# file number, variable list

INPUT# works like INPUT except it takes data from a previously OPENed file or device instead of the keyboard. No prompt string is allowed. This command can only be used in program mode.

Example:

```
10 INPUT#2, A$, C, D$  
(Inputs three variable values from file 2.)
```

LET

[LET]variable = expression

LET is rarely used in programs, since it is not necessary. Whenever a variable is assigned a value, LET is always implied. The variable name which receives a value is on the left side of the equal sign, and the value itself is on the right side.

Example:

```
10 LET A = 5  
20 LET B = 6  
LET is specified (but not necessary) in lines 10 and 20.  
LET is implied in lines 30 and 40.  
30 C = A * B + 3  
40 D$ = "HELLO"
```

NEXT

NEXT [index variable, . . . ,variable]

The NEXT statement completes a FOR . . . NEXT loop. When the computer encounters a NEXT statement, the program goes back to the corresponding FOR . . . TO statement and checks the index variable. If the index variable is less than or equal to the limit of the loop in the FOR . . . TO statement,

the loop cycles again. If the index variable is greater than the limit of the loop, the program continues with the statement directly following the NEXT statement.

Specifying a variable is optional in a NEXT statement, though it may be followed by a variable name or a list of variable names separated by commas. If there are no names listed, the last loop started is the one incremented. If the variables are specified, they are incremented 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 statement makes the GOTO and GOSUB statements into conditional statements like the IF . . . THEN statement. The word ON is followed by a mathematical expression and either a GOSUB or GOTO statement and a list of line numbers. The result of the expression determines which line number or subroutine is executed. If the result of the expression is 1, the first line in the list is executed. If the result is 10, the tenth line number is executed, and so on. The result of the expression should not exceed the number of line numbers in the list. If the result is larger than the number of line numbers in the list or zero, the program continues with the line directly following the ON statement. If the number is negative, an ILLEGAL QUANTITY ERROR results.

Example:

```
10 INPUT X:IF X<0 THEN 10           When X = 1, ON sends control to  
                                     the first line number in the list  
  
20 ON X GOTO 50, 30, 30, 70         When X = 2, ON sends control to  
                                     the second line (30), etc.  
  
25 PRINT "FELL THROUGH":GOTO 10  
30 PRINT "TOO HIGH":GOTO 10  
50 PRINT "TOO LOW":GOTO 10  
70 END
```


OPEN

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

The OPEN statement opens a channel to a peripheral device such as a printer, disk drive or Datassette for input and output operations. The word OPEN is followed by a logical file number and a device number. The OPEN statement can also include the following optional information: secondary address, a filename, a file type and a file mode.

The logical file number is the number assigned to a file between 1 and 255. The logical file number is referred to by the other input and output commands such as PRINT#, INPUT#, CMD and CLOSE. The OPEN statement associates a file number to a device number.

The device number is the number assigned to a peripheral device. For example, device 1 is the Datassette, device 4 is the printer and device 8 is the disk drive. The device number is implied in other input and output statements with the logical file number.

The optional secondary address specifies an input or output operation on a peripheral device. For example, secondary address 0 on the Datassette specifies a read operation from tape. Secondary address 1 specifies a write operation. These secondary addresses specify, different operations on different devices. Consult your peripheral's user's guide for more secondary address information.

The OPEN statement may specify a filename enclosed in quotation marks, but it is not required for printer or tape files. The filename has a maximum length of 16 characters.

File type specifies which kind of files are being used. There are four types of disk drive files: sequential (SEQ), relative (REL), program (PRG) and user (USR) files. The Datassette uses only program and sequential files. If the file type is not specified, the disk drive and Datassette assume it is a program file unless the mode is specified.

The mode specifies what type of output operation is performed. The modes are Read (R), and Write (W). The mode is usually specified when using disk files.

Example:

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,"UP"	OPENS the Datassette for reading; file to be searched for is named UP.

OPEN 4,4	OPENS a channel to use the printer.
OPEN 15,8,15	OPENS the disk drive command channel.
5 OPEN 8,8,12,"TESTFILE,SEQ,W"	Opens a sequential disk file for writing.

See also: CLOSE, CMD, GET#, INPUT#, and PRINT# statements.

POKE

POKE address, value

The POKE statement changes the contents of a Commodore 64 Random Access Memory (RAM) location. The word POKE is always followed by two numbers. The first number is a memory location. This can have a 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	Places the value 8 in location 28000
20 POKE 28*1000,27	Places the value 27 in location 28000

PRINT

PRINT print list

The PRINT statement outputs characters to the screen. The word PRINT can be followed by any of the following:

Characters inside of quotes ("text lines")	
Variable names	(A1, B, A\$, X\$)
Functions	(SIN(23), ABS(33))
Punctuation marks	(; ,)

The characters inside of quotes are referred to as literals since they are PRINTed exactly as they appear. When a variable name is PRINTed, the contents of the variable is PRINTed, not the variable name (unless it appears within quotation marks).

