COMMODORE 128

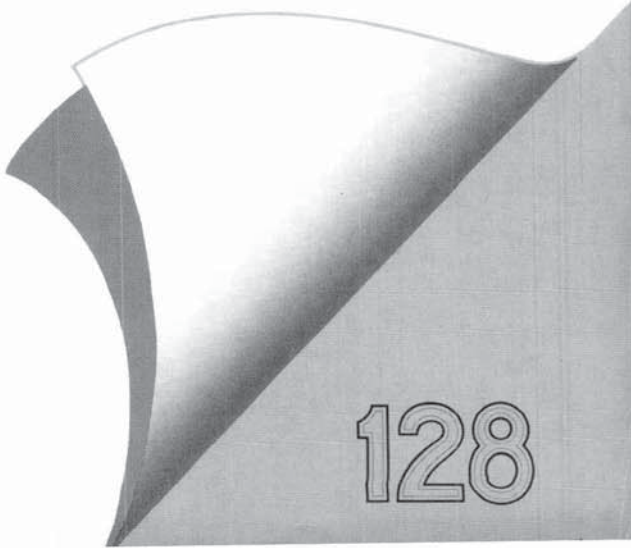
PERSONAL COMPUTER

System Guide



Personal Computer System Guide

128



©Commodore Electronics, Ltd. 1985. All rights reserved.

USER'S MANUAL STATEMENT

WARNING:
This equipment has been certified to comply with the limits for a Class B computing device, pursuant to subpart J of Part 15 of the Federal Communications Commission's rules, which are designed to provide reasonable protection against radio and television interference in a residential installation. If not installed properly, in strict accordance with the manufacturer's instructions, it may cause such interference. If you suspect interference, you can test this equipment by turning it off and on. If this equipment does cause interference, correct it by doing any of the following:

- Reorient the receiving antenna or AC plug.
- Change the relative positions of the computer and the receiver.
- Plug the computer into a different outlet so the computer and receiver are on different circuits.

CAUTION: Only peripherals with shield-grounded cables (computer input-output devices, terminals, printers, etc.), certified to comply with Class B limits, can be attached to this computer. Operation with non-certified peripherals is likely to result in communications interference.

Your house AC wall receptacle must be a three-pronged type (AC ground). If not, contact an electrician to install the proper receptacle. If a multi-connector box is used to connect the computer and peripherals to AC, the ground must be common to all units.

If necessary, consult your Commodore dealer or an experienced radio-television technician for additional suggestions. You may find the following FCC booklet helpful: "How to Identify and Resolve Radio-TV Interference Problems." The booklet is available from the U.S. Government Printing Office, Washington, D.C. 20402, stock no. 004-000-00345-4.

Copyright © 1985 by Commodore Electronics Limited
All rights reserved

This manual contains copyrighted and proprietary information. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of Commodore Electronics Limited.

Commodore BASIC 7.0

Copyright © 1985 by Commodore Electronics Limited
All rights reserved

Copyright © 1977 by Microsoft Corp.
All rights reserved

CP/M® Plus Version 3.0

Copyright © 1982 by Digital Research Inc.
All rights reserved

CP/M is a registered trademark of Digital Research Inc.

TABLE OF CONTENTS

Chapter I—Introduction

Section 1—How to Use this Guide 3
Section 2—Overview of the Commodore C128 Personal Computer 7

Chapter II—Using C128 Mode

Section 3—Getting Started in BASIC 17
Section 4—Advanced BASIC Programming 49
Section 5—Some BASIC Commands and Keyboard Operations Unique to C128 Mode 73
Section 6—Color, Animation and Sprite Graphics Statements Unique to the C128 93
Section 7—Sound and Music in C128 Mode 129
Section 8—Using 80 Columns 161

Chapter III—Using C64 Mode

Section 9—Using the Keyboard in C64 Mode 171
Section 10—Storing and Reusing your Programs in C64 Mode 177

Chapter IV—Using CP/M Mode

Section 11—Introduction to CP/M 3.0 185
Section 12—Files, Disks and Disk Drives in CP/M 3.0 193
Section 13—Using the Console and Printer in CP/M 3.0 203
Section 14—Summary of Major CP/M 3.0 Commands 209
Section 15—Commodore Enhancements to CP/M 3.0 219

Chapter V—Basic 7.0 Encyclopedia

Section 16—Introduction 227
Section 17—BASIC Commands And Statements 233
Section 18—BASIC Functions 303
Section 19—Variables And Operators 323
Section 20—Reserved Words and Symbols 329

Appendices

A. BASIC Language Error Messages 335
B. DOS Error Messages 341
C. Connectors/Ports for Peripheral Equipment 347
D. Screen Display Codes 353
E. ASCII and CHR$ Codes 355
F. Screen and Color Memory Maps 357
G. Derived Trigonometric Functions 361
H. Memory Map 363
I. Control and Escape Codes 365
J. Machine Language Monitor 369
K. BASIC 7.0 Abbreviations 379
L. Disk Command Summary 383

Glossary 385

Index 399

CHAPTER

1

INTRODUCTION



SECTION 1
How to Use this Guide

How to Use this Guide

This ***Commodore 128 System Guide*** is designed to help you make full use of the advanced capabilities of the Commodore 128 computer. Here's how to use this Guide:

Before you read any further in this ***System Guide,*** make sure you have read the other book packed in the computer carton, ***Introducing The Commodore 128 Personal Computer,*** which contains important information on getting started with the Commodore 128.

If you are primarily interested in using the BASIC language to create and run your own programs, you should first read Section 2 of this chapter. This section summarizes the three operating modes of the Commodore 128. Then read Chapter II, USING C128 MODE. This chapter introduces you to the BASIC programming language as used in both C128 and C64 modes; describes the Commodore 128 keyboard; defines some advanced commands you can use in both C128 and C64 modes; shows how to use a number of powerful new BASIC commands (including color, graphic and sound commands) that are unique to C128 mode; and describes how to use the 80-column capabilities available in C128 mode.

If you want to use BASIC in C64 mode, read Chapter III, USING C64 MODE. You can use all the Commodore 64 BASIC 2.0 commands in C64 mode. Note, however, that the Commodore 128 BASIC 7.0 language provides many more BASIC commands than BASIC 2.0, and the C128 BASIC commands are more powerful and easier to use than equivalent BASIC 2.0 commands. *Remember, you can use C64 mode to run any of the thousands of C64 software packages currently available.*

If you want to use CP/M on the Commodore 128, read Chapter IV, USING CP/M MODE. This chapter tells you how to start up and use CP/M on the Commodore 128. In CP/M mode you can choose from thousands of commercial software packages, including the PERFECT series (PERFECT WRITER, PERFECT CALC, PERFECT FILER). You can also create your own CP/M programs.

If you want details on the BASIC 7.0 commands, read Chapter V, BASIC 7.0 ENCYCLOPEDIA. This chapter gives format and usage details on all BASIC 7.0 commands, statements and functions.

If, after reading Chapters I through V, you are looking for additional technical information about a particular Commodore 128

topic, first check the *Appendices* to this ***System Guide.*** These appendices contain a wide range of information, such as a complete list of BASIC and DOS error messages and a summary of disk commands. A *Glossary* following the Appendices provides definitions of computing terms.

For complete technical details about any feature of the Commodore 128, consult the Commodore 128 Programmer's Reference Guide.

SECTION 2
Overview of the Commodore C128 Personal Computer

OVERVIEW OF THE COMMODORE C128 PERSONAL COMPUTER 9
C128 Mode 10
C64 Mode 10
CP/M Mode 11

TURNING ON YOUR COMMODORE C128 11

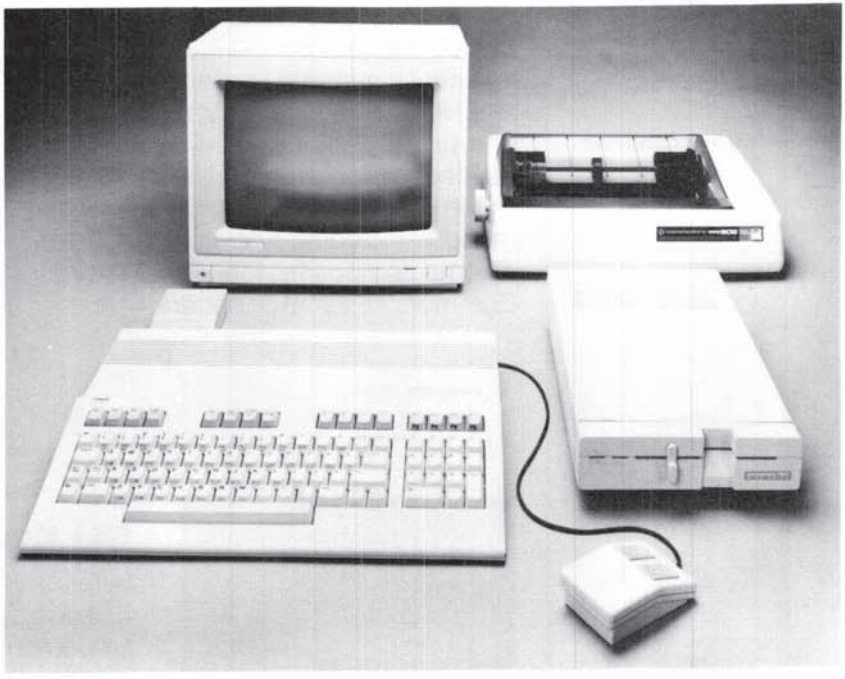
USING SOFTWARE 12

SWITCHING BETWEEN MODES 13

Overview of the Commodore C128 Personal Computer

The Commodore 128 incorporates many powerful new features, including:

- A greatly enhanced BASIC language—Commodore BASIC 7.0—that provides extensive new commands and capabilities
- 128K of RAM, which can be expanded to 256 or 512K with optional RAM expansion modules
- 40- and 80-column output
- Operative with new 1571 fast disk drive
- 2 mHz operation
- CP/M 3.0 operation
- A professional-type keyboard including a full numeric keypad
- A built-in machine language monitor



The Commodore 128 Personal Computer is actually three computers in one, offering three primary operating modes:

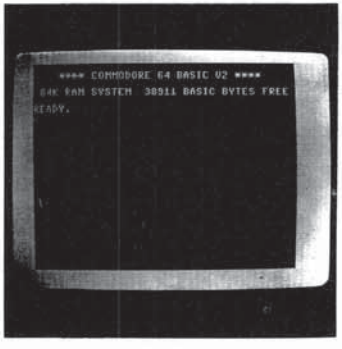
- C128 Mode
- C64 Mode
- CP/M Mode

Here's a summary of what each mode offers:

C128 Mode



In C128 mode, the Commodore 128 Personal Computer provides access to 128K of RAM and a powerful extended BASIC language known as BASIC 7.0. BASIC 7.0—which offers over 140 commands, statements and functions—has been created by Commodore to provide better and easier ways to perform many sophisticated programming tasks, including those involving graphics, animation, sound and music. C128 mode also provides both 40- and 80-column output capabilities and full use of the 92-key keyboard. The keyboard includes a numeric keypad in addition to Escape, Tab, Alpha Lock and Help keys. A built-in machine language monitor allows you to create and debug your own machine language programs. You can use these programs in conjunction with a BASIC program. In C128 mode you can use a number of new peripheral devices from Commodore, including a new fast-serial disk drive, a mouse, and a 40/80-column composite video/RGBI monitor. And you can use all standard Commodore serial peripherals.



C64 Mode

In C64 mode, the Commodore 128 operates exactly like a Commodore 64 computer. The Commodore 128 retains all the capabilities of the commercially successful C64, thus allowing you to take full advantage of the wide range of available C64 software. You also have full compatibility with C64 peripherals, including standard cassette, joystick, user port and serial devices, as well as C64 composite video monitor and TV outputs.

C64 mode provides the BASIC 2.0 language, 40-column output and access to 64K of RAM. The main keyboard layout, except for the placement of the function keys, is the same as that of a Commodore 64 computer. All the C64 graphics, color and sound capabilities are retained, used exactly as on a Commodore 64.

CP/M Mode



In CP/M mode, an onboard Z80 microprocessor gives you all the capabilities of Digital Research's CP/M Plus version 3.0, plus several new capabilities added by Commodore. The Commodore 128's CP/M package, called CP/M Plus, provides 128K of RAM; 40- and 80-column output; access to the full keyboard, including the numeric keypad and special keys; and access to the new Commodore 1571 fast serial disk drive as well as standard serial peripherals.

Chapters II, III and IV, which include Sections 3 through 15, tell you how to access and use the capabilities of the three powerful and versatile operating modes of the Commodore 128 Personal Computer.



Turning On Your Commodore 128

Before you turn on your Commodore 128, there are a few things to check to make sure that you get started properly. One thing you should do before powering up the computer is to make sure the 40/80 key on the top row of the keyboard is set to match your monitor. For example, if you have a 40-column monitor, the 40/80 key should be in the up position. If you have an 80-column monitor the 40/80 key should be depressed.

If you are using the Commodore 1902 dual monitor in 40-column format, the 40/80 key should be up and the slide switch on the front of the monitor should be in the middle position. In 80-column format using the 1902 dual monitor, the 40/80 key should be depressed and the switch on the front of the monitor should be in the extreme right position.

Regardless of which screen format you are using, check to see that both the ALL CAPS and SHIFT LOCK keys are in the up position. If they're not, you may get no picture at all, or the screen may display unfamiliar symbols. (See Section 5 for a description of all the special keys used in C128 mode.)

Using Software

If you are using a MAGIC VOICE speech module, insert the module in the expansion port and, while holding down the Commodore key, turn on the power switch. **Never plug in any cartridge with the power turned on.**

If you experience difficulty getting a cartridge to power-up in C64 mode, plug in the cartridge with the power off; then hold down the Commodore key and turn on the computer.

If you have the external CP/M 2.2 cartridge marketed for the Commodore 64, do not plug it into the Commodore 128. The Commodore 128 has a Z80 microprocessor already on-board for CP/M 3.0. If you do plug in the CP/M 2.2 cartridge, it can cause unpredictable results.

If you are using software involving a light pen, plug the light pen into Controller Port 1, located on the right side of the C128 near the power switch.

Switching Between Modes

The following chart tells how to switch to one mode from another.

TO	FROM: OFF	FROM: C128 40 COL	FROM: C128 80 COL	FROM: C64	FROM: CP/M 40 COL	FROM: CP/M 80 COL
C128 40 COL	1. Check that **40/80** key is UP. 2. Turn computer ON.		1. Press **ESC** key; release. 2. Press **X** key. **OR** 1. Check that **40/80** key is UP. 2. Press RESET button.	1. Check that **40/80** key is UP. 2. Turn computer OFF, then ON.	1. Check that **40/80** key is UP. 2. Turn computer OFF, then ON.	1. Check that **40/80** key is UP. 2. Turn computer OFF, then ON.
C128 80 COL	1. Press **40/80** key DOWN. 2. Turn computer ON.	1. Press **ESC** key; release. 2. Press **X** key. **OR** 1. Press **40/80** key DOWN. 2. Press RESET button.		1. Press **40/80** key DOWN. 2. Turn computer OFF, then ON.	1. Press **40/80** key DOWN. 2. Remove CP/M system disk from drive, if necessary. 3. Turn computer OFF, then ON.	1. Check that **40/80** key is DOWN. 2. Remove CP/M system disk from drive, if necessary. 3. Turn computer OFF, then ON.
C64	1. Hold C= key DOWN. 2. Turn computer ON. **OR** 1. Insert C64 cartridge. 2. Turn computer ON.	1. Type **GO 64**; press **RETURN.** 2. The computer responds: **ARE YOU SURE?** Type **Y**; press **RETURN.**	1. Type **GO 64**; press **RETURN.** 2. The computer responds: **ARE YOU SURE?** Type **Y**; press **RETURN.**		1. Turn computer OFF. 2. Check that **40/80** key is UP. 3. Hold DOWN C= key while turning computer ON. **OR** 1. Turn computer OFF. 2. Insert C64 cartridge. 3. Turn power ON.	1. Turn computer OFF. 2. Check that **40/80** key is UP. 3. Hold DOWN C= key while turning computer ON. **OR** 1. Turn computer OFF. 2. Insert C64 cartridge. 3. Turn power ON.
CP/M 40 COL	1. Turn disk drive ON. 2. Insert CP/M system disk in drive 3. Check that **40/80** key is UP. 4. Turn computer ON.	1. Turn disk drive ON. 2. Insert CP/M system disk in drive. 3. Check that **40/80** key is UP. 4. Type: **BOOT** 5. Press **RETURN**	1. Turn disk drive ON. 2. Insert CP/M system disk in drive. 3. Check that **40/80** key is UP. 4. Type: **BOOT** 5. Press **RETURN**	1. Check that **40/80** key is UP. 2. Turn disk drive ON. 3. Insert CP/M system disk in drive. 4. Turn computer OFF, then ON.		1. Insert CP/M utilities disk in drive. 2. At screen prompt, **A)** type: **DEVICE CONOUT: = 40 COL** 3. Press **RETURN.**
CP/M 80 COL	1. Turn disk drive ON. 2. Insert CP/M system disk in drive. 3. Press **40/80** key DOWN. 4. Turn computer ON.	1. Turn disk drive ON. 2. Insert CP/M system disk in drive. 3. Press **40/80** key DOWN. 4. Type: **BOOT** 5. Press **RETURN**	1. Turn disk drive ON. 2. Insert CP/M system disk in drive. 3. Check that **40/80** key is DOWN. 4. Type: **BOOT** 5. Press **RETURN**	1. Press **40/80** key DOWN. 2. Turn disk drive ON. 3. Insert CP/M system disk in drive. 4. Turn computer OFF, then ON.	1. Insert CP/M utilities disk in drive. 2. At screen prompt, **A)** type: **DEVICE CONOUT: = 80 COL** 3. Press **RETURN.**	

NOTE: If you are using a Commodore 1902 dual monitor, remember to move the video switch on the monitor from COMPOSITE or SEPARATED to RGBI when switching from 40-column to 80-column display; reverse this step when switching from 80 to 40 columns.