If more than one variable is PRINTed, they must be separated by commas or semi-colons. A comma places 15 spaces between each output variable or string. A semi-colon PRINTs output variables and strings separated by one space.

Example:

PRINT statement	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; 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 number, print list

PRINT# works just like the PRINT statement except it outputs characters to a previously OPENed file number on a peripheral device instead of the screen.

Example:

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

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 a TYPE MISMATCH ERROR message.

Example:

```
Read A$, G$
```

REM

REM message

The REMark is just a note to whoever is reading a LISTing of the program. It may explain a section of the program, give information about the author, etc. REM statements in no way effect the operation of the program, except to add to its length. The word REM may be followed by any text, although graphic characters may cause unexpected results.

Example:

```
10 NEXT X:REM UPDATE LOOP  
20 REM THIS LINE IS UNNECESSARY .
```

RESTORE

RESTORE line number

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 number 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
```

RETURN

RETURN

This statement is always used in conjunction with GOSUB. When the program encounters a RETURN, it will go to the statement immediately following the GOSUB command. If no GOSUB was previously issued, a RETURN WITHOUT GOSUB ERROR will occur.

STOP

STOP

The STOP statement halts execution of a program. The message, BREAK IN LINE #, is displayed when the program encounters the line number that contains the STOP statement. The program can be re-started at the statement following STOP using the CONT command. The STOP statement is commonly used while debugging programs.

SYS

SYS address

The SYS statement is followed by a decimal number or numeric variable in the range 0 to 65535. The program begins executing a machine language program starting at the specified address. This is similar to the USR function, but does not pass a parameter. See the Commodore 64 Programmer's Reference Guide for information about machine language programs.

WAIT

WAIT address, value 1, value 2

The WAIT statement is used to halt a program until the contents of a memory location change in a specific way. The address must be in the range between 0 to 65535. Value 1 and value 2 must be in the range between 0 and 255.

The contents of the memory location is first logically ANDed with value 1. If value 2 is present, the specified memory location is exclusive-ORed with value 2. If the result is zero, the program checks the memory location again. When the result is not zero, the program continues with the statement directly following WAIT.

• BASIC FUNCTIONS

NUMERIC FUNCTIONS

Numeric functions return a numeric value. The functions they perform range from calculating mathematical functions to specifying the contents of a memory 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.

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.

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. $\text{INT}(-4.5) = -5$).

If the INT function is to be used for rounding up or down, the form is INT(X + / - .5).

Example:

INT(4.75 + .5)

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.

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(0), to start things off differently every time. After this, the number 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 reseeded 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 to 1 is multiplied by 6, which expands the range from 0 to 6 (actually, greater than zero and less than six).

Then 1 is added, making the range 1 to 7. The INT function truncates 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

100 X = INT(RND(1)*1000) + 1

100 X + INT(RND(1)*150) + 100

Simulates 2 dice

Number from 1-1000.

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 785 and 786. 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 Commodore 64 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 that are recognizable. If the Commodore 64 finds illegal characters, only the portion of the string up to that point is converted.

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.

LEFT\$(X\$,X)

This function returns a string containing the leftmost X characters of X\$.

LEN(X\$)

This function returns the number of characters (including spaces and other symbols) in the string X\$.

MID\$(X\$,S,X)

This function returns a sub-string containing X characters, starting from the character specified by S in X\$. MID\$ can also be used on the left side of assignment statement as a variable as well as a function.

RIGHT\$(X\$,X)

This function returns the number of right-most characters specified by X in X\$.

STR\$(X)

This function returns a sub-string which is identical to the PRINTed version of X\$.

Example:

A\$ = STR\$(X)

OTHER FUNCTIONS**FRE(X)**

This function returns the number of available bytes 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 function is used in the PRINT statement to skip X spaces. X can have a value from 0-255.

TAB(X)

This function 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.

• VARIABLES AND OPERATORS

VARIABLES

The Commodore 64 uses three types of variables in BASIC. These are: floating point numeric, integer numeric, and string (alphanumeric) variables.

FLOATING POINT VARIABLES can be displayed up to nine digits. When a number becomes larger than nine digits, as in 10^9 or 10^{-9} , your computer displays it in scientific notation form. For example, the number 12345678901 is displayed as $1.234356789E + 10$. There is a limit to the size of floating-point numbers that BASIC can handle, even in scientific notation. The largest number is $+1.70141183E + 38$. Calculations which result in a larger number will display the BASIC error message ?OVERFLOW ERROR. The smallest floating-point number is $+2.93873588E - 39$. Calculations which result in a smaller value give you zero as an answer and NO error message.

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 your Commodore 64 can make. An example of a string variable is $A\$ = \text{"COMMODORE 64"}$.

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

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 three variable names which are reserved for use by the Commodore 64, and may not be used for another purpose. These are the variables 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 Commodore 64 Programmer's Reference Guide, but in general, if the value of ST is 0 the operation was successful.

TI and TI\$ are variables that relate to the real-time clock built into your Commodore 64. The system clock is updated every 1/60th of a second. It starts at 0 when your Commodore 64 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 second.

TI\$ is a six-character 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 next two characters are the minutes, and the last two 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: $TI\$ = \text{"101530"}$ sets the clock to 10:15 and 30 seconds (AM)

The value of the clock is lost when your Commodore 64 is turned off. It starts at zero when your computer is turned on, and is reset to zero when the value of the clock exceeds 235959 (23 hours, 59 minutes and 59 seconds).

BASIC OPERATORS

The ARITHMETIC operators include the following signs:

- + addition
- subtraction
- * multiplication
- / division
- ↑ exponentiation (raising to a power)

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, Commodore 64 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.

- = equal to
- < less than
- > greater than
- < = or = < less than or equal to
- > = or = > greater than or equal to
- < > or > < 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.

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

GLOSSARY

The following glossary contains definitions for some of the computer terms used in this Guide. Most of the terminology in the glossary is universal. However, there are certain terms that are unique to the Commodore 64. Consult the Commodore 64 Programmer's Reference Guide for more detailed information on the Commodore 64 computer.

Alphanumeric—Letters, numbers and special symbols found on the keyboard, excluding graphic characters.

Animation—The simulation of motion of an object on the screen through gradual, progressive movements, using computer instructions.

Array—A data-storage structure in which a series of related constants or variables are stored in consecutive memory locations. Each constant or variable contained in an array is referred to as an element. An element is accessed using a subscript. See Subscript.

ASCII Code—Acronym for American Standard Code for Information Interchange (ASCII). A standard computer code of numeric values representing each keyboard character. See Character String Code.

Assignment Statement—A BASIC statement that sets a variable, constant or array element to a specific numeric or string value.

Attack—The rate at which the volume of a musical note rises from zero to peak volume.

Background Color—The color of the portion of the screen that does not contain characters. The background color is dark blue when you turn on the Commodore 64.

BASIC—Acronym for *Beginner's All-purpose Symbolic Instruction Code*. BASIC is the high-level language built into the Commodore 64.

Binary—A base-2 number system. All numbers are represented as a sequence of zeros and ones.

Bit—The abbreviation for binary digit. A bit is the smallest unit in a computer. Each binary digit can have one of two values, zero or one. A bit is referred to as enabled or "on" if it equals one. A bit is disabled or "off" if it equals zero.

Bit Map Mode—An advanced graphic mode in the Commodore 64 in which you can control every dot on the screen.

Border Color—The color of the edges around the screen. The border color is cyan (light blue) when you turn on the Commodore 64.

Branch—To jump to a section of a program and execute it. GOTO and GOSUB are examples of BASIC branch instructions.