CHAPTER

2

USING C128 MODE



SECTION 3
Getting Started in Basic

BASIC PROGRAMMING LANGUAGE 19
- Direct Mode 19
- Program Mode 19

USING THE KEYBOARD 20
- Keyboard Character Sets 21
- Using the Command Keys 21
- Function Keys 27
- Displaying Graphic Characters 27
- Rules for Typing BASIC Language Programs 27

GETTING STARTED—The PRINT COMMAND 28
- Printing Numbers 28
- Using the Question Mark to Abbreviate the PRINT Command 29
- Printing Text 29
- Printing in Different Colors 30
- Using the Cursor Keys Inside Quotes with the PRINT Command 31

BEGINNING TO PROGRAM 31
- What a Program Is 31
- Line Numbers 31
- Viewing your Program—The LIST Command 32
- A Simple Loop—The GOTO Statement 33
- Clearing the Computer's Memory—The NEW Command 34
- Using Color in a Program 34

EDITING YOUR PROGRAM 35
- Erasing a Line from a Program 35
- Duplicating a Line 35
- Replacing a Line 35
- Changing a Line 35

MATHEMATICAL OPERATIONS 36
- Addition and Subtraction 36
- Multiplication and Division 36
- Exponentiation 37
- Order of Operations 37
- Using Parentheses to Define the Order of Operations 38

CONSTANTS, VARIABLES AND STRINGS 38
- Constants 38
- Variables 39
- Strings 40

SAMPLE PROGRAM 41

STORING AND REUSING YOUR PROGRAMS 41
Formatting a Disk—The HEADER Command 42
SAVEing on Disk 44
SAVEing on Cassette 44
LOADing from Disk 45
LOADing from Cassette 45
Other Disk-Related Commands 46

BASIC Programming Language

The BASIC programming language is a special language that lets you communicate with your Commodore 128. Using BASIC is one means by which you instruct your computer what to do.

BASIC has its own vocabulary (made up of **commands, statements and functions**) and its own rules of structure (called **syntax**). You can use the BASIC vocabulary and syntax to create a set of instructions called a **program,** which your computer can then perform or **"run."**

Using BASIC, you can communicate with your Commodore 128 in two ways: within a program, or directly (outside a program).

Direct Mode

Your Commodore 128 is ready to accept BASIC commands in **direct** mode as soon as you turn on the computer. 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 direct mode commands immediately after you press the RETURN key. Most BASIC commands in your Commodore 128 can be used in direct mode as well as in a program.

Program Mode

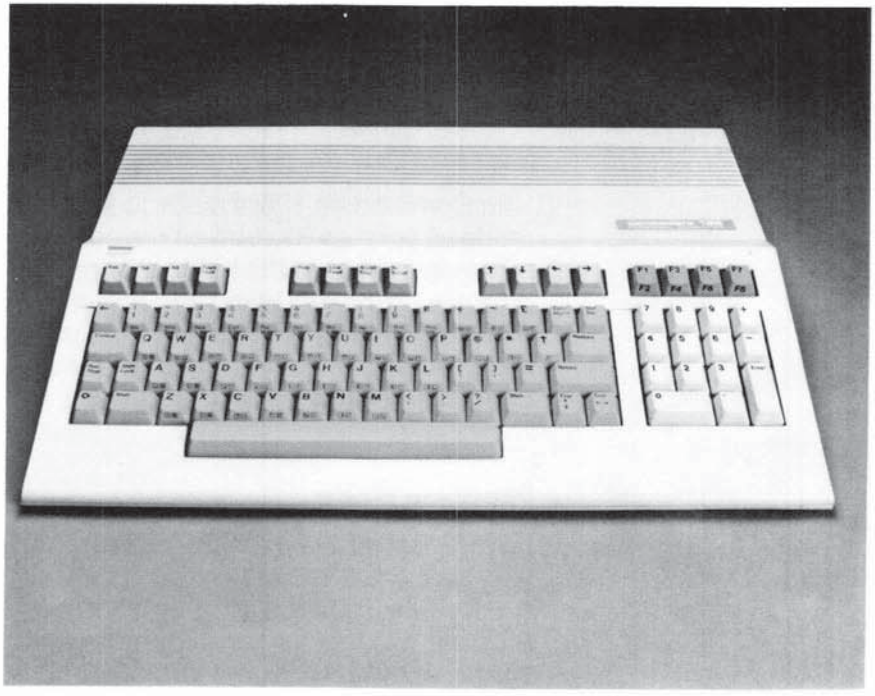
In **program mode** you enter a set of instructions that perform a specific task. Each instruction is contained in a sequential **program** line. A statement in a program may be as long as 160 characters; this is equivalent to four full screen lines in 40-column format, and two full screen lines in 80-column format.

Once you have typed a program, you can use it immediately by typing the RUN command and pressing the RETURN key. You can also store the program on disk or tape by using the DSAVE (or SAVE) command. Then you can recall it from the disk or tape by using the DLOAD (or LOAD) command. This command copies the program from the disk or tape and places that program in the Commodore 128's memory. You can then use or "execute" the program again by entering the RUN command. All these commands are explained later in this section. Most of the time you will be using your computer with programs, including programs you yourself write, and commercially available software packages. The only time you operate in direct mode is when you are manipulating or editing your programs with

commands such as LIST, LOAD, SAVE and RUN. As a rule, the difference between direct mode and operation within a program is that direct mode commands have no line numbers.

Using the Keyboard

Shown below is the keyboard of the Commodore 128 Personal Computer.



Using BASIC is essentially the same in both C64 and C128 modes. Most of the keys, and many of the commands you will learn, can be used to program BASIC in either mode. The keys that are shaded in the diagram above can be used in C64 mode. In C128 mode you can use all of the keys on the keyboard.

Keyboard Character Sets

The Commodore 128 keyboard offers two different sets of characters:

- Upper-case letters and graphic characters
- Upper- and lower case letters

In 80-column format, both character sets are available simultaneously. This gives you a total of 512 different characters that you can display on the screen. In 40 column format you can use only one character set at a time.

When you turn on the Commodore 128 in 40-column format, the keyboard normally is using the upper-case/graphic character set. This means that everything you type is in capital letters. To switch back and forth between the two character sets, press the SHIFT key and the C= key (the COMMODORE key) at the same time. To practice using the two character sets turn on your computer and press several letters or graphic characters. Then press the SHIFT key and the C= (Commodore) key. Notice how the screen changes to upper- and lower-case characters. Press SHIFT and C= again to return to the upper-case and graphic character set.

Using the Command Keys

COMMAND keys are keys that send messages to the computer. Some command keys (such as RETURN) are used by themselves. Other command keys (such as SHIFT, CTRL, C= and RESTORE) are used with other keys. The use of each of the command keys is explained below.

Return

When you press the RETURN key, what you have typed is sent to the Commodore 128 computer's memory. Pressing the RETURN key also moves the cursor (the small flashing rectangle that marks where the next character you type will appear) to the next line.

At times you may misspell a command or type in something the computer does not understand. Then, when you press the RETURN key, you

probably will get a message like SYNTAX ERROR on the screen. This is called an "Error Message." Appendix A lists the error messages and tells how to correct the errors.

NOTE: In the examples given in this book, the following symbol indicates that you must press the RETURN key:

RETURN

Shift

There are two SHIFT keys on the bottom row of the keyboard. One key is on the left and the other is on the right, just as on a standard typewriter keyboard.

The SHIFT key can be used in three ways:

1. With the upper/lower-case character set, the SHIFT key is used like the shift key on a regular typewriter. When the SHIFT key is held down, it lets you print capital letters or the top characters on double-character keys.
2. The SHIFT key can be used with some of the other command keys to perform special functions.
3. When the keyboard is set for the upper-case/graphic character set, you can use the SHIFT key to print the graphic symbols or characters that appear on the front face of certain keys. See the paragraphs entitled "Displaying Graphic Characters" at the end of this section for more details.

Shift Lock

When you press this key down, it locks into place. Then, whatever you type will either be a capital letter, or the top character of a double-character key. To release the lock, press down on the SHIFT LOCK key again.

Moving the Cursor

In C128 mode, you can move the cursor by using either the four arrow keys located just above the top right of the main keyboard, or the two keys labeled CRSR, at the right of the bottom row of the main keyboard.

Using the Four Arrow Cursor Keys

In C128 mode, the cursor can be moved in any direction simply by using the arrow key in the top row that points in the direction you want to move the cursor. (These keys cannot be used in C64 mode).

Using the CRSR keys

In both C128 and C64 mode, you can use the two keys on the right side of the bottom row of the main keyboard to move the cursor:

- Pressing the CRSR key alone moves the cursor **down.**
- Pressing the CRSR and SHIFT keys together moves the cursor **up.**
- Pressing the CRSR key alone moves the cursor **right.**
- Pressing the CRSR and SHIFT keys together moves the cursor **left.**

You don't have to keep tapping a cursor key to move more than one space. Just hold the key down and the cursor continues to move until it reaches the position you want.

Notice that when the cursor reaches the right side of the screen, it "wraps", or starts again at the beginning of the next row. When moving left, the cursor will move along the line until it reaches the edge of the screen, then it will jump up to the end of the preceding line.

You should try to become very familiar with the cursor keys, because moving the cursor makes your programming much easier. With a little practice you will find that you can move the cursor almost without thinking about it.

Inst/Del

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

Inserting Characters

You must use the SHIFT key with the INST/DEL key when you want to insert characters in a line. Suppose you left some characters out of a line, like this:

```
WHILE U WERE OUT
```

To insert the missing characters, first use the cursor keys to move the cursor back to the error, like this:

```
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:

```
WHILE ■  U WERE OUT
```

Notice that INST doesn't move the cursor; it just adds space between the cursor and the character to its right. To make the correction, simply type in the missing "Y" and "O", like this:

```
WHILE YOU WERE OUT
```

Deleting Characters

When you press the DEL key, the cursor moves one space to the left and erases the character that is there. This means that when you want to delete something, you move the cursor just to the right of the character you want to DELete. Suppose you have made a mistake in typing, like this:

```
PRINT "ERROER"
```

You wanted to type the word ERROR, not ERROER. To delete the incorrect E that precedes the final R, position the cursor in the space where the final R is located. When you press the DEL key, the character to the right of the cursor (the R) automatically moves over one space to the left. You now have the correct wording like this:

```
PRINT "ERROR"
```

Using INSerT and DELete Together

You can use the INSerT and DELete functions together to fix incorrect characters. First, move the cursor to the incorrect characters and press the INST/DEL key by itself to delete the characters. Next, press the SHIFT key and the INST/DEL key together to add any necessary space. Then type in the corrections. You can also type directly on top of undesired characters, then use INST to add any needed space.

Control

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

One control function that is used often is setting the character and cursor color. To select a color, hold down the CTRL key while you press a number key (1 through 8), on the top row of the keyboard. There are eight more colors available to you; these can be selected with the C= key, as explained later.

Run/Stop

This is a dual function key. Under certain conditions you can use the RUN function of this key by pressing the SHIFT and RUN/STOP together. It is also possible to use the STOP function of this key to halt a program or a printout by pressing this key while the program is running. How-

ever, in most prepackaged programs, the STOP function of the RUN/STOP key is intentionally disabled (made unusable). This is done to prevent the user from trying to stop a program that is running before it reaches its normal end point. If the user were able to stop the program, valuable data could be lost.

Restore

The RESTORE key is used with the RUN/STOP key to return the computer to its standard condition.

Most prepackaged programs disable the RESTORE key for the same reason they disable the STOP function of the RUN/STOP key: to prevent losing valuable data.

CLR/Home

CLR stands for CLeaR. HOME refers to the upper-left corner of the screen, which is called the HOME position. If you press this key by itself the cursor returns to 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.

Commodore Key (C=)

The C= key (known as the COMMODORE key) has a number of functions, including the following ones:

1. The C= key lets you switch back and forth between the upper/lower-case character set (which displays the letters and characters on the top of the keys), and the upper-case/graphic display character set (which displays capital letters and the graphics symbols on the front face 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 for the cursor. To get these colors, you hold down the C= key while you press a number key (1 through 8) in the top row.

3. If you hold down the C= key while turning on the computer, you can immediately access C64 mode.

Function Keys

The four keys located above the numeric keypad (marked F1, F3, F5 and F7 on the top and F2, F4, F6 and F8 on the front) are called **function keys.** In both C128 and C64 modes, you can program the function keys. (See the KEY command descriptions in Section 5 of Chapter II and in Chapter V, BASIC 7.0 ENCYCLOPEDIA). These keys are also often used by prepackaged software to allow you to perform a task with a single keystroke.

Displaying Graphic Characters

To display the graphic symbol on the right front face of a key, hold down the SHIFT key while you press the key that has the graphic character you want to print. You can display the right side graphic characters only when the keyboard is in the upper-case/graphics character set (one normal character set usually available at power-up).

To display the graphic character on the left front face of a key, hold down the C= key while you press the key that has the graphic character you want. You can display the left graphic character while the keyboard is in either character set.

Rules for Typing BASIC Language Programs

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

1. Spacing between words is not critical; e.g., typing FORT = 1TO10 is the same as typing FOR T = 1 TO 10. However, a BASIC keyword itself must not be broken up by spaces. (See the BASIC 7.0 Encyclopedia in Chapter V for a list of BASIC keywords).
2. Any characters can be typed inside quotation marks. Some characters have special functions when placed inside quotation marks. These functions are explained later in this Guide.

3. Be careful with punctuation marks. Commas, colons and semicolons also have special properties, explained later in this section.
4. Always press the RETURN key (indicated in this Guide by RETURN) after completing a numbered line.
5. Never type more than 160 characters in a program line. Remember, this is the same as four full screen lines in 40-column format, or two full screen lines in 80-column format. See Section 8 for more details on 40- and 80-column formats.
6. Distinguish clearly between the letter I and the numeral 1 and between the letter O and the numeral 0.
7. The computer ignores anything following the letters REM on a program line. REM stands for REMark. You can use the REM statement to put comments in your program that tell anyone listing the program what is happening at a specific point.

Follow these guidelines when you type the examples and programs shown in this section.

Getting Started—The PRINT Command

The PRINT command tells the computer to display information on the screen. You can print both numbers and text (letters), but there are special rules for each case, described in the following paragraphs.

Printing Numbers

To print numbers, use the PRINT command followed by the number(s) you want to print. Try typing this on your Commodore 128:

```
PRINT 5
```

Then press the RETURN key. Notice the number 5 is now displayed on the screen.

Now type this and press RETURN:

```
PRINT 5,6
```

In this PRINT command, the comma tells the Commodore 128 that you want to print more than one number. When the computer finds commas in a string of numbers in a PRINT statement, each number that follows a comma is printed 10 spaces to the right of the preceding number. If you don't want all the extra spaces, use a semicolon (;)

in your PRINT statement instead of a comma. The semicolon tells the computer to place the numbers only three spaces apart. Type these examples and see what happens:

PRINT 5;6 RETURN

PRINT 100;200;300;400;500 RETURN

Using the Question Mark to Abbreviate the PRINT Command

You can use a question mark (?) as an abbreviation for the PRINT command. Many of the examples in this section use the ? symbol in place of the word PRINT. In fact, most of the BASIC commands can be abbreviated. The abbreviations for BASIC commands can be found in Appendix K of this Guide.

Printing Text

Now that you know how to print numbers, it's time to learn how to print text. It's actually very simple. Any words or characters you want to display are typed on the screen, with a quote symbol at each end of the string of characters. **String is the BASIC name for any set of characters surrounded by quotes.** The quote character is obtained by pressing SHIFT and the numeral 2 key on the top row of the keyboard (not the 2 in the numeric keypad). Try these examples:

? "COMMODORE 128" RETURN

? "4*5" RETURN

Notice that when you press RETURN, the computer displays the characters within the quotes on the screen. Also note that the second example did not calculate 4*5 since it was treated as a string and not a mathematical calculation. If you want to calculate the result 4*5, use the following command:

? 4*5 RETURN

You can PRINT any string you want by using the PRINT command and surrounding the printed characters with quotes. You can combine text and calculations in a single PRINT command like this:

? "4*5 = "4*5 RETURN

See how the computer PRINTS the characters 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. Type the following statement:

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

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

? " OVER HERE" RETURN

Printing in Different Colors

The Commodore 128 is capable of displaying 16 different colors on the screen. You can change colors easily. All you do is hold down the CTRL key and press a numbered key between zero and eight on the top row of the main keyboard. 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.

Table 3-1 lists the colors available in C128 mode, for both 40- and 80-column screen formats. The table also shows the key sequence (CONTROL key plus number key, or C= key plus number key) used to specify a given color.

Color Code	Color	Color Code	Color
1	Black	9	Orange
2	White	10	Brown
3	Red	11	Light Red
4	Cyan	12	Dark Gray
5	Purple	13	Medium Gray
6	Green	14	Light Green
7	Blue	15	Light Blue
8	Yellow	16	Light Gray

Color Numbers in 40-Column Format

Color Code	Color	Color Code	Color
1	Black	9	Dark Purple
2	White	10	Dark Yellow
3	Dark Red	11	Light Red
4	Light Cyan	12	Dark Cyan
5	Light Purple	13	Medium Gray
6	Dark Green	14	Light Green
7	Dark Blue	15	Light Blue
8	Light Yellow	16	Light Gray

Color Numbers in 80-Column Format

Using the Cursor Keys Inside Quotes with the PRINT Command

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. Try typing a question mark (?), open quotes (SHIFTed 2 key); then press either of the down cursor keys 10 times, enter the words "DOWN HERE", and close the quotes. The line should look like this:

```
? "QQQQQQQQQQ DOWN HERE"
```

Now press RETURN. The Commodore 128 prints 10 blank lines, and on the eleventh line, it prints "DOWN HERE". As this example shows, you can tell the computer to print anywhere on your screen by using the cursor control keys inside quotation marks.

Beginning to Program

So far most of the commands we have discussed have been performed in DIRECT mode. That is, the command was executed as soon as the RETURN key was pressed. However, most BASIC commands and functions can also be used in programs.

What a Program Is

A program is just a set of numbered BASIC instructions that tells your computer what you want it to do. These numbered instructions are referred to as **statements** or **lines.**

Line Numbers

The lines of a program are numbered so that the computer knows in what order you want them executed or RUN. The computer executes the program lines in numerical order, unless the program instructs otherwise. You can use any whole number from 0 to 63999 for a line number. **Never** use a comma in a line number.

Many of the commands you have learned to use in DIRECT mode can be easily made into program statements. For example, type this:

```
10 ? "COMMODORE 128" RETURN
```

Notice the computer did not display COMMODORE 128 when you pressed RETURN, as it would do if you were using the PRINT command in DIRECT mode. This is because the number, 10, that comes before the PRINT symbol (?) tells the computer that you are entering a BASIC program. The computer just stores the numbered statement and waits for the next input from you.

Now type RUN and press RETURN. The computer prints the words COMMODORE 128. This is not the same as using the PRINT command in DIRECT mode. What has happened here is that YOU HAVE JUST WRITTEN AND RUN YOUR FIRST BASIC PROGRAM as small as it may seem. The program is still in the computer's memory, so you can run it as many times as you want.

Viewing Your Program—The LIST Command

Your one-line program is still in the C128 memory. Now clear the screen by pressing the SHIFT and CLR/HOME keys together. The screen is empty. At this point you may want to see the program listing to be sure it is still in memory. The BASIC language is equipped with a command that lets you do just this—the LIST command.

Type LIST and press RETURN. The C128 responds with:

```
10 PRINT "COMMODORE 128"

READY.
```

Anytime you want to see all the lines in your program, type LIST. This is especially helpful if you make changes, because you can check to be sure the new lines have been registered in the computer's memory. In response to the command, the computer displays the changed version of the line, lines, or program. Here are the rules for using the LIST command.

—To see line N only, type LIST N and press RETURN. Substitute N for the line number you wish to see.

—To see from line N to the end of the program, type LIST N- and press RETURN.

—To see the lines from the beginning of the program to line N, type LIST-N and press RETURN.

—To see from line N1 to line N2 inclusive, type LIST N1-N2 and press RETURN.

A Simple Loop—The GOTO Statement

The line numbers in a program have another purpose besides putting your commands in the proper order for the computer. They serve as a reference for the computer in case you want to execute the command in that line repetitively in your program. You 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 line 20, you add it to your program in the computer's memory.

Notice that we numbered the first line 10 and the second line 20. It is very helpful to number program lines in increments of 10 (that is, 10, 20, 30, 40, etc.) in case you want to go back and add lines in between later on. You can number such added lines by fives (15, 25 . . .) ones (1, 2 . . .)—in fact, by any whole number—to keep the lines in the proper order. (See the RENUMBER and AUTO commands in the BASIC Encyclopedia.)

Type RUN and press RETURN, and watch the words COMMODORE 128 move down your screen. To stop the message from printing on the screen, press the RUN/STOP key on the left side of your keyboard.

The two lines that you have typed make up a simple program that repeats itself endlessly, because the second line keeps referring the computer back to the first line. The program will continue indefinitely unless you stop it or turn off the computer.

Now type **LIST** **RETURN**. The screen should say:

```
10 PRINT "COMMODORE 128"
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 even though you used the ? symbol for the PRINT statement, your computer has converted it into the full command. This happens when you LIST any command you have abbreviated in a program.

Clearing the Computer's Memory—The NEW Command

Anytime you want to start all over again or erase a BASIC program in the computer's memory, just type NEW and press RETURN. This command clears out the computer's BASIC memory, the area where programs are stored.

Using Color in a Program

To select color within a program, you must include the color selection information within a PRINT statement. For example, clear your computer's memory by typing NEW and pressing RETURN, then type the following, being sure to leave space 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 entering the first set of quote marks. 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 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. Press the SHIFT and the 2 keys to type a set of closing quotation marks and press the RETURN key. Now type RUN and press the RETURN key. The computer 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 computer what color you want for each printed letter. Note that these graphic characters do not appear when the Commodore 128 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 128 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.

Editing Your Program

The following paragraphs will help you to type in your programs and make corrections and additions to them.

Erasing a Line from a Program

Use the LIST command to display the program you typed previously. Now type 10 and press 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.

Duplicating a Line

Hold down the SHIFT key and press the CLR/HOME key on the upper right side of your keyboard. This will clear your screen. Now LIST your program. Move the cursor up again so that it is blinking on the "0" in the line numbered 10. Now type a 5 and press RETURN. You have just duplicated (i.e., copied) line 10. The duplicate line is numbered 15. Type LIST and press RETURN to see the program with the duplicated lines.

Replacing a Line

You can replace a whole line by typing in the old line number followed by the text of the new line, then pressing RETURN. The old version of the line will be erased from memory and replaced by the new line as soon as you press RETURN.

Changing a Line

Suppose you want to add something in the middle of a line. Simply move the cursor to the character or space that immediately follows the spot where you want to insert the new material. Then hold down the SHIFT key and the INST/DEL key together until there is enough space to insert your new characters.

Try this example. Clear the computer's memory by typing NEW and pressing RETURN. Then type:

10 ? "MY 128 IS GREAT" **RETURN**

Let's say that you want to add the word COMMODORE in front of the number 128. Just move the cursor so that it is blinking on the "1" in 128. Hold down the SHIFT and INST/DEL keys until you have enough room to type in COMMODORE (don't forget to leave enough room for a space after the E). Then type in the word COMMODORE.

If you want to delete something in a line (including extra blank spaces), move the cursor to the character following the material you want to remove. Then hold down the INST/DEL key by itself. The cursor will move to the left, and characters or spaces will be deleted as long as you hold down the INST/DEL key.

Mathematical Operations

You can use the PRINT command to perform calculations like addition, subtraction, multiplication, division and exponentiation. You type the calculation after the PRINT command.

Addition and Subtraction

Try typing these examples:

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 that the fourth calculation (75-100) resulted in a negative number. Also notice that you can tell the computer to make more than one calculation with a single PRINT command. You can use either a comma or a semicolon in your command, depending on whether or not you want your results printed 10 spaces apart or three spaces apart.

Multiplication and Division

Find the asterisk key (*) on the right side of your keyboard. This is the symbol that the Commodore 128 uses for multiplication. The slash (/) key, located next to the right SHIFT key, is used for division.

Try these examples:

PRINT 5*3 RETURN

PRINT 100/2 RETURN

Exponentiation

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

PRINT 3↑2 RETURN

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 addition or subtraction operations, you may not get the result you expect. For example, type:

PRINT 4 + 6/2 RETURN

If you assumed you were dividing 10 by 2, you were probably surprised when the computer responded with the answer 7. The reason you got this answer is that multiplication and division operations are performed by the computer before addition or subtraction. Multiplication and division are said to take precedence over addition and subtraction. It doesn't matter in what order you type the operation. In computing, the order in which mathematical operations are performed is known as the order of operations.

Exponentiation, or raising a number to a power, takes precedence over the other four mathematical operations. For example, if you type:

PRINT 16/4↑2 RETURN

the Commodore 128 responds with a 1 because it squares the 4 before it divides 16.

Using Parentheses to Define the Order of Operations

You can tell the Commodore 128 which mathematical operation you want performed first by enclosing that operation in parentheses in the PRINT command. For instance, in the first example above, if you want to tell the computer to add before dividing, type:

PRINT (4 + 6)/2 RETURN

This gives you the desired answer, 5.

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

PRINT (16/4)↑2 RETURN

Now you have the expected answer, 16.

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

PRINT 4*5/10*6 RETURN

Since the operations in this example are performed in order from left to right, the result is 12 (4*5 = 20 . . . 20/10 = 2 . . . 2*6 = 12). If you want to divide 4*5 by 10*6 you type:

PRINT (4*5)/(10*6) RETURN

The answer is now .333333333.

Constants, Variables and Strings

Constants

Constants are numeric values that are permanent: that is, they do not change in value over the course of an equation or program. For example, the number 3 is a constant, as is any number. This statement illustrates how your computer uses constants:

```
10 PRINT 3
```

No matter how many times you execute this line, the answer will always be 3.

Variables

Variables are values that can change over the course of an equation or program statement. There is a part of the computer's BASIC memory that is reserved for the characters (numbers, letters and symbols) you use in your program. Think of this memory as a number of storage compartments in the computer that store information about your program; this part of the computer's memory is referred to as variable storage. Type in this 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. The letter X is called a variable, because the value of X varies depending on the value to the right of the equals sign. We call this an assignment statement because now there is a storage compartment 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 storage compartment (a memory location) labeled with the letter X to the left of the equals sign.

The variable name on the left side 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. This means the names PA and PART would refer to the same storage compartment. Also, the words used for BASIC commands (LOAD, RUN, LIST, etc.) or functions (INT, ABS, SQR, etc.) cannot be used as names in your programs. Refer to the BASIC Encyclopedia in Chapter 5 if you have any questions about whether a variable name is a BASIC keyword. Notice that the = in assignment statements is not the same as the mathematical symbol meaning "equals", but rather means allocate a variable (storage compartment) and assign a value to it.

In the sample program you just typed, the value of the variable X remains at 5 throughout. You can put calculations to the right of the = sign to assign the result to a variable. You can mix text with constants in a PRINT statement to identify them. Type NEW and press RETURN to clear the Commodore 128's memory; 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 variables, 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 variable, just put another assignment statement in the program. Add these lines to the program above and RUN it again.

```
50 A=900*30/10
60 B=95+32+128
70 GOTO 30
```

You'll have to press the STOP key to halt the program.

Now 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 replaced and cannot be recovered unless the computer executes lines 10 and 20 again. When the computer is sent to line 30 to begin printing the values of A and B again, it prints the new values calculated in lines 50 and 60. Lines 50 and 60 reassign the same values to A and B and line 70 sends the computer back to line 30. This is called an endless loop, because lines 30 through 70 are executed over and over again until you press the RUN/STOP key to halt the program. Other methods of looping are discussed later in this and the following two chapters.

Strings

A string is a character or group of characters enclosed in quotes. These characters are stored in the computer's memory as a variable in much the same way numeric variables are stored. You can also use variable names to represent strings, just as you use them to represent numbers. When you put the dollar sign ($) after the string variable name, it tells the computer that the name is for a string variable, and not a numeric variable.

Type NEW and press RETURN to clear your computer's memory, then type in the program below:

```
10 A$="COMMODORE "
20 X=128
30 B$=" COMPUTER"
40 Y=1
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.

You can print the value of a variable in DIRECT mode, after the program has been RUN. Type ?A$;B$;X;Y after running the program above and see that those three variable values 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 command. Just type CLR ⟨RETURN⟩ and all constants, variables and strings are erased. But when you type LIST, you can see the program is still in memory. The NEW command discussed earlier erases both the program and the variables.

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 numbers (X, Y and Z) and prints their values and their averages on the screen. You can edit the program and change the calculations in line 10 through 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 to tell the computer to add the numbers before it divides.

> **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 equal.