Byte—The number of bits that make up the smallest unit of addressable storage in a computer. Each memory location in the Commodore 64 contains one byte of information. One byte is the unit of storage needed to represent one character in memory. One byte is made up of eight bits. See Bit.

Character—Any symbol on the computer keyboard that is printed on the screen. Characters include numbers, letters, punctuation and graphic symbols.

Character Memory—The area in Commodore 64's memory which stores the encoded character patterns that are displayed on the screen.

Character Set—A group of related characters. The Commodore 64 character sets consist of: upper-case letters, lower-case letters and graphic characters.

Character String Code—The numeric value assigned to represent a Commodore 64 character in the computer's memory.

Chip—A miniature electronic circuit that performs a computer operation such as graphics, sound and input/output.

Color Memory—The area in the Commodore 64's memory that controls the color of each location in screen memory.

Command—A BASIC instruction used in direct mode to perform an action. See Direct Mode.

Computer—An electronic, digital device that stores and processes information.

Condition—Expression(s) between the words IF and THEN, evaluated as either true or false in an IF . . . THEN statement. The conditional IF . . . THEN statement gives the computer the ability to make decisions.

Coordinate—A single point on a grid having vertical (Y) and horizontal (X) values.

Counter—A variable used to keep track of the number of times an event has occurred in a program.

Crunch—To minimize the amount of computer memory used to store a program.

Cursor—The flashing square that marks the current location on the screen.

Data—Numbers, letters or symbols that are input into the computer to be processed.

Datassette—A device used to store programs and data files sequentially on tape.

Debug—To correct errors in a program.

Decay—The rate at which the volume of a musical note decreases from its peak value to a mid-range volume called the sustain level. See Sustain.

Decrement—To decrease an index variable or counter by a specific value.

Delay Loop—An empty FOR . . . NEXT loop that slows the execution of a program.

Dimension—The property of an array that specifies the direction along an axis in which the array elements are stored. For example, a two-dimensional array has an X-axis for rows and a Y-axis for columns. See Array.

Direct Mode—The mode of operation that executes BASIC commands immediately after the RETURN key is pressed. Also called Immediate Mode. See Command.

Disable—To turn off a bit, byte or specific operation of the computer.

Disk Drive—A random access, mass-storage device that saves and loads files to and from a floppy diskette.

Duration—The length of time a musical note is played.

Enable—To turn on a bit, byte or specific operation of the computer.

Envelope Generator—Portion of the Commodore 64 that produces specific waveforms (sawtooth, triangle, pulse width and noise) for musical notes. See Waveform.

Execute—To perform the specified instructions in a command or program statement.

Expression—A combination of constants, variables or array elements acted upon by logical, mathematical or relational operators that return a numeric value.

File—A program or collection of data stored on diskette or cassette.

Firmware—Computer instructions stored in ROM, as in a game cartridge.

Frequency—The number of sound waves per second of a tone. The frequency corresponds to the pitch of the audible tone.

Function—A predefined operation that returns a single value.

Function Keys—The eight keys on the far right of the Commodore 64 keyboard. Each key can be programmed to execute a series of instructions.

Graphics—Visual screen images representing computer data in memory (i.e., characters, symbols and pictures).

Graphic Characters—Non-alphanumeric characters on the computer's keyboard.

Grid—A two-dimensional matrix divided into rows and columns. Grids are used to design sprites and programmable characters.

Hardware—Electronic components in a computer system such as keyboard, disk drive and printer.

Home—The upper-left corner of the screen.

Increment—To increase an index variable or counter with a specified value.

Index—The variable counter within a FOR . . . NEXT loop.

Input—Data fed into the computer to be processed. Data can be input through the keyboard, disk drive, Datassette or modem.

Integer—A whole number containing no fractional part.

Interface—An attachment that connects a computer to a peripheral device.

Keyboard—Input component of a computer system.

Kilobyte (K)—1,024 bytes.

Loop—A program segment executed repetitively a specified number of times.

Machine Language—The lowest level language the computer understands. The computer converts all high-level languages such as BASIC into machine language before executing any statements.

Matrix—A two-dimensional rectangle with row and column values.

Memory—Storage locations inside the computer. ROM and RAM are two different types of memory.

Memory Location—A specific storage address in the computer. There are 65,536 memory locations (0-65535) in the Commodore 64.

Mode—A state of operation.

Modem—Abbreviation for modulator-demodulator. A modem is a computer attachment that interfaces a computer to a telephone. This allows you to communicate with other computers using the same lines as your telephone.

Monitor—Video screen.

Multi-Color Character Mode—A graphic mode that allows you to display four different colors within an 8 X 8 character grid.

Multi-Color Bit Map Mode—A graphic mode that allows you to display one of four colors for each pixel within an 8 X 8 character grid. See Pixel.

Null String—An empty character (""), a character that is not yet assigned a character string code. Produces an illegal quantity error if used in a GET statement.

Octave—One full series of eight notes on the musical scale.

Operating System—A built-in program that controls everything your computer does.

Operator—A symbol that tells the computer to perform a mathematical, logical or relational operation on the specified variables, constants or array elements in the expression. The mathematical operators are +, -, *, / and †. The relational operators are <, =, >, <=, >= and <>. The logical operators are AND, OR and NOT.

Order of Operations—Sequence in which computations are performed in a mathematical expression. Also called Hierarchy of Operations.

Peripheral—Any accessory device attached to the computer such as a disk drive, printer, modem or joystick.

Pitch—The highness or lowness of a tone that is determined by the frequency of the sound wave. See Frequency.

Pixel—Computer term for picture element. Each dot on the screen that makes up an image is called a pixel. Each character on the screen is displayed within an 8 X 8 grid of pixels. The entire screen is composed of a 320 X 200 pixel grid. In bit map mode, each pixel corresponds to one bit in the computer's memory.

Pointer—A register used to indicate the address of a location in memory.

Printer—Peripheral device that outputs the contents of the computer's memory onto a sheet of paper. This paper is referred to as a hard copy.

Program—A series of instructions that direct the computer to perform a specific task. Programs can be stored on diskette or cassette, reside in the computer's memory, or be listed on a printer.

Programmable—Capable of being processed with computer instructions.

Program Line—A statement or series of statements preceded by a line number in a program. The maximum length of a program line on the Commodore 64 is 80 characters.

Random Access Memory (RAM)—The programmable area of the computer's memory that can be read from and written to (changed). All RAM locations are equally accessible at any time in any order. The contents of RAM are erased when the computer is turned off.

Random Number—A nine-digit decimal number from 0.000000001 to 0.999999999 generated by the RND function.

Read Only Memory (ROM)—The permanent portion of the computer's memory. The contents of ROM locations can be read, but not changed. The ROM in the Commodore 64 contains the BASIC language interpreter, character image patterns and portions of the operating system.

Register—Any memory location in RAM. Each register stores one byte. A register can store any value between 0 and 255 in binary form.

Release—The rate at which the volume of a musical note decreases from the sustain level to zero.

Remark—Comments used to document a program. Remarks are not executed by the computer, but are displayed in the program listing.

Resolution—The density of pixels on the screen that determine the fineness of detail of a displayed image.

Screen—Video display unit which can be either a television or video monitor.

Screen Code—The number assigned to represent a character in screen memory. When you type a key on the keyboard, the screen code for that character is entered into screen memory automatically. You can also display a character by storing its screen code directly into screen memory with the POKE command.

Screen Memory—The area of the Commodore 64's memory that contains the information displayed on the video screen. The Commodore 64 screen memory ranges from memory location 1024 through 2023.

Software—Computer programs stored on diskette or cassette that can be loaded into random access memory.

Sound Interface Device (SID)—The MOS 6581 sound synthesizer chip responsible for all of the audio features of the Commodore 64. See the Commodore 64 Programmer's Reference Guide for chip specifications.

Sprite—A programmable, movable, high-resolution graphic image. Also called a movable object block (MOB).

Standard Character Mode—The mode the Commodore 64 operates in when you turn it on and when you write programs.

Statement—A BASIC instruction contained in a program line.

String—An alphanumeric character or series of characters surrounded by quotation marks.

Subroutine—An independent program segment separate from the main program that performs a specific task. Subroutines are called from the main program with the GOSUB statement and must end with a RETURN statement.

Subscript—A variable or constant that refers to a specific element in an array by its position within the array.

Sustain—The midranged volume of a musical note.

Syntax—The grammatical rules of a programming language.