```
10 X=46
20 Y=72
30 Z=114
40 A=(X+Y+Z)/3
50
60 ?"THE AVERAGE OF"X;Y;"AND "Z;"IS"A;
70 GOTO 90
90 END
```

Storing and Reusing Your Program

Once you have created your program, you will probably want to store it permanently so you will be able to recall and use it at some later time. To do this, you'll need either a Commodore disk drive or the Commodore 1530 Datassette.

You will learn several commands that let you communicate between your computer and your disk drive or Datassette. These commands are constructed with the use of a command word followed by several parameters. Parameters are letters, words or symbols in a command that supply specific information to the computer, such as a filename, or a numeric variable that specifies a device number. Each command may have several parameters. For example, the parameters of the disk format command include a name for the disk and an identifying number or code, plus several other parameters. Parameters are used in almost every BASIC command; some are variables which change and others are constant. These are the parameters that supply disk information to the C128 and disk drive:

Disk Handling Parameters

disk name—	arbitrary 16 character identifying name you supply.
file name—	arbitrary 16 character identifying name you supply.
i.d. number—	arbitrary two-character identifying number you supply
drive number—	must use 0 for a single disk drive, 0 or 1 in a dual drive.
device number—	a preassigned number for a peripheral device. For example, the device number for a Commodore disk drive is 8.

Formatting a Disk—The HEADER Command

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

The formatting process divides the disk into sections called tracks and sectors. A table of contents, called a directory, is created. Each time you store a program on disk, the name you assign to that program will be added to the directory.

The Commodore 128 has two kinds of formatting commands. One can be used only in C128 mode, and one can be used in both C64 and C128 mode. The following paragraphs describe C128 mode format commands here. See Chapter III on C64 mode for more information about C64 programming and disk handling.

The command that formats a diskette is called the HEADER command. It has a long form and a short form. To format a blank (new) disk, you MUST use the long form as follows:

HEADER "diskname", i.d.[,Ddrive number] [,[ON]U device number]

After the word HEADER, you type a name of your choice for the disk, within quotes. You can choose any name with up to 16 characters. You should choose disk names that help you identify what will be stored on the disk.

Follow the diskname with a comma and the letter "I". Now a two character i.d., followed by a comma. Your disk i.d. does not have to be numbers; you can also choose letters. You may want to develop a consecutive coding system for your disks, such as A1, A2, B1, B2.

If you have one single disk drive, just press RETURN at this point since the Commodore 128 automatically assumes the drive number is 0 and the device number is 8. You can specify these parameters if you have more than one drive or a dual drive.

The next parameter in the command selects the drive number. Press the "D" key and if you have a single disk drive, press the zero key followed by a comma. Dual drives are labeled 0 and 1. The device number parameter starts with the letter U so press the "U" key followed by the preassigned device number for a Commodore disk drive which is 8.

Here is an example of the long form of the HEADER command:

HEADER"RECS",IA1,D0,U8 RETURN

This command formats the diskette, calling the directory RECS, the i.d. number A1, on drive 0, unit 8.

The default values for disk drive (0) and device number (8) will be used if none are supplied. This is an acceptable long form of the HEADER command:

HEADER "MYDISK", I23 RETURN

The HEADER command can also be used to erase all data from a used disk, so the disk can be reused as if it were a brand new disk. Be careful that you don't erase a disk that contains data you may want someday.

The quick form of the HEADER command can be used if the disk was previously formatted with the long form of the HEADER command.

The quick form clears the directory, erasing all data in the same way as the long form, but keeps the same i.d. as was previously used. Here is what the quick HEADER might look like:

HEADER "NEWPROGS" RETURN

SAVEing on Disk

In C128 mode, you can store your program on disk by using either of the following commands:

DSAVE"PROGRAM NAME" RETURN
SAVE"PROGRAM NAME",8 RETURN

Either command can be used. Remember that the character sequence "DSAVE" can be displayed on the screen by pressing the function key labeled F5, or you can type the sequence yourself. The program name can be any name you choose, up to 16 characters long. Be sure to enclose the program name in quotes. You cannot put two programs with the same name on the same disk. If you do, the second program will not be accepted; the disk will retain the first one. In the second example, the 8 indicates that you are saving your program on device number 8. You do not need the 8 with DSAVE, because the computer automatically assumes you are using device number 8.

SAVEing on Cassette

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

You must type the word SAVE, followed by the program name. The program name can be any name you choose up to 16 characters.

> **NOTE:** The screen will go blank while the program is being SAVEd, but returns to normal when the process is completed.

Unlike disk, you can save two programs to tape under the same name. However when you load it back into the computer, the first program sequentially on the tape will be loaded, so avoid giving programs the same name.

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

LOADing from Disk

Loading a program simply copies the contents of the program from the disk into the computer's memory. If a BASIC program was already in memory before you issued the LOAD command, it is erased.

To load your BASIC program from a disk, use either of the following commands in C128 mode:

DLOAD"PROGRAM NAME" RETURN
LOAD"PROGRAM NAME",8 RETURN

Remember, in C128 mode you can use the F2 function key (which you activate by pressing SHIFT and F1) to display the sequence DLOAD", or you can type the letters yourself. In the second example, the 8 indicates to the computer that you are loading from device number 8. Again, like DSAVE, DLOAD assumes the disk-drive device number is 8. Be careful to type the program name exactly as you typed it when SAVEing the program, or the computer will respond "FILE NOT FOUND."

Once the program is loaded, type RUN to execute. The Commodore 128 has a special form of the RUN command used to LOAD and RUN the program in C128 mode with one command. Type RUN, followed by the name of the program (also known as the filename) in quotes:

RUN"MYPROG" RETURN

LOADing from Cassette

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 found. While the Datassette is searching for the program the screen is blank. When the program is found, the screen displays

FOUND PROGRAM NAME

To actually load the program, you then press the Commodore key.

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

Other Disk-Related Commands

Verifying a Program

To verify that a program has been correctly saved, use the following command in C128 mode.

DVERIFY"PROGRAM NAME" RETURN

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

The VERIFY command also works for tape programs. You type:

VERIFY"PROGRAM NAME" RETURN

You do not enter the comma and a device number.

Displaying Your Disk Directory

In C128 mode, you can see a list or directory of the programs on your disk by using the following command:

DIRECTORY RETURN

This lists the contents of the directory. The easy way is to press the F3 function key. When you press F3, the C128 displays the word "DIRECTORY" and performs the command.

For further information on SAVEing and LOADing your programs, or other disk related information, refer to your Datassette or disk drive manual. Also consult the LOAD and SAVE command descriptions in the Chapter V, BASIC 7.0 Encyclopedia.

****

You now know something about the BASIC language and some elementary programming concepts. The next section builds on these concepts, introducing additional commands, functions and techniques that you can use to program in BASIC.

SECTION 4
Advanced BASIC Programming

COMPUTER DECISIONS—The IF-THEN Statement 51
Using the Colon 52

LOOPS—The FOR-NEXT Command 53
Empty Loops—Inserting Delays in a Program 54
The STEP Command 54

INPUTTING DATA 55
The INPUT Command 55
Assigning a Value to a Variable 55
Prompt Messages 56
The GET Command 57
Sample Program 58
The READ-DATA Command 59
The RESTORE Command 60
Using Arrays 61
Subscripted Variables 61
Dimensioning Arrays 62
Sample Program 63

PROGRAMMING SUBROUTINES 64
The GOSUB-RETURN Command 64
The ON GOTO/GOSUB Command 65

USING MEMORY LOCATIONS 65
Using PEEK and POKE for RAM Access 65
Using PEEK 66
Using POKE 66

BASIC FUNCTIONS 67
What Is a Function? 67
The INTEGER Function (INT) 67
Generating Random Numbers—The RND Function 68
The ASC and CHR$ Commands 69
Converting Strings and Numbers 69
The VAL Function 70
The STR$ Function 70
The Square Root Function (SQR) 70
The Absolute Value Function (ABS) 70

THE STOP AND CONT (CONTINUE) COMMANDS 70

This section describes how to use a number of powerful BASIC commands, functions and programming techniques that can be used in both C128 and C64 modes.

These commands and functions allow you to program repeated actions through looping and nesting techniques; handle tables of values; branch or jump to another section of a program, and return from that section; assign varying values to a quantity—and more. Examples and sample programs show just how these BASIC concepts work and interact.

Computer Decisions—The IF-THEN Statement

Now that you know how to change the values of variables, the next step is to have the computer make decisions based on these updated values. You do this with the IF-THEN statement. You tell the computer to execute a command only IF a condition is true (e.g., IF X = 5). The command you want the computer to execute when the condition is true comes after the word THEN in the statement. Clear your computer's memory by typing NEW and pressing RETURN, then type this program:

```
10 J=0
20 ? J,"COMMODORE 128"
30 J=J+1
40 IF J=5 THEN GOTO 60
50 GOTO 20
60 END
```

You 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 128" and incrementing (increasing) J until J = 5 is true. When an IF condition is false, the computer jumps to the next line of the program, no matter what comes after the word THEN.

Notice the END command in line 60. It is good practice to put an END statement as the last line of your program. It tells the computer where to stop executing statements.

Below is a list of comparison symbols that may be used in the IF statement and their meanings:

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

You should be aware that these comparisions work in expected mathematical ways with numbers. There are different ways to determine if one string is greater than, less than, or equal to another. You can learn about these "string handling" functions by referring to Chapter V, BASIC 7.0 Encyclopedia.

Section 5 describes some powerful extensions of the IF-THEN concept, consisting of BASIC 7.0 commands like BEGIN, BEND, and ELSE.

Using the Colon

A very useful tool in programming is the colon (:). You can use the colon to separate two (or more) BASIC commands on the same line.

Statements after a colon on a line will be executed in order, from left to right. In one program line you can put as many statements as you can fit into 160 characters, including the line number. This is equivalent to four full screen lines in 40-column format, and two full lines in 80-column format. This provides an excellent opportunity to take advantage of the THEN part of the IF-THEN statment. 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 N=1
20 IF N<5 THEN PRINT N;"LESS THAN 5":GOTO 10
30 ? N; "GREATER THAN OR EQUAL TO 5"
40 END
```

Now change line 10 to read N = 20, and RUN the program again. Notice you can tell the computer to execute more than one statement when N is less than 5. You can put any statement(s) you want after the THEN command. Remember that the GOTO 10 will not be reached until N<5 is true. Any command that should be followed **whether or not the specified condition is met** should appear on a separate line.

Loops—The FOR-NEXT Command

In the program used for the IF-THEN example, we made the computer print Commodore five times by telling it to increase or "increment" the variable J by units of one, until the value of J equalled five; then we ended the program. There is a simpler way to do this in BASIC. We can use a FOR-NEXT loop, like this:

```
10 FOR J=1 TO 5
20 ? "Commodore"
30 NEXT J
40 END
```

Type and 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 programming tool. You can specify the number of times the computer should repeat an action. 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 in line 10 tells the computer to execute all statements between the FOR statement and the NEXT statement, until J is equal to 5. In this case there is just one statement—the PRINT statement.

After the computer assigns a value of 1 to J, it compares 1 to 5 to see if J = 5 is true—in much the same way as the IF-THEN statement does. Since J = 5 is not true yet, the computer continues with the program by executing the PRINT statement. The computer then goes to the NEXT J statement, which says to go back to the FOR statement. The FOR statement tells the computer to increment J, called a counter variable, by 1; compare J to 5; then 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. In this case the following statement is the END command, so the program stops running.

Empty Loops—Inserting Delays in a Program

Before you proceed any further, it will be helpful to understand about loops and some 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 speed with which the computer executes commands). See if you can predict what this program will do before you run it.

```
10 A$ = "COMMODORE C128"
20 FOR J = 1 TO 20
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 and 50 tells the computer to count to 1500 before executing the remainder of the program. This is known as a delay loop and is often useful. Because it is inside the main loop of the program, it is called a nested loop. Nested loops can be very useful when you want the computer to perform a number of tasks in a given order, and repeat the entire sequence of commands a certain number of times.

Section 5 describes an advanced way to insert delays through use of the new BASIC 7.0 command, SLEEP.

The STEP Command

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

```
10 FOR X = 0 TO 100 STEP 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 later in this section. Also, note that you do not have to increase (or "increment") your counter—you can decrease (or "decrement") it as well. For example, change line 10 in the program above to read:

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

The computer will count backward from 100 to 0, in units of 10.

If you don't use a STEP command with a FOR statement, the computer will automatically increment the counter by units of 1.

The parts of the FOR-NEXT command are:

FOR — word used to indicate beginning of loop
X — counter variable; any number variable can be used
1 — starting value; may be any number, positive or negative
TO — connects starting value to ending value
100 — ending value; may be any number, positive or negative
STEP — indicates an increment other than 1 will be used
2 — increment; can be any number positive or negative

Inputting Data

The INPUT Command

Assigning a Value to a Variable

Clear the computer's memory by typing NEW and pressing RETURN, and then type and RUN this program.

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

In this program 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. You have to do this when you are typing the program, before it is RUN. 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?

In other words, you want to be able to change the value of the variable 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. For example, replace line 10 in the program 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 enter what you want the value of K to be. Type 15 and press RETURN. The computer will execute the loop 15 times.

Prompt Messages

You can also 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 the message to be printed in quotes. This message is called a prompt. Also, notice that you must use a semicolon between the ending quote marks of the prompt and the K. You may put any message you want in the prompt, but the INPUT statement must fit on two screen lines, just as any BASIC command must.

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 by typing NEW and pressing RETURN. Then type in this program.

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

Now RUN the program. When the computer prompts "WHAT IS YOUR NAME?", type your name. Don't forget to press RETURN after you type your name.

Once the value of a variable (numeric or string) has been inserted into a program through the use of INPUT, you can refer to it by its variable name any time in the program. Type ?N$ ⟨RETURN⟩—your computer remembers your name.

The GET Command

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

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

When you type RUN and press RETURN, nothing seems to happen. The reason is that the computer is waiting for you to press a key. The GET command, in effect, tells the computer to check the keyboard and find out what character or key is being pressed. The computer is satisfied with a null character (that is, no character). This is the reason for line 20. This line tells the computer that if it gets a null character, indicated by the two double quotes with no space between them, it should go back to line 10 and try to GET another character. This loop continues until you press a key. The computer then assigns the character on that key to A$.

The GET command is very important because you can use it, in effect, to program a key on your keyboard. The example below prints a message on the screen when Q is pressed. Type the program and RUN it. Then press Q and see what happens.

```
10 ?"PRESS Q TO VIEW MESSAGE"
20 GET A$
30 IF A$ = "" 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 not display the message, but will go back to line 20 to GET another character.

Section 5 describes how to use the DO/LOOP and GETKEY statements, which are new and more powerful BASIC 7.0 commands that can be used to perform a similar task.

Sample Program

Now that you know how to use the FOR-NEXT loop and the INPUT command, clear the computer's memory by typing NEW RETURN, then type the following program:

```
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
```

This program lets you tell the computer how many numbers you want to average. You can change the numbers every time you run the program without having to change the program itself.

Let's see what the program does, line by line:

Line 10 assigns a value of 0 to T (which will be the running total of the numbers).

Line 20 lets you determine how many numbers to average, stored in variable N.

Line 30 tells the computer to execute a loop N 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 (N) after the loop has been executed N times.

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.

The READ-DATA Command

There is another powerful 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. The program uses the variable T to keep a running total, and calculates the average in the same way as the INPUT average program. The READ-DATA average program, however, finds the numbers to average on a DATA line. Notice line 30, READ X. The READ command tells the computer there must be a DATA statement in the program. It finds the DATA line, and uses the first number as the current value for the variable X. The next time through the loop the second number in the DATA statement will be used as the value for X, and so on.

You can put any number you want in a DATA statement, but you cannot 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. Be sure to separate your data items with commas, but be sure not to put a comma between the word DATA and the first number in the list.

If you have more than one DATA statement in your program, the computer will refer to the one that is closest after the READ statement being executed at the time. 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. What happened is that when the computer executed the loop for the fifth time, there was no data for it to read. That is what the error message is telling you. Putting too much into the DATA statement doesn't create a problem because the computer never realizes the extra data exists.

The RESTORE Command

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 RUN the program. The program will run continuously using the same DATA statement. NOTE: If the computer gives you an OUT OF DATA ERROR message, it is because you forgot to replace the number that you removed previously from the DATA statement, so the data is all used before the READ statement has been executed the specified number of times.

You can use DATA statements to assign values to string variables. The same rules apply as for numeric data. Clear the computer's memory and type the following program:

```
10 FOR J=1 TO 3
20 READ A$
30 ? A$
40 NEXT
50 END
60 DATA COMMODORE,128,COMPUTER
```

If the READ statement calls for a string variable, you can place letters or numbers in the DATA statement. Notice however, that since the computer is READing a string, numbers will be stored as a string of characters, not as a value which can be manipulated. Numbers stored as strings can be printed, but not used in calculations. Also, you cannot place letters in a DATA statement if the READ statement calls for a number variable.

Using Arrays

You have seen how to use READ-DATA to provide many values for a variable. But 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, 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.

Subscripted Variables

If the array contains all of the values assigned to the variable X in the READ-DATA example, it is called the X array. The first value assigned to X in the program is named X(1), the second value is 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. The following is another version of the averaging program, this time 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 the loop counter (J). Line 40 calculates the total, just as 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 128 COMPUTER READ-DATA program so the computer will remember the elements in the A$ array.

```
5 DIM A$(3)
10 FOR J=1 TO 3
20 READ A$(J)
30 ? A$(J)
40 NEXT
50 END
60 DATA COMMODORE,C128,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.

Dimensioning Arrays

Arrays can be used with nested loops, so the computer can handle data in a more advanced way. What if you had a large chart with 10 rows and 5 numbers in each row. Suppose you wanted to find the average of the five numbers in each row. You could create 10 arrays and have the computer calculate the average of the five numbers in each one. This is not necessary, because you can put all 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 will still reserve enough space for all of the positions in the array.

Sample Program

Now it becomes very easy to refer to any number in the chart by its column and row position. Refer to the chart 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 DIMX(10,5),A(10)
20 FOR R=1 TO 10
30 T=0
35 FOR C=1 TO 5
40 READ X(R,C)
50 T=T+X(R,C)
60 NEXT C
70 A(R)=T/5
80 NEXT R
90 FOR R=1 TO 10
100 PRINT "ROW #";R
110 FOR C=1 TO 5
120 PRINT X(R,C):NEXT C
130 PRINT "AVERAGE = ";A(R)
140 FOR D=1 TO 1000:NEXT
150 NEXT R
160 DATA 1,3,5,7,9
170 DATA 2,4,6,8,10
180 DATA 5,10,15,20,25
190 DATA 10,20,30,40,50
200 DATA 20,40,60,80,100
210 DATA 30,60,90,120,150
220 DATA 40,80,120,160,200
230 DATA 50,100,150,200,250
240 DATA 100,200,300,400,500
250 DATA 500,1000,1500,2000,2500
260 END
```

The GOSUB-RETURN Command

Until now, the only method you have had to tell the computer to jump to another part of your program is to use the GOTO command. What if you want the computer to jump to another part of the program, execute the statements in that section, then return to the point it left off and continue executing the program?

The part of 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 will square the numbers you type and print 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 command. The RETURN statement tells the computer to go back in the program to the line immediately following the GOSUB command 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-NEXT and IF-THEN), otherwise the computer will give an error message.

The ON GOTO/GOSUB Command

There is another way to make the computer jump to another section of your program (called branching). 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 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 ONE":RETURN
200 ? "YOUR NUMBER WAS TWO":RETURN
300 ? "YOUR NUMBER WAS THREE":RETURN
400 ? "YOUR NUMBER WAS FOUR":RETURN
500 ? "YOUR NUMBER WAS FIVE":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.

Using Memory Locations

Using PEEK and POKE for RAM/ROM Access

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 part of memory, called RAM, is 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 locations that controls the music features of the computer.

There are two BASIC commands—PEEK and POKE—that you can use to access and manipulate the computer's memory. Use of PEEK and POKE commands can be a powerful programming device because the contents of the computer's memory locations determine exactly what the computer should be doing at a specific time.

Using PEEK

PEEK 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(2594) RETURN
? P RETURN
```

The computer assigns the value in memory location 2594 to the variable P when you press RETURN after the first line. Then it prints the value when you press RETURN after entering the ? P command. Memory location 2594 determines whether or not keys like the spacebar and CRSR repeat when you hold them down. A 128 in location 2594 tells the computer to repeat these keys when you hold them down. Hold down the spacebar and watch the cursor move across the screen.

Using POKE

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

```
POKE 2594,96
```

The computer stores the value after the comma (96) in the memory location before the comma (2594). A 96 in memory location 2594 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. The cursor moves one position to the right, but it does not repeat. To return your computer to its normal state, type:

```
POKE 2594,128 RETURN
```

You cannot alter the value of all the memory locations in the computer—the values in ROM can be read, but not changed.

NOTE: These examples assume you are in bank Ø. See the description of the BANK command in Chapter V, BASIC 7.0 Encyclopedia for details on banks. Refer to the Commodore 128 Programmer's Reference

Guide for a complete memory map of the computer, which shows you the contents of all memory locations.

Basic Functions

What Is a Function?

A function is predefined operation of the BASIC language that generally provides you with a single value. When the function provides the value, it is said to "return" the value. For instance, the SQR (square) function is a mathematical function that returns the value of a specific number when it is raised to the second power—i.e., squared.

There are two kinds of functions:

Numeric—returns a result which is a single number. Numeric functions range from calculating mathematical values to specifying the numeric value of a memory location.

String—returns a result which is a character.

Following are descriptions of some of the more commonly used functions. For a complete list of BASIC 7.0 functions see Chapter V, BASIC 7.0 Encyclopedia.

The INTEGER Function (INT)

What if you want to round off a number to the nearest integer? You'll need to use INT, the integer function. The INT function takes away everything after the decimal point. Try typing these examples:

```
? INT(4.25) RETURN
? INT(4.75) RETURN
? INT(SQR(50)) RETURN
```

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

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

The computer added 0.5 to 4.75 before it executed the INT function, so that it rounded 5.25 down to 5 for the result. If you want to round off the result of a calculation, do this:

```
? INT((100/6)+0.5) RETURN
```

You can substitute any calculation for the division shown in the inner parentheses.

What if you want to round off numbers to the nearest 0.01? Instead of adding 0.5 to your number, add 0.0005, 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 use the INT function to get rid of everything after the decimal point (which moves two places to the right when you multiply by 100). You are left with:

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

which gives you a value of 288. All that's left to do is divide by 100 to get the value of 2.88, which is the answer you want. Using this technique, you can round off calculations like the following to the nearest 0.01:

```
? INT((2.876+1.29+16.1-9.534)+0.005)*100/100 RETURN
```

Generating Random Numbers—The RND Function

The RND functions tells the computer to generate a random number. This can be useful in simulating games of chance, and in creating interesting graphic or music programs. All random (RND) numbers are nine digits, in decimal form, between the values 0.000000001 and 0.999999999. Type:

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

Multiplying the randomly generated number by six makes the range of generated numbers increase to greater than 0 and less than 6. In order to include 6 among the numbers generated, we add one to the result of RND(0)*6. This makes the range 1<X<7. If we use the INT function to eliminate the decimal places, the command will generate whole numbers from 1 to 6. This process can be used to simulate the rolling of a die. Try this program:

```
10 R= INT(RND(1)*6+1)
20 ? R
30 GOTO 10
```

Each number generated represents one toss of a die. To simulate a pair of dice, use two commands of this nature. Each number is generated separately, and the sum of the two numbers represents the total of the dice.

The DO/LOOP statement described in Section 5 provides another way to generate random numbers.

The ASC and CHR$ Functions

Every character that the Commodore 128 can display (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 128. There are two functions associated with this concept that are 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 character for Q in the command above to find out the Commodore ASCII code number for any character.

The second function is the CHR$ function. Type:

CHR$(81) **RETURN**

The computer responds with Q. In effect, the CHR$ function is the opposite of the ASC function. They both refer to the table of character string codes in the computer's memory. CHR$ values can be used to program function keys. See Section 5 for more information about this use of CHR$. See Appendix E of this Guide for a full listing of ASC and CHR$ codes.

Converting Strings and Numbers

Sometimes you may 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 from numeric to string type and vice versa.

The VAL Function

The VAL function returns a numeric value for a string argument. Clear the computer's memory and type this program:

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

The STR$ Function

The STR$ function returns the string representation of a numeric value. Clear the computer's memory and type this program.

```
10 A = 65
20 A$ = STR$(A)
30 ? A" IS THE VALUE OF";A$
```

The Square Root Function (SQR)

The square root function is SQR. For example, to find the square root of 50, type:

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

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

The Absolute Value Function (ABS)

The absolute value function (ABS) is very useful in dealing with negative numbers. You can use this function to get the positive value of any number—positive or negative. Try these examples:

```
? ABS( – 10) RETURN
? ABS(5)" IS EQUAL TO "ABS( – 5) RETURN
```

The STOP and CONT (Continue) Commands

You can make the computer stop a program, and resume running it when you are ready. The STOP command must be included in the program. You can put a STOP statement anywhere you want to in a program. When the computer "breaks" from the program (that is, stops running the program), you can use DIRECT mode commands to find out exactly what is going on in the program. For example, you can find the value of a loop counter or other variable. This is a powerful device when you are "debugging" or fixing your program. Clear the computer's memory and type 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 ? "STOP AND CONTINUE"
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. Often when you are debugging 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.

Once you have all the information you need, you can type CONT (for CONTinue) and press RETURN assuming you have not edited anything on the screen. The computer then CONTinues with the program, starting with the statement after the STOP command.

****

This section and the preceding one have been designed to familiarize you with the BASIC programming language and its capabilities. The remaining four sections of this chapter describe commands that are unique to Commodore 128 mode. Some Commodore 128 mode commands provide capabilities that are not available in C64 mode. Other Commodore 128 mode commands let you do the same thing as certain C64 commands, but more easily. Remember that more information on every command and programming technique in this book can be found in the Commodore 128 Programmer's Reference Guide. The syntax for all Commodore 7.0 commands is given in Chapter V, BASIC 7.0 Encyclopedia.

SECTION 5
Some BASIC Commands and Keyboard Operations Unique to C128 Mode

INTRODUCTION 75

ADVANCED LOOPING 75
The DO/LOOP Statement 75
Until 75
While 76
Exit 76
The ELSE Clause with IF-THEN 77
The BEGIN/BEND Sequence with IF-THEN 77
The SLEEP Command 78

FORMATTING OUTPUT 78
The PRINT USING Command 78
The PUDEF Command 79

SAMPLE PROGRAM 79

INPUTTING DATA WITH THE GETKEY COMMAND 80

PROGRAMMING AIDS 81
Entering Programs 81
AUTO 81
RENUMBER 81
DELETE 82
Identifying Problems in Your Programs 83
HELP 83
Error Trapping—The TRAP Command 83
Program Tracing—The TRON and TROFF Commands 85

WINDOWING 86
Using the WINDOW Command to Create a Window 86
Using the ESC key to Create a Window 87

2 MHZ OPERATION 89
The FAST and SLOW Commands 89

KEYS UNIQUE TO C128 MODE 89
Function Keys 89
Redefining Function Keys 90
Other Keys Used in C128 Mode Only 90
HELP 90
NO SCROLL 91
CAPS LOCK 91
40/80 DISPLAY 91
ALT 91
TAB 92
LINE FEED 92

Introduction

This section introduces you to some powerful BASIC commands and statements that you probably haven't seen before, even if you are an experienced BASIC programmer. If you're familiar with programming in BASIC, you've probably encountered many situations in which you could have used these commands and statements. This section explains the concepts behind each command and gives examples of how to use each command in a program. (A complete list and an explanation of these commands and statements may be found in Chapter V, BASIC 7.0 Encyclopedia.) This section also describes how to use the special keys that are available to you in C128 mode.

Advanced Looping

The DO/LOOP Statement

The DO/LOOP statement provides more sophisticated ways to create a loop than do the GOTO, GOSUB or FOR/NEXT statements. The DO/LOOP statement combination brings to the BASIC language a very powerful and versatile technique normally available only in structured programming languages. We'll discuss just a few possible uses of DO/LOOP in this explanation.

If you want to create an infinite loop, you start with a DO statement, then enter the line or lines that specify the action you want the computer to perform. Then end with a LOOP statement, like this:

```
100 DO
110 PRINT "REPETITION"
120 LOOP
```

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

The directions following the DO statement are carried out until the program reaches the LOOP statement (line 120); control is then transferred back to the DO statement (line 100). Thus, whatever statements are in between DO and LOOP are performed indefinitely.

Until

Another useful technique is to combine the DO/LOOP with the UNTIL statement. The UNTIL statement sets up a condition that directs the loop. The loop will run continually unless the condition for UNTIL happens.

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

The DO/LOOP statement is often used to repeat an entire routine indefinitely in the body of a program, as in the following:

```
10 PRINT "PROGRAM CONTINUES UNTIL
YOU TYPE 'QUIT'"
20 DO UNTIL A$ = "QUIT"
30 INPUT "DEGREES FAHRENHEIT";F
40 C = (5/9)*(F - 32)
50 PRINT F;" DEGREES FAHRENHEIT
EQUALS ";C; " DEGREES CELSIUS"
60 INPUT "AGAIN OR QUIT";A$
70 LOOP
80 END
```

Another use of DO/LOOP is as a counter, where the UNTIL statement is used to specify a certain number of repetitions.

```
10 N = 2*2
20 PRINT"TWO DOUBLED EQUALS"; N
30 DO UNTIL X = 25
40 X = X + 1
50 N = N*2
60 PRINT"DOUBLED";X + 1;"TIMES . . . ";N
70 LOOP
80 END
```

Notice that if you leave the counter statement out (the UNTIL X = 25 part in line 30), the number is doubled indefinitely until an OVERFLOW error occurs.

While

The WHILE statement works in a similar way to UNTIL, but the loop is repeated only while the condition is in effect, such as in this reworking of this brief program:

```
100 DO: INPUT "DO YOU LIKE YOUR
COMPUTER";A$
110 LOOP WHILE A$ <> "YES"
120 PRINT "THANK YOU"
```

Exit

An EXIT statement can be placed within the body of a DO/LOOP. When the EXIT statement is encountered, the program jumps to the next statement following the LOOP statement.

The ELSE Clause with IF-THEN

The ELSE clause provides a way to tell the computer how to respond if the condition of the IF-THEN statement is false. Rather than continuing to the next program line, the computer will execute the command or branch to the program line mentioned in the ELSE clause. For example, if you wanted the computer to print the square of a number, you could use the ELSE clause like this:

```
10 INPUT "TYPE A NUMBER TO BE SQUARED";N
20 IF N< 100 THEN PRINT N*N: ELSE 40
30 END
40 ?"NUMBER MUST BE < 100": GOTO 10
```

Notice that you must use a colon between the IF-THEN statement and the ELSE clause.

The BEGIN/BEND Sequence with IF-THEN

BASIC 7.0 allows you to take the IF-THEN condition one step further. The BEGIN/BEND sequence permits you to include a number of program lines to be executed if the IF condition is true, rather than one simple action or GOTO. The command is constructed like this:

```
IF condition THEN BEGIN:
   (program lines):
   BEND:ELSE
```

Be sure to place a colon between BEGIN and any instructions to be computer, and again between the last command in the sequence and the word BEND. BEGIN/BEND can be used without an ELSE clause, or can be used following the ELSE clause when only a single command follows THEN. Try this program:

```
10 INPUT A
20 IF A< 100 THEN BEGIN: ? "YOUR NUMBER WAS ";A
30 SLEEP 2:REM DELAY
40 FOR X=1 TO A
50 ?"THIS IS AN EXAMPLE OF BEGIN/BEND"
60 NEXT X
70 ?"THAT'S ENOUGH":BEND:ELSE ?"TOO MANY"
80 END
```

This program asks for a number from the user. IF the number is less than 100, the statements between the keywords BEGIN and BEND are performed, along with any statements on the same line as BEND (except for ELSE). The message "YOUR NUMBER WAS N" appears on the screen. Line 30 is a delay loop used to keep the message on

the screen long enough so it can be read easily. Then a FOR/NEXT loop is used to display a message for the number of times specified by the user. If the number is greater than 100, the THEN condition is skipped, and the ELSE condition (printing "TOO MANY") is carried out. The ELSE keyword must be on the same line as BEND.

The SLEEP Command

Note the use of the SLEEP command in line 30 of the program just discussed. SLEEP provides an easier, more accurate way of inserting and timing a delay in program operation. The format for the SLEEP command is

SLEEP n

where n indicates the number of seconds, in the range 1 to 65535, that you want the program to delay. In the command shown in line 30, the 2 specifies a delay of two seconds.

Formatting Output

The PRINT USING Command

Suppose you were writing a sales program that calculated a dollar amount. Total sales divided by number of salespeople equals average sales. But performing this calculation might result in dollar amounts with four or five decimal places! You can format the results the computer prints so that only two decimal places are displayed. The command which performs this function is PRINT USING.

PRINT USING lets you create a format for your output, using spaces, commas, decimal points and dollar signs. Hash marks (the # sign) are used to represent spaces or characters in the displayed result. For example:

PRINT USING "#$#####.##";A

tells the computer that when A is printed, it should be in the form given, with up to five places to the left of the decimal point, and two places to the right. The hash mark in front of the dollar sign indicates that the $ should float; that is, it should always be placed next to the left-most number in the format.

If you want a comma to appear before the last three dollar places, as in $1,000.00, include the comma in the PRINT USING statement. Remember you can format output with spaces, commas, decimal points, and dollar signs. There are several other special characters for PRINT USING, see the BASIC Encyclopedia for more information.

The PUDEF Command

If you want formatted output representing something other than dollars and cents, use the PUDEF (Print Using DEFine) command. You can replace any of four format characters with any character on the keyboard.

The PUDEF command has four positions, but you do not have to redefine all four. The command looks like this:

```
PUDEF" _ , . $ "
       1 2 3 4
```

Here:

- position 1 is the filler character. A blank will appear if you do not redefine this position.
- position 2 is the comma character. Default is the comma.
- position 3 is the decimal point.
- position 4 is the dollar sign.

If you wrote a program that converted dollar amounts to English pounds, you could format the output with these commands:

```
10 PUDEF "  £"
20 PRINT USING "#$####.##";X
```

Sample Program

This program calculates interest and loan payments, using some of the commands and statements you just learned. It sets a minimum value for the loan using the ELSE clause with an IF-THEN statement, and sets up a dollar and cents format with PRINT USING.

```
10 INPUT "LOAN AMOUNT IN DOLLARS";A
20 IF A< 100 THEN 80: ELSE P=.15
30 I=A*P
40 ?"TOTAL PAYMENT EQUALS";
50 PRINT USING "#$#####.##";A+I
60 GOTO 80
70 ?"LOANS OF UNDER $100 NOT AVAILABLE"
80 END
```

Inputting Data with the GETKEY Command

You have learned to use INPUT and GET commands to enter DATA during a program. Another way for you to enter data while a program is being RUN is with the GETKEY statement. The GETKEY statement accepts only one key at a time. GETKEY is usually followed by a string variable (A$, for example). Any key that is pressed is assigned to that string variable. GETKEY is useful because it allows you to enter data one character at a time wihtout having to press the RETURN key after each character. The GETKEY statement may only be used in a program.

Here is an example of using GETKEY in a program:

```
1000 PRINT "PLEASE CHOOSE A, B, C, D, E, OR F"
1010 GETKEY A$
1020 PRINT A$;" WAS THE KEY YOU PRESSED."
```

The computer waits until a single key is pressed; when the key is pressed, the character is assigned to variable A$, and printed out in line 1020. The following program features GETKEY in more complex and useful fashions: for answering a multiple-choice question and also asking if the question should be repeated. If the answer given is incorrect, the user has the option to try again by pressing the "Y" key (line 80). The key pressed for the multiple choice answer is assigned to variable A$ while the "TRY AGAIN" answer is assigned to B$, through the GETKEY statements in lines 60 and 90. IF/THEN statements are used for loops in the program to get the proper computer reaction to the different keyboard inputs.

```
10 PRINT "WHO WROTE "THE RAVEN?""
20 PRINT "A. EDGAR ELLEN POE"
30 PRINT "B. EDGAR ALLAN POE"
40 PRINT "C. IGOR ALLEN POE"
50 PRINT "D. ROB RAVEN"
60 GETKEY A$
70 IF A$ = "B" THEN 150
80 PRINT "WRONG. TRY AGAIN? (Y OR N)"
90 GETKEY B$
100 IF B$ = "Y" THEN PRINT "A,B,C, OR D?":GOTO 60
110 IF B$ = "N" THEN 140
120 PRINT "TYPE EITHER Y OR N—TRY AGAIN"
130 GOTO 90
140 PRINT "THE CORRECT ANSWER IS B."
145 GOTO 160
150 PRINT "CORRECT!"
160 END
```

GETKEY is very similar to GET, except GETKEY will automatically wait for a key to be pressed.

Programming Aids

In earlier sections, you learned how to make changes in your programs, and correct typing mistakes with INST/DEL. BASIC also provides other commands and functions which help you locate actual progam errors, and commands which you can use to make programming sessions flow more smoothly.

Entering Programs

Auto

C128 BASIC provides an auto-numbering process. You determine the increment for the line numbers. Let's say you want to number your program in the usual manner, by tens. Before you begin to program, while in DIRECT mode, type:

AUTO 10 RETURN

The computer will automatically number your program by tens. When you press the RETURN key, the next line number appears, and the cursor is in the correct place for you to type the next statement. You can choose to have the computer number the commands with any increment; you might choose 5 or even 50. Just place the number after the word AUTO and press RETURN. To turn off the auto-numbering feature, type AUTO with no increment, and press RETURN.

Renumber

If you write a program and later add statements to it, sometimes the line numbering can be awkward. Using the RENUMBER command you can change the line numbers to an even increment for part or all of your program. The RENUMBER command has several optional parameters, as listed below in brackets:

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

The new starting line is what the first program line will be numbered after the RENUMBER command is used. If you don't specify, the default is 10. The increment is the spacing between line numbers, and it also defaults to 10. The old starting line number is the line number where renumbering is to begin. This feature allows you to renumber a portion of your program, rather than all of it. It defaults to the first line of the program. For example,

RENUMBER 40,,80

tells the computer to renumber the program starting at line 80, in increments of 10. Line 80 becomes line 40.

Notice that this command, like AUTO, can only be executed in DIRECT mode.

Delete

You know to delete program lines by typing the line number and pressing the RETURN key. This can be tedious if you want to erase an entire portion of your program. The DELETE command can save you time because you can specify a range of program lines to erase all at once. For example,

DELETE 10—50

will erase lines 10, 50, and any in between. The use of DELETE is similar to that of LIST, in that you can specify a range of lines up to a given line, or following it, or a single line only, as in these examples:

DELETE—120
erases all lines up to and including 120
DELETE 120—
erases line 120 and any line after it
DELETE 120
erases line 120 only

Identifying Problems in Your Programs

When a program doesn't work the way you expected, an error message usually occurs. Sometimes the messages are vague, however, and you still don't understand the problem. The Commodore 128 computer has several ways of helping you locate the problem.

Help

The Commodore 128 provides a HELP command that specifies the line in which a problem has occurred. To actuate the HELP command, just press the special HELP key on the row of keys located above the main keyboard.

Type the following statement. It contains an intentional error, so type it just as is:

```
10 ?3;4:5;6
```

When you RUN this one-line program, the computer prints 3 and 4 as expected, but then responds "SYNTAX ERROR IN 10". Let's suppose you can't see the error (a colon instead of a semicolon between 4 and 5). You press the HELP key. (You can also type HELP and press RETURN.) The computer displays the line again, but the 5;6 is highlighted to show the error is in that line.

Error Trapping—The TRAP Command

Usually, if an error occurs in a program, the program "crashes" (stops running). At that point, you can press the HELP key to track down the error. However, you can use the BASIC 7.0 TRAP command to include an error-trapping capability within your program. The TRAP command advises you to locate and correct an error, then resumes program operation. Usually, the error-trapping function is set in the first line of a program:

```
5 TRAP 100
```

tells the computer that if an error occurs to go to a certain line (in this case, line 100). Line 100 appears at the end of the program, and sets up

a contingency. Neither line is executed UNLESS there is an error. When an error occurs, the line with the TRAP statement is enacted, and control is directed to another part of the program. You can use these statements to catch anticipated errors in entering data, resume execution, or return to text mode from a graphics mode, to name just a few options. If you run the DO/LOOP example (which doubled numbers) without an UNTIL statement, you can get an OVERFLOW error and the program crashes. You can prevent that from happening by adding two lines, one at the beginning of the program and one at the end. For this example, you might add these two lines:

```
5 TRAP 100
100 IF N>1 THEN END
```

Even though N has been much greater than one for the entire program, the statement isn't considered until there is an error. When the number "overflows" (is greater than the computer can accept), the TRAP statement goes into effect. Since N is greater than one, the program is directed to END (rather than crashing.)

Here is an example in which trapping is used to prevent a zero from being input for division:

```
10 TRAP 1000
100 INPUT "I CAN DIVIDE BY ANY
NUMBER. GIVE ME A NUMBER TO
DIVIDE";D
110 INPUT "WHAT SHOULD I DIVIDE IT
BY";B
120 A=D/B
130 PRINT D;"DIVIDED BY";B;"EQUALS";A
140 END
1000 IF B=0 THEN PRINT"EVEN I CAN'T
DO THAT"
1100 INPUT "PICK A DIFFERENT
NUMBER";B:RESUME 120
```

Notice the RESUME in line 1100. This tells the computer to return to the line mentioned (in this case, 120) and continue. Depending on the error that was trapped, resuming execution may or may not be possible.

For additional information on error trapping, see the error functions ERR$, EL and ER, described in Chapter V, BASIC 7.0 Encyclopedia.

Program Tracing—The TRON and TROFF Commands

When a problem in a program occurs, or you do not get the results you expect, it can be useful to methodically work through the program and do exactly what the computer would do. This process is called tracing. Draw variable boxes and update the values according to the program statements. Perform calculations and print results following each instruction.

Tracing may show you, for example, that you have used a GOTO with an incorrect line number, or calculated a result but never stored it in a variable. Many program errors can be located by pretending to be the computer, and following only one instruction at a time. Your C128 can perform a type of trace using the special commands TRON and TROFF (short for TRace ON and TRace OFF). When the program is run, with TRACE ON the computer prints the line numbers in the order they are executed, as well as any results. In this way, you may be able to see why your program is not giving the results you expected.

Type any short program we have used so far, or use one of your own design. To activate trace mode, type TRON in DIRECT mode. When you run the program, notice how line numbers appear in brackets before any results are displayed. Try to follow the line numbers and see how many steps the computer needed to arrive at a certain point. TRON will be more interesting if you pick a program with many branches, such as GOTO, GOSUB and IF-THEN-line number. Type TROFF to turn trace mode off before continuing.

You don't have to trace an entire program. You can place TRON within a program as a line prior to the program section causing problems. Put the word TROFF as a program line after the troublesome section. When you run the program, only the lines between TRON and TROFF will be bracketed in the results.

Windowing

Windows are a specific area of the screen that you define as your workspace. Everything you type (lines you type, listings of programs, etc.) after setting a window appears within the window's boundaries, not affecting the screen outside the window area. The Commodore 128 provides two methods of creating windows: the WINDOW command and ESCAPE key functions.

Using the WINDOW Command to Create a Window

The Commodore 128 BASIC 7.0 language features a command that allows you to create and manipulate windows: the WINDOW command. The command format is:

WINDOW top-left column, top-left row, bottom-right column, bottom-right row [,clear option]

The first two numbers after WINDOW specify the column and row number of where you want the **top left corner** of the window to be; the next two numbers are the coordinates for the **bottom right corner.** Remember that the screen format (40 or 80 columns) dictates the acceptable range of these coordinates. You can also include a clear option with this command. If you add 1 to the end of the command, the window screen area is cleared, as in this example:

WINDOW 10, 10, 20, 20, 1

Here's a sample program that creates four windows on the screen, in either 40- or 80-column format.

```
10 PRINT"♥"  :REM CLEAR THE SCREEN
20 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30 B$=A$+A$+A$
40 FOR I=1TO 25 :PRINT B$:NEXT :REM FILL SCREEN WITH CHARACTERS
50 WINDOW 1 ,1 ,8 ,20 :REM DEFINE WINDOW 1
60 PRINT"
70 REM THE PREVIOUS LINE FILLS WINDOW 1 WITH RED
80 WINDOW 15,15,39,20,1 :REM DEFINE 2ND WINDOW
90 PRINT "  "; B$;A$     :REM FILL WINDOW WITH CHARACTERS
100 WINDOW 30,1,39,22,1 :REM DEFINE 3RD WINDOW
110 PRINT"  ":  LIST     :REM SELECT YELLOW AND LIST IN WINDOW
120 WINDOW 5,5,33,18,1  :REM DEFINE 4TH WINDOW ON TOP OF THE OTHER THREE
130 PRINT"♠":PRINTA$:LIST: REM CHANGE COLOR - PRINT A$ AND LIST IN WINDOW
```

Using the ESC Key to Create a Window

To set a window with the ESC (Escape) Key, follow these steps:

1. Move the cursor to the screen position you want as the top left corner of the window.
2. Press the ESC key and release it, and then press T.
3. Move the cursor to the position you want to be the bottom right corner of the window.
4. Press ESC and release, then B. Your window is now set.

You can manipulate the window and the text inside using the ESC key. Screen editing functions, such as inserting and deleting text, scrolling, and changing the size of the window, can be performed by pressing ESC followed by another key. To use a specific function, press ESC and release it. Then press any of the following keys listed for the desired function:

@ Erase everything from cursor to end of screen window
A Automatic insert mode
B Set the bottom right corner of the screen window (at the current cursor location)
C Cancel insert and quote modes
D Delete current line
E Set cursor to non-flashing mode
F Set cursor to flashing mode
G Enable bell (by Control-G)
H Disable bell
I Insert a line
J Move to the beginning of the current line
K Move to the end of the current line
L Turn on scrolling
M Turn off scrolling
N Return to normal (non-reverse video) screen display (80-column only)
O Cancel automatic insert mode
P Erase everything from the beginning of line to the cursor
Q Erase everything from the cursor to the end of the line
R Reverse video screen display (80-column only)
S Change to block cursor (■)
T Set the top left corner of the screen window (at the current cursor location)
U Change to underline cursor (_)
V Scroll screen up one line
W Scroll screen down one line
X Toggle between 40 and 80 columns
Y Restore default TAB stops
Z Clear all TAB stops

Experiment with the ESCape key functions. You will probably find certain functions more useful than others. Note that you can use the usual INST/DEL key to perform text editing inside a window as well.

When a window is set up, all screen output is confined to the "box" you have defined. If you want to clear the window area, press SHIFT and CLEAR/HOME together. To cancel the window, press the CLEAR/HOME key twice. The window is then erased, and the cursor is positioned in the top left corner of the screen. Windows are particularly useful in writing, listing and running programs because they allow you to work in one area of the screen while the rest of the screen stays as is.

2 MHz Operation

The FAST and SLOW Commands

The 2 MHz operating mode allows you to run non-graphic programs in 80-column format at twice the normal speed. You can switch normal and fast operation by using the FAST and SLOW commands.

The FAST command places the Commodore in 2 MHz mode. The format of this command is:

FAST

The SLOW command returns the Commodore 128 to 1 MHz mode. The format of this command is:

SLOW

Keys Unique To C128 Mode

Function Keys

The four keys on the Commodore 128 keyboard on the right side above the numeric keypad are special function keys that let you save time by performing repetitive tasks with the stroke of just one key. The first key reads **F1/F2,** the second **F3/F4,** the third **F5/F6,** and the last **F7/F8.** You can use functions 1 through 4 by pressing the key by itself. To use function keys 5, 6, 7 and 8, press **SHIFT** along with the function key.

Here are the standard functions for each key:

F1	F2	F3	F4
GRAPHIC	**DLOAD"**	**DIRECTORY**	**SCNCLR**
F5	F6	F7	F8
DSAVE"	**RUN**	**LIST**	**MONITOR**

Here's what each function involves:

KEY 1 enters one of the GRAPHICS modes when you supply the number of the graphics area and press RETURN. The GRAPHICS command is necessary for giving graphics commands such as CIRCLE or PAINT. For more on GRAPHICS, see Section 6.

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

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

KEY 4 clears the screen using the SCNCLR command.

KEY 5 prints DSAVE " on the screen. All you do is enter the program name, and press RETURN to save the current program on disk.
KEY 6 RUNs the current program.
KEY 7 displays a LISTing of the current program.
KEY 8 lets you enter the Machine Language Monitor. See Appendix J for a description of the Monitor.

Redefining Function Keys

You can redefine or program any of these keys to perform a function that suits your needs. Redefining is easy, using the KEY command. You can redefine the keys from BASIC programs, or change them at any time in direct mode. A situation where you might want to redefine a function key is when you use a command frequently, and want to save time instead of repeatedly typing in the command. The new definitions are erased when you turn off your computer. You can redefine as many keys as you want and as many times as you want.

If you want to reprogram the F7 function key to return you to text mode from high-resolution or multicolor-graphic modes, for example, you would use the key command in this fashion:

```
KEY 7,"GRAPHIC 0" + CHR$(13)
```

CHR$(13) is the ASCII code character for RETURN. So when you press the F7 key after redefining the key, what happens is the command "GRAPHIC 0" is automatically typed out and entered into the computer with RETURN. Entire commands or series of commands may be assigned to a key.

Other Keys Used in C128 Mode Only

Help

As noted previously, when you make an error in a program, your computer displays an error message to tell you what you did wrong. These error messages are further explained in Appendix A of this manual. You can get more assistance with errors by using the HELP key. After an error message, press the HELP key to locate the exact point where the error occurred. When

you press HELP, the line with the error is highlighted on the screen in reverse video (in 40 column), or underlined (in 80 column output). For example:

?SYNTAX ERROR IN LINE 10 Your computer displays this.

HELP You press HELP.

10 PRONT "COMMODORE COMPUTERS"
The line with the mistake is highlighted in reverse if in 40-column output, or underlined in 80-column output.

No Scroll

Press this key down to stop the text from scrolling when the cursor reaches the bottom of the screen. This turns off scrolling until you press the NO SCROLL key again.

Caps Lock

This key lets you type in all capital letters without using the SHIFT key. The CAPS LOCK key locks when you press it, and must be pressed again to be released. CAPS LOCK only affects the lettered keys.

40/80 Display

The **40/80** key selects the main (default) screen format: either 40 or 80 column. The selected screen displays all messages and output at power-up, or when RESET, or RUN/STOP/RESTORE are used. This key may be used to set the display format **only before** turning on or resetting the computer. You cannot change modes with this key **after** the computer is turned on. Section 8 provides an explanation of 40/80 column modes.

Alt

The ALT key allows programs to assign a special meaning to a given key or set of keys.

Unless a specific application program redefines it, holding down the ALT key and any other key has no effect.

Tab This key works like the TAB key on a typewriter. It may be used to set or clear tab stops on the screen and to move the cursor to the columns where tabs are set.

Line Feed Pressing this key advances the cursor to the next line, similar to a cursor down key.

****

This section covers only some of the concepts, keys and commands that make the Commodore 128 a special machine. You can find further explanations of the BASIC language in the BASIC 7.0 Encyclopedia in Chapter V.

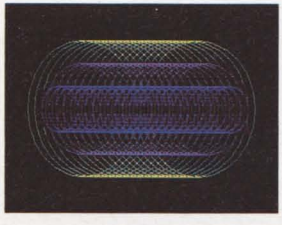
INTRODUCING THE COMMODORE 128

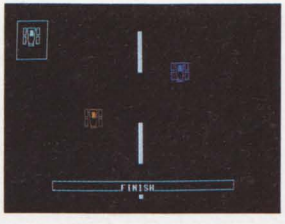


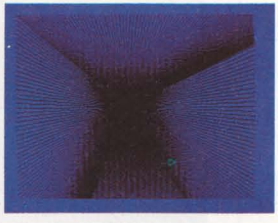
AN EXTRA PAIR OF HANDS FOR THE BUSY EXECUTIVE

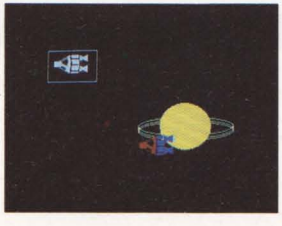










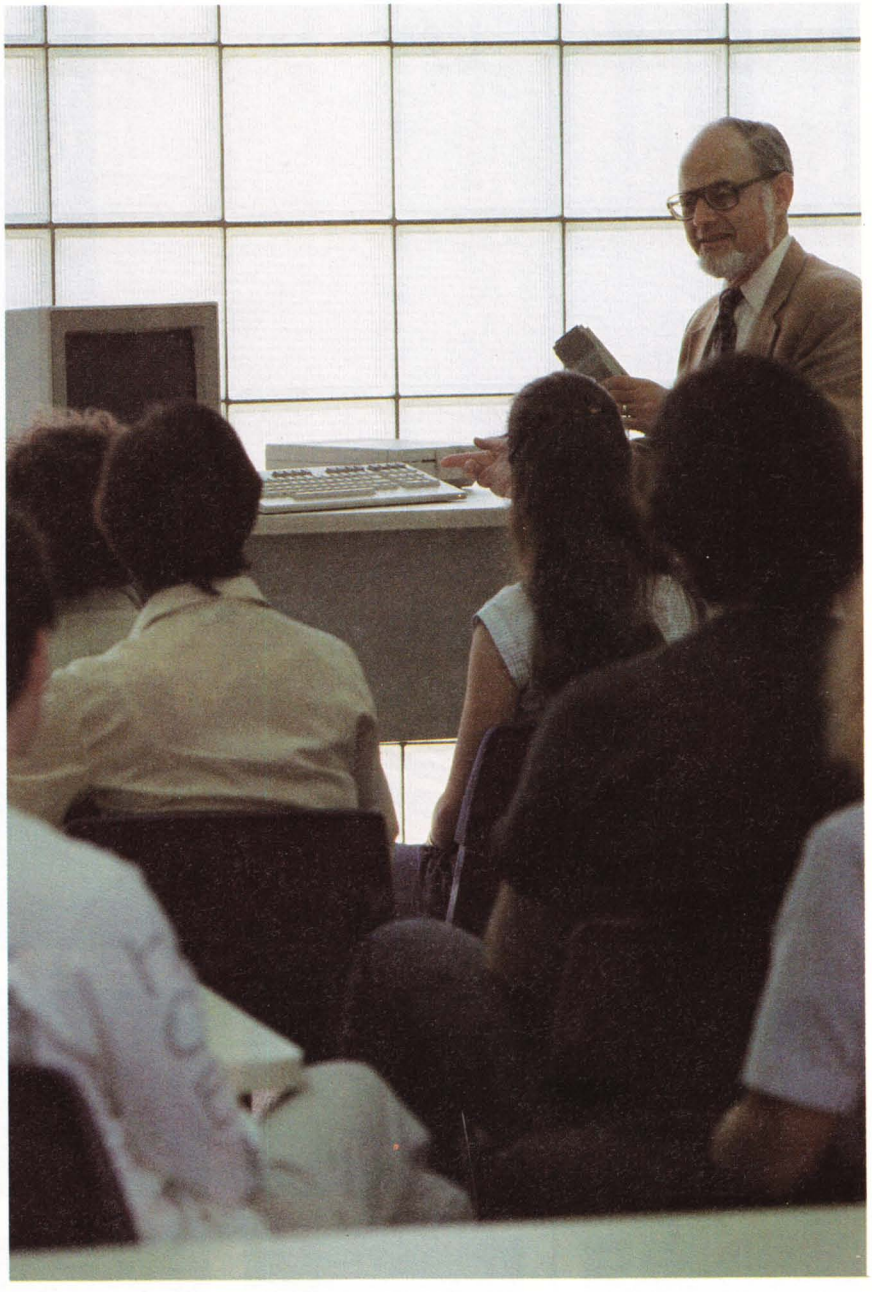




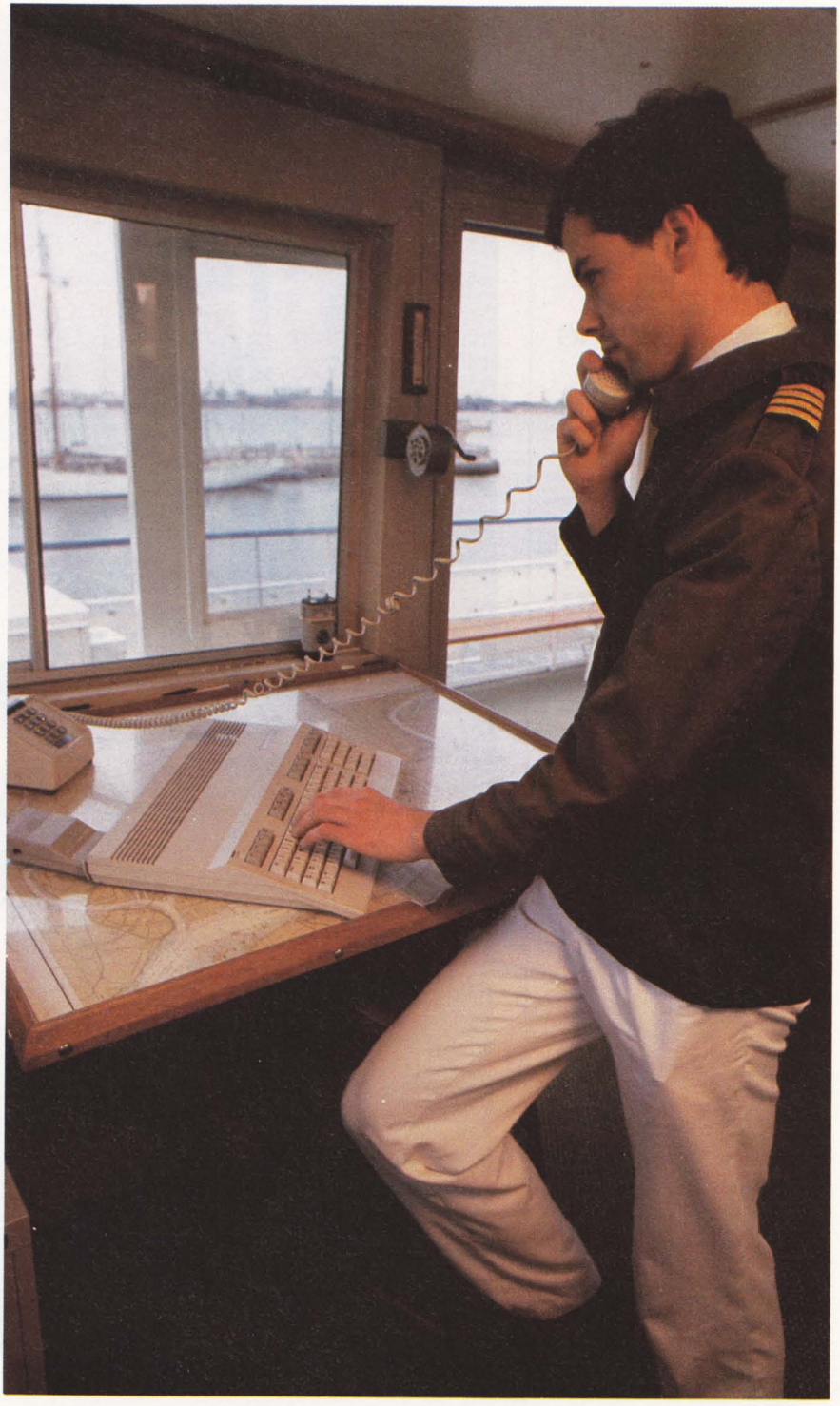


GRAPHICS ARE EASY ON YOUR COMMODORE 128

A POWERFUL LEARNING TOOL AT HOME OR IN THE CLASSROOM

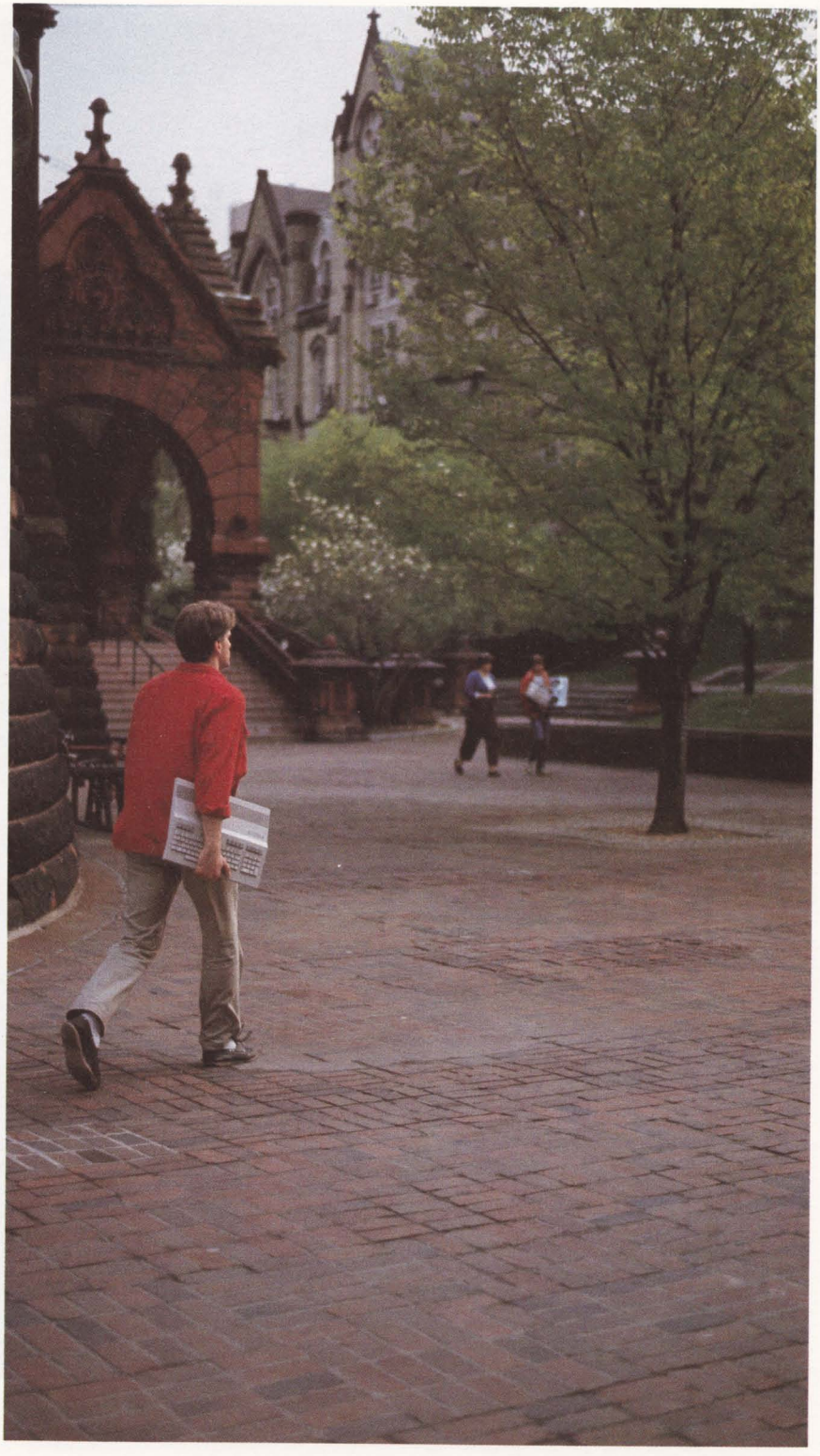


SHIP TO SHORE TELECOMMUNICATING MADE EASY WITH YOUR COMMODORE COMPUTER AND MODEM



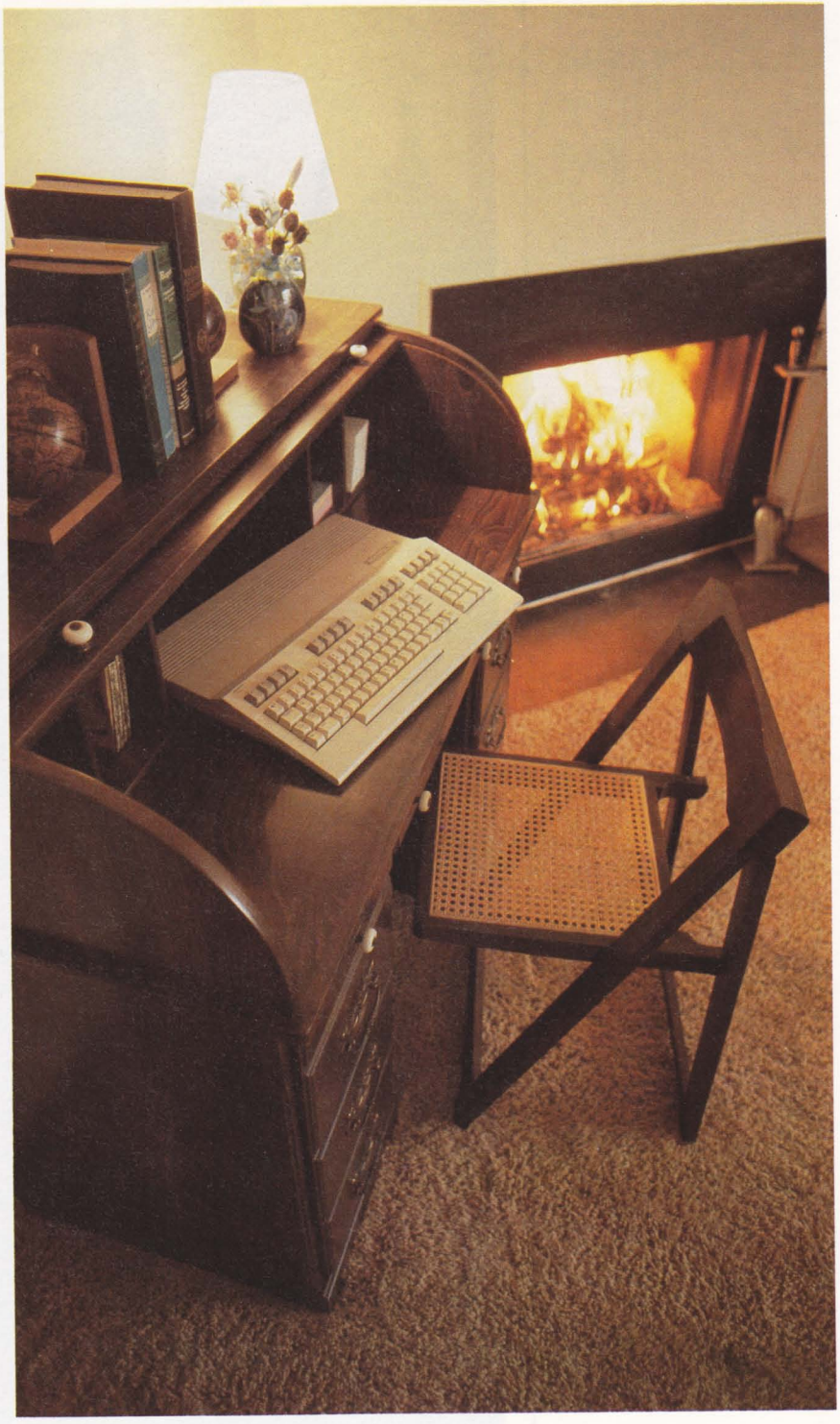
PRODUCTION
PROBLEM SOLVING
ON YOUR
COMMODORE 128





THE COMMODORE 128 AND STUDENT HEADING FOR CLASS

THE BUDGET FINALLY BALANCED– THANKS TO COMMODORE 128



SECTION 6
Color, Animation and Sprite Graphics Statements Unique to the C128

GRAPHICS OVERVIEW 95
Graphics Features 95
Command Summary 96

GRAPHICS PROGRAMMING ON THE COMMODORE 128 97
Choosing Colors 97
Types of Screen Display 98
Selecting the Graphic Mode 99
Displaying Graphics on the Screen 101
Drawing a Circle—The CIRCLE Command 101
Drawing a Box—The BOX Command 102
Drawing Lines, Points and Other Shapes—The DRAW Command 102
PAINTing Outlined Areas—The PAINT Command 103
Displaying Characters on a Bit-Mapped Screen—The CHAR Command 104
Changing the Size of Graphic Images—The SCALE Command 104
Creating a Graphics Sample Program 106

SPRITES: PROGRAMMABLE, MOVABLE OBJECT BLOCKS 109
Sprite Creation 109
Using Sprite Statements in a Program 109
Drawing the Sprite Image 110
Storing the Sprite Data with SSHAPE 111
Saving the Picture Data in a Sprite 112
Turning on Sprites 112
Moving Sprites with MOVSPR 113
Creating a Sprite Program 115
Sprite Definition Mode—The SPRDEF Command 116
Sprite Creation Procedure in SPRite DEFinition Mode 117
Adjoining Sprites 119
Storing Sprite Data in Binary Files 124
Using Binary Files
BSAVE 127
BLOAD 128

Graphics Overview

In C128 mode, the Commodore 128 BASIC 7.0 language provides many new and powerful commands and statements that make graphics programming much easier. Each of the two screen formats available in C128 mode (40 columns and 80 columns) is controlled by a separate microprocessor chip. The 40-column chip is called the Video Interface Controller, or **VIC** for short. The 80-column chip is referred to as the **8563.** The **vic** chip, which provides 16 colors and controls all the highly detailed graphics called bit-mapped graphics. The 80-column chip, which also offers 16 colors, only displays characters and character graphics. Thus, all detailed graphic programs in C128 mode must be done in 40-column format.

Graphics Features

As part of its impressive C128 mode graphics capabilities, the Commodore 128 provides:

- 13 specialized graphics commands
- 16 colors
- Six different display modes
- Eight programmable movable objects called SPRITES
- Combined graphics/text displays

All these features are integrated to provide a versatile, easy-to-use graphics system.

Command Summary

Here is a brief explanation of each graphics command:

BOX	— Draws rectangles on the bit-map screen
CHAR	— Displays characters on the bit-map screen
CIRCLE	— Draws circles, ellipses and other geometric shapes
COLOR	— Selects colors for screen border, foreground, background and characters
DRAW	— Displays lines and points on the bit-map screen
GRAPHIC	— Selects a screen display (text, bit map or split-screen bit map)
PAINT	— Fills area on the bit-map screen with color
SCALE	— Sets the relative size of the images on the bit-map screen
SPRDEF	— Enters sprite definition mode to edit sprites
SPRITE	— Enables, colors, sets sprite screen priorities, and expands a sprite
SPRSAV	— Stores a text string variable into a sprite storage area and vice versa
SSHAPE	— Stores the image of a portion of the bit-map screen into a text-string variable

Most of these commands are described in the examples in this section. See Chapter V, BASIC 7.0 Encyclopedia, for detailed format and information on all graphics commands and functions, including those not discussed in this section.

Graphics Programming on the C128

The following section describes a step-by-step graphics programming example. As you learn each graphics command, add it to a program you will build as you read this section. When you are finished, you will have a complete graphics program.

Choosing Colors

The first step in graphics programming is to choose colors for the screen background, foreground and border. To select colors, type:

COLOR source, color

where **source** is the section of the screen you are coloring (background, foreground, border, etc.), and **color** is the color code for the source. See Figure 6-1 for source numbers, Figure 6-2 for 40-column-format color numbers, and Figure 6-3 for 80-column-format color numbers.

Number	Source
0	40-column background color (VIC)
1	Foreground for the graphics screen (VIC)
2	Foreground color 1 for the multicolor screen (VIC)
3	Foreground color 2 for the multicolor screen (VIC)
4	40-column (VIC) border (whether in text or graphics mode)
5	Character color for 40- or 80-column text screen
6	80-column background color (8563)

Figure 6-1. Source Numbers

Color Code	Color	Color Code	Color
1	Black	9	Orange
2	White	10	Brown
3	Red	11	Light Red
4	Cyan	12	Dark Gray
5	Purple	13	Medium Gray
6	Green	14	Light Green
7	Blue	15	Light Blue
8	Yellow	16	Light Gray

Figure 6-2. Color Numbers in 40-Column Format

Color Code	Color	Color Code	Color
1	Black	9	Dark Purple
2	White	10	Dark Yellow
3	Dark Red	11	Light Red
4	Light Cyan	12	Dark Cyan
5	Light Purple	13	Medium Gray
6	Dark Green	14	Light Green
7	Dark Blue	15	Light Blue
8	Light Yellow	16	Light Gray

Figure 6-3. Color Numbers in 80-Column Format

Types of Screen Display

Your C128 has several different ways of displaying information on the screen; the parameter "source" in the COLOR command pertains to different modes of screen display. The types of video display fall into three categories.

The first one is text display, which displays only characters, such as letters, numbers, special symbols and the graphics characters on the front faces of most C128 keys. The C128 can display text in both 40-column and 80-column screen formats.

The second category of display mode is used for highly detailed graphics, such as pictures and intricate drawings. This type of display mode includes standard bit-map mode and multicolor bit-map mode. Bit-map modes allow you to control each and every individual screen dot or **pixel** (picture element). This allows considerable detail in drawing pictures and other computer art. These graphic displays are only available in 40-column format. The 80-column display is dedicated to text display.

The difference between text and bit map modes lies in the way in which each screen addresses and stores information. The text screen can only manipulate entire characters, each of which covers an area of 8 by 8 pixels on your screen. The more powerful bit-map mode exercises control over each and every pixel on your screen.

The third type of screen display, split screen, is a mixture of the first two types. The split-screen display outputs part of the screen as text and part in bit-map mode (either standard or multicolor). The C128 is

capable of this because it uses two separate parts of the computer's memory to store the two screens: one part for the text, and the other for the graphics screen.

Type the following short program:

```
10 COLOR 0,1: REM TEXT BACKGROUND COLOR = BLACK
20 COLOR 1,3: REM FOREGROUND COLOR FOR BIT MAP
   SCREEN = RED
30 COLOR 4,1: REM BORDER COLOR = BLACK
```

This example colors the background black, the foreground red and the border black.

Selecting the Graphic Mode

The next graphics programming step is to select the appropriate graphic mode. This is done using the GRAPHIC command, whose format is as follows:

GRAPHIC mode [,c] ,s] or GRAPHIC CLR

where **mode** is a digit between 0 and 5, c is either a 0 or 1 and s is a value between 0 and 25. Figure 6-4 shows the values corresponding to the graphic modes.

Mode	Description
0	40-column standard text
1	Standard bit map
2	Standard bit map (split screen)
3	Multicolor bit map
4	Multicolor bit map (split screen)
5	80-column text

Figure 6-4. Graphic Modes

The parameter CLR stands for CLEAR. Figure 6-5 explains the values associated with CLEAR.

C Value	Description
0	Do not clear the graphics screen
1	Clear the graphics screen

Figure 6-5. CLEAR Parameters

When you first run your program, you will want to clear the graphics screen for the first time, so set c equal to 1 in the GRAPHIC command. If you run it a second time, you may want to leave your picture on the screen, instead of drawing it all over again. In this case, set c equal to 0.

The s parameter specified where the start of the text screen in split-screen mode is to begin at the line after the specified line number. If you omit the s parameter and select a split-screen graphic mode (2 or 4), the text screen portion is displayed in rows 20 through 25; the rest of the screen is bit mapped. The s parameter allows you to change the starting line of the text screen to any line on the screen, ranging from 1 through 25. A zero as the s parameter indicates the screen is not split, and is all text.

The final GRAPHIC command parameter is CLR. When you first issue a bit-map graphic command, the Commodore 128 allocates a 9K area for your bit-mapped screen information. 8K is reserved for the data for your bit map and the additional 1K is dedicated for the color data (video matrix). Since 9K is a substantial block of memory, you may want to use it again for another purpose later on in your program. This is the purpose of CLR. It reorganizes the Commodore 128 memory and gives you back the 9K of memory that was dedicated to the bit-map screen, so you can use it for other purposes.

The format for CLR is as follows:

GRAPHIC CLR

When using this format, omit all other GRAPHIC command parameters.

Add the following command to your program. It places the C128 in standard bit-map mode and allocates an 8K bit-map screen (and 1K of color data) for you to create graphics.

40 GRAPHIC 1,1

The second 1 in this command clears the bit-map screen. If you do not want to clear the screen, change the second 1 to 0 (or omit it completely).

NOTE: If you are in bit-map mode and are unable to return to the text screen, press the RUN/STOP and RESTORE keys at the same time, or press the ESC key followed by X, to return to the

80-column screen. Even though you can only **display** graphics with the VIC (40-column) chip, you can still **write** graphics programs in 80-column format. If you have the Commodore 1902 dual monitor and you want to view your graphics program while it is running, you must select the 40-column output by switching the slide switch on the monitor to 40-column output.

Displaying Graphics on the Screen

So far, you have selected a graphics mode and the colors you want. Now you can start displaying graphics on the screen. Start with a circle.

Drawing a Circle—The CIRCLE Command

To draw a circle, use the CIRCLE statement as follows:

```
60 CIRCLE 1,150,130,40,40
```

This displays a circle in the center of the screen. The CIRCLE statement has nine parameters you can select to achieve various types of circles and geometric shapes. For example, by changing the numbers in the CIRCLE statement in line 60 you can obtain different size circles or variations in the shape (e.g., an oval). The CIRCLE statement adds power and versatility in programming Commodore 128 graphics in BASIC. The meaning of the numbers in the CIRCLE statement is explained under the CIRCLE listing in Chapter V, BASIC 7.0 Encyclopedia.

On your Commodore 128 screen, the point where X = 0 and Y = 0 is at the top left corner of the screen, and is referred to as the HOME position. In standard geometry, however, the point where X and Y both equal 0 is the bottom left corner of a graph. Figure 6-6 shows the arrangement of the X (horizontal) and Y (vertical) screen coordinates and the four points at the corners of the C128 screen.

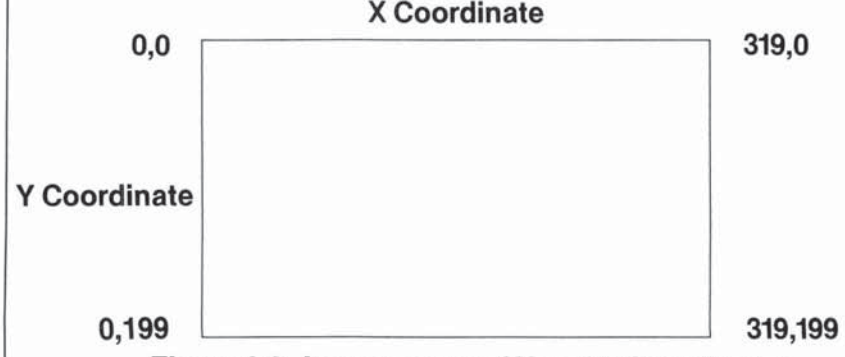


Figure 6-6. Arrangement of X and Y Coordinates

Here's what the numbers mean:

- 1 is the color source (in this case the foreground)
- 20 is the starting X (horizontal) coordinate
- 180 is the starting Y (vertical) coordinate
- 280 is the ending horizontal coordinate
- 180 is the ending vertical screen coordinate.

Drawing a Box—The BOX Command

Now try a box. To draw a box, type:

```
80 BOX1,20,100,80,160,90,1
```

This draws a solid box to the left of the circle. To find out what the numbers in the box statement mean, consult Chapter V, BASIC 7.0 Encyclopedia. The BOX statement has seven parameters you can select and modify to produce different types of boxes. Change the foreground color and draw the outline of a box to the right of the CIRCLE with these statements:

```
90 COLOR1,9:REM CHANGE FOREGROUND COLOR
100 BOX1,220,100,280,160,90,0
```

Experiment with the BOX statement to produce different variations of rectangles and boxes.

Drawing Lines, Points and Other Shapes—The DRAW Command

You now know how to select graphic modes and colors and how to display circles and boxes on the screen. Another graphics statement, DRAW, lets you draw lines on the screen just as you would with a pencil and a piece of paper. The following statement draws a line below the boxes and circle.

```
120 DRAW 1,20,180 TO 280,180
```

To erase a drawn line, change the source (1) in the DRAW statement to 0. The line is drawn with the background color which erases the line. Try using different coordinates and other sources to become accustomed to the DRAW statement.

The DRAW statement can take another form that allows you to DRAW a line, change direction and then DRAW another line, so the lines are continuous. For example, try this statement:

```
130 DRAW 1,10,20 TO 300,20 TO 150,80 TO 10,20
```

This statement DRAWs a triangle on the top of the screen. The four pairs of numbers represent the X and Y coordinates for the three points of the triangles. Notice the first and last coordinates are the

same, since you must finish drawing the triangle on the same point you started. This form of draw statement gives you the power to DRAW almost any geometric shape, such as trapezoids, parallelograms and polygons.

The DRAW statement also has a third form.

You can DRAW one point at a time by specifying the starting X and Y values as follows:

```
150 DRAW 1,155,175
```

This statement DRAWS a dot below the CIRCLE.

As you can see, the DRAW statement has versatile features which give you the capability to create shapes,lines points and a virtually unlimited number of computer drawings on your screen.

PAINTing Outlined Areas—The PAINT Command

The DRAW statement allows you to outline areas on the screen. What if you want to fill areas within your drawn lines? That's where the PAINT statement comes in. The PAINT statement does exactly what the name implies—it fills in, or PAINTs, outlined areas with color. Just as a painter covers a canvas with paint, the PAINT statement covers the areas of the screen with any of the 16 colors. For example, type:

```
160 PAINT 1,150,97
```

Line 160 PAINTS the circle you have drawn in line 60. The PAINT statement fills a defined area until a specified boundary is detected according to which source is indicated. When the Commodore 128 finishes PAINTing, it leaves the pixel cursor at the point where PAINTing began (in this case, at point 150,97).

Here are two more PAINT statements:

```
180 PAINT 1,50,25
200 PAINT 1,225,125
```

Line 180 PAINTS the triangle and line 200 PAINTS the empty box.

*** IMPORTANT PAINTING TIP:** If you choose a starting point in your PAINT statement which is already colored from the same source, the Commodore 128 will not PAINT that area. You must choose a starting point which is entirely inside the boundary of the shape you want to PAINT. The starting point cannot be on

the boundary line of a pixel that is colored from the same source. The source numbers of the screen coordinate and the coordinate specified in the PAINT command must be different.

Displaying Characters on a Bit Mapped Screen—The CHAR Command

So far, the example program has operated in standard bit map mode. Bit map mode uses a completely different area of memory to store the screen data than text mode (the mode in which you enter programs and text). If you enter bit map mode, and try to type characters onto the screen, nothing happens. This is because the characters you are typing are being displayed on the text screen and you are looking at the bit map screen. Sometimes it is necessary to display characters on the bit map screen, when