Tone—An audible sound of specific pitch and waveform.

Variable—A unit of storage representing a changing alphanumeric value. Variable names can be any length, but only the first two characters are stored by the Commodore 64. The first character must be a letter.

Video Interface Controller (VIC)—The MOS 6566 chip responsible for all the graphics features of the Commodore 64. See the Commodore 64 Programmer's Reference Guide for chip specifications.

Voice—A sound producing component inside the SID chip. There are three voices within the SID chip so the Commodore 64 can produce three different sounds simultaneously. Each voice consists of a tone oscillator/waveform generator, an envelope generator and an amplitude modulator.

Waveform—A graphic representation of the shape of a sound wave. The waveform determines some of the physical characteristics of the sound.

• INDEX

A

Abbreviations—BASIC, 152–154
ABSolute function, 41, 181
Accessories, 11
Addition, 36
ADSR, 112
Animation, 83, 97
Arrays, 63–64, 187
ASC function, 70, 181
ASCII character codes, 141–143
Asterisk key, 31, 37
At symbol, 49
Attack, 149, 193
ATN function, 181
Automodem, 14

B

BASIC

abbreviations, 152–154
commands, 39, 47–50, 56,
162–167
conversions, 155
language, 35, 193
math functions, 40–41
numeric functions, 40, 72,
181–182
operators, 37, 45
control functions, 184–185
statements, 45, 53–56, 61–69,
168–180
string functions, 70–71, 183–184
variables, 42–44, 186–188

Binary, 193

Bit, 193

Bit Map mode, 103, 194

Books, 158–159

Byte 77, 194

C

Cartridge slot, 124

Cartridges, 27

Cassette tape recorder, 11

Channel selector, 125

Character Display mode, 103

Checklist, 134

CHR\$ codes, 141–143

CHR\$ function, 70, 76, 183

CLR statement, 168

CLR/HOME key, 23

Clock, 187

CLOSE statement, 50, 168

CMD, 168

Colon, 65

Color

code display, 76

control, 75

CHR\$ codes, 76

keys, 76

memory map, 82, 137

screen and border registers, 77

screen codes, 79

Comma, 36

Command, BASIC, 194

Commodore Information network,
122

Commodore key, 23, 75
Commodore Library, 135
Compuserve, 122
Concatenation, 155
Connections, 123–128
constants, 42
CONT command, 56, 162
ConTRoL key, 22, 75
COPY command, 162
Copying music, 109
Copying programs, 35
COSine function, 181
CuRSoR keys, 21
Cursor, 8
Customer Support, 122
D
Datasette, 11
DATA statement, 61, 169
Debug, 195
Decay, 149, 195
DEFine statement, 169
Delay loop, 80
DELeTe key, 21, 22
Dice, 72
DIMension statement, 63, 66, 169
Direct mode, 39, 195
Disk commands, 49–50
Disk Directory, 49
Disk Drive, 11, 13, 30
Disk Software, 30
Disks, 27, 47
Division, 37
Dollar sign, 49
Duration, 108, 196
E
Editing programs, 46
END statement, 63, 170
Error messages, 117–120
EXPoNent function, 181
Extended background color, 104

F
File, 176, 196
Floppy, 30
FN function, 181
FOR . . . NEXT statement, 53, 170
Formatting disks, 47
FRE function, 184
Frequency, 149, 196
Function keys, 24, 70, 196

G
Game controls and ports, 124
GET statement, 55, 171
GET# statement, 172
GOSUB statement, 67–68, 172
GOTO statement, 42, 172
Graphic keys, 24
Graphic modes, 102

H
High resolution mode, 103
HOME key, 23
Hyperbolic functions, 151

I
IF . . . THEN statement, 45, 173
Initialize command, 50
INPUT statement, 54, 173
INPUT#, 174
INSeRT key, 21
INTeGer function, 40, 72, 181
Integer variable, 186

J
Joystick ports, 124
Joysticks, 13

K
Keyboard, 19–24

L
LEFT\$ function, 184
LEnGth function, 184
LET statement, 155, 174
LIST command, 49, 162
LOAD command, 48–49, 163
LOADing cassette software, 28–29
LOADing cartridge software, 28
LOADing disk software, 30–31
LOGarithm function, 182
Loops, 64, 171

M
Machine language, 179, 183, 197
Magazine subscription, 122
Memory, 69, 77, 197
Memory maps, 80–82, 136–137
MID\$ function, 184
Modem, 13, 197
Multicolors, 103, 197
Multiplication, 37
Music programs, 109–111
Musical notes, 108, 147
Musical scale, 107

N
NEW command, 165
NEXT statement, 53, 174
Noise, 156
Null string, 198
Numeric variables, 71

O
ON statement, 68, 175
OPEN statement, 50, 176
Operating System, 198
Operators, 198
arithmetic, 37, 188
logical, 173, 189
order of, 37, 188–189,
198
relational, 45, 173, 188

P
Paddles, 13
Parentheses, 38, 50
PAUSE, 156
PEEK function, 69, 182
Peripherals, 198
Pitch, 198
Pixel, 87, 198
POKE statement, 69, 177
Ports, 124–127
POS function, 185
PRINT statement, 36–39, 177
PRINT AT, 156
Printers, 12, 13
PRINT#, 178
Program, 198
line numbering, 42
mode, 41
music, 110
Programmable keys, 24
Programmer's Reference Guide, 32,
104, 112, 147, 180
Pulse, 149

Q
Question mark, 37
Quotation marks, 38
Quote mode, 39

R
RAM, 14, 69, 199
RaNDom function, 72, 182
Random numbers, 72, 156, 199
READ statement, 61, 178
Registers, 77, 199
Release, 149, 199
REMArk statement, 178, 199
Replace a program, 49
Reserved variables, 187
Restore key, 23, 102
RESTORE statement, 62, 179
Return key, 20

RETURN statement, 67, 179
RIGHT\$ function, 184
ROM, 14, 69
RUN command, 29, 31, 165
RUN/STOP key, 22

S

SAVE command, 47–48, 165
Saving programs (tape), 47
Saving programs (disk), 48
SCRATCH command, 50, 166
Screen codes, 138–140, 199
Screen memory map, 81, 136
Semicolon, 36
serial port, 126
SGN function, 183
Shift key, 20
Shift lock key, 21
SID chip, 107, 200
SINe function, 183
Slash key, 37
Software
 business, 130
 educational, 130–132
 financial, 129
 games, 132–133
 music, 134
 productivity, 128–129
 programming aids, 134
 using, 27–32
Song, 109
Sound effects, 112–114
Sound registers, 108, 150
SPC function, 40, 185
Sprite control, 94–99
Sprite programming, 100
Sprite Register Map, 144–146
Sprite viewing area, 99
Sprites, 87–94, 200
SQaRe function, 40, 183
STEP, 84, 170

STOP statement, 56, 179
STOP key, 22, 45
Storing Programs, 47
String variables, 44, 186
Strings, 44, 71, 200
STR\$ function, 71, 184
Subroutine, 67, 200
Subscripts, 200
Subtraction, 36
Sustain, 149, 200
Syntax, 163, 200
Syntax error, 119
SYS statement, 179

T

TAB function, 39, 185
TAN function, 183
Telecommunications, 122
THEN, 45
TI variable, 187
TI\$ variable, 187
Trackball, 13
Troubleshooting chart, 9, 120–121

U

Up arrow key, 37
Upper case/graphic mode, 20
Upper/Lower Case mode, 20
User groups, 122
User port, 127
USR function, 183

V

VALue function, 71, 183
Variables
 array, 65
 dimensions, 65
 floating point, 186
 integer, 186
 numeric, 43
 string (\$), 44, 186

VERIFY command, 49, 166
VIC chip, 87, 201
Voice, 201

W

WAIT command, 180
Waveform, 149, 201

NEW! RECREATIONAL BLOCKBUSTERS FOR YOUR COMMODORE 64!



**Don't Miss
Out On The Fun!**

SATAN'S HOLLOW—Battle Satan's hordes as you build a bridge across the River of Fire to Satan's Hollow. But you must be brave to complete the righteous challenge as you now cross the bridge and face Satan himself. Exciting graphics and action make this BALLY/MIDWAY conversion a winner! (Diskette)

VIDUZZLES—Have fun building puzzles on your computer screen. Construct an owl, clown or dog puzzle using 25 or 50 pieces. Compete against the clock or a friend. Ideal for children. You don't even have to worry about losing the pieces! (Cartridge)

RALLY SPEEDWAY—Gentlemen, start your engines! The only raceway game with unique two player action. Hear the roar of your engines as you thunder around the challenging track. Design custom courses to enjoy endless racing challenge as your skill increases. (Diskette)

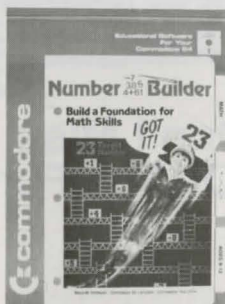
 **commodore**

Just Released Educational Programs That Get An "A"

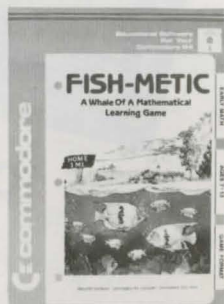
SKY TRAVEL—(Ages 10-Adult) The most advanced and complete astronomy program on the market today. Shows the location of over 1,200 stars, 88 constellations, the sun, the earth's moon, 8 planets, Halley's comet, and many deep space objects. Detailed information on each is available through special Inform and Find functions. Create your own star and planetary charts. Even travel 10,000 years into the past or future to gaze at the stars! (Diskette)



JUST IMAGINE—(Ages 4-14) One of the most unique educational programs today! Children can create their own animated stories and watch as they come to life. Different background scenes combine with animated objects and written stories to construct a film type sequence. Develops spelling, reading and creative writing skills. (Diskette)



NUMBER BUILDER—(Ages 8-13) You must reach a designated number, but how? Should you add, subtract, multiply or divide? By using the correct sequence of mathematical operations, Number Builder allows you to arrive at the number and in addition strengthen your math skills. Includes arcade type format, several difficulty levels, and self test. (Diskette)



FISH-METIC—(Ages 7-13) A whale of a mathematical learning game that teaches the concepts of greater than, less than, and equal to in a game type format. Apply these to positive and negative whole numbers, fractions, and decimals. Includes 15 preliminary difficulty levels and an additional super 16th level for endless challenge. (Diskette)

Two New Arrivals To Our Productivity Line Of Software

B/GRAPH—A professional graphics-charting and statistical analysis program that allows you to create numerous types of graphs and charts, as well as providing statistical and analytical tools for evaluating data. Not only valuable, but easy to learn too. (Diskette)

SILENT BUTLER—You don't have to be rich to have a butler. With this new home accounting program you can keep track of bill paying, household transactions, and up to 6 accounts—3 checking and 3 saving. Pay bills using your personal checks and a printer. Includes a tax summary and appointment reminder. (Diskette)



commodore
COMPUTERS

COMMODORE 64 BOOKS

MATHEMATICS on the Commodore 64—Learn the math techniques that professional programmers use. Covers a wide variety of mathematical techniques which you can incorporate in all types of BASIC. Contains numerous subroutines and examples with illustrations.

ADVANCED PROGRAMMING TECHNIQUES on the Commodore 64—Packed with advice and tips for intermediate and advanced BASIC programmers. Learn the shortcuts you need to move up and become a more "professional" programmer.

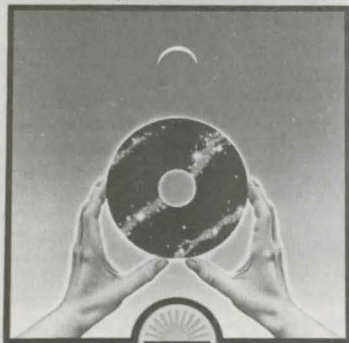
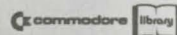
PROGRAMMING FOR EDUCATION on the Commodore 64—Learn how to write creative learning programs in BASIC for pre-schoolers and elementary age children. Excellent for parents as well as teachers.

COMMODORE 64 ADVENTURES—Learn to create your own adventure programs! See how each module of a real adventure is built. Contains many diagrams, maps, charts and playing tips. A great way to make your dreams come alive on the screen.

commodore 1541 disk companion

secrets of the 1541
disk drive

david lawrence
and mark england

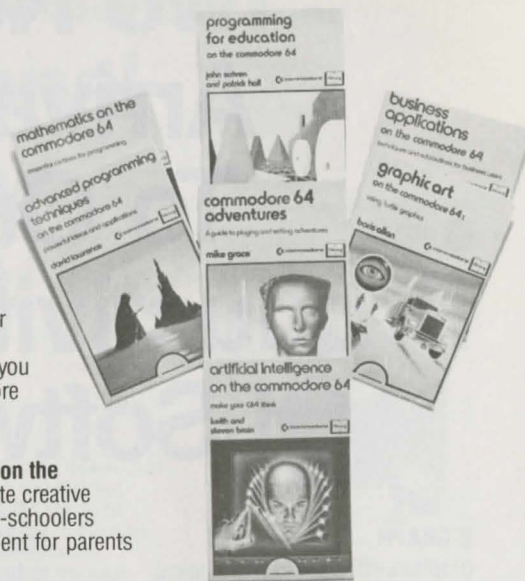


GRAPHIC ART on the Commodore 64—Contains information on high resolution graphics techniques and subroutines. Develop your own turtle graphics system. All programs are in BASIC.

ARTIFICIAL INTELLIGENCE on the Commodore 64—One of the hottest new topics in personal computing! Learn the concepts and fundamentals of making your computer "think" for itself. A topic that is both entertaining and intriguing. Artificial Intelligence is almost certainly the wave of the future.

BUSINESS APPLICATIONS on the Commodore 64—Covers the techniques of writing business programs in BASIC. Provides a rich library of solid routines that can be combined to build up programs for accounts, stock control, inventories, invoices and much, much more.

COMMODORE 1541 DISK COMPANION—Everything you always wanted to know about disks and the 1541. Includes a full range of advanced disk drive techniques. THE text if you are looking for a comprehensive bible on the Commodore 1541 disk drive.



NOTES

NOTES

NOTES

NOTES

NOTES

A SPECIAL OFFER FOR NEW OWNERS!

Get the most out of your Commodore computer.

... With Commodore magazines. And save 20% off the regular newsstand price.

Subscribe to *Commodore Power/Play* and *Commodore Microcomputers* magazines and you're on your way to realizing the full power and potential of your new Commodore computer.

Each issue brings you new ways to use and enjoy your computer. The first word on new software and hardware. Programming techniques for both beginners and advanced users. In-depth product reviews of the best software and hardware. The latest games or education and applications programs. Visits with other users who have discovered new and interesting ways to use their Commodore computers.

You'll find practical articles on linking up with user groups in your area. Telecommunications and using on-line services such as CompuServe. Computer music and art, and much, much more.

In addition, every issue contains programs you can type in yourself and use right away. There's entertainment and games or practical household and business applications programs in each issue!

Together, they're the perfect combination of pure fun and productivity!

And if you take advantage of this special offer—only for new computer owners—you can save as much as 20% off the regular newsstand price!

Subscribe now and get the most out of your Commodore computer. And save as much as 20%!

----- DETACH AND MAIL TODAY -----

Please sign me up for

- year(s) of *Power/Play* and *Microcomputers* (12 issues total per year) at \$24/year (a savings of 20% off the regular newsstand price).
- year(s) of *Power/Play* only (entertainment and games—6 issues per year) at \$15/year.
- year(s) of *Microcomputers* only (more in-depth information about practical ways to use your computer—6 issues per year) at \$15/year.

ALL PRICES IN US CURRENCY. Canadian add \$5.00 to each subscription to cover postage. Overseas: \$25.00/6 issues (includes postage).

Name _____ Phone _____

Address _____

City _____ State _____ Zip _____

Signature _____

METHOD OF PAYMENT

- Enclosed is my check or money order for \$ _____ (Make check or money order payable to **COMMODORE PUBLICATIONS**)
- Bill me
- Charge my VISA or MasterCard (circle one) Card number

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

 Expiration Date _____

or call 800-345-8112 to order (in Penna. 800-662-2444)



BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 251 HOLMES, PA

POSTAGE WILL BE PAID BY ADDRESSEE

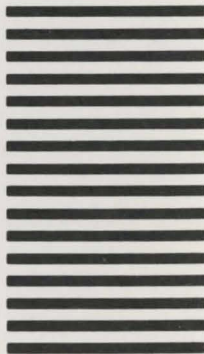
Commodore Publications

Magazine Subscription Department

Box 651

Holmes, PA 19043

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



About the Commodore 64 User's Manual

The Commodore 64 user's manual is an easy-to-use guide that helps you get started in the world of computing. Through clear, step-by-step instructions, you can learn how your Commodore 64 can be put to an assortment of fun and valuable uses.

Introductory topics discussed in this manual include:

- How to set up your computer
- BASIC programming for beginners
- Sprite graphics
- Creating sound

For those of you already familiar with microcomputers, the advanced programming sections help to explain the special features of the Commodore 64. Learn to expand your present programming capabilities through such informative sections as:

- Advanced BASIC
- Advanced color and graphic commands
- Advanced data handling

Also supplied is an informative appendix which contains charts, tables and error messages.



Commodore Business Machines, Inc.
1200 Wilson Drive • West Chester, PA 19380

Commodore Business Machines, Limited
3370 Pharmacy Avenue • Agincourt, Ontario, M1W 2K4

Copyright 1984 Commodore Electronics, Ltd. Printed in U.S.A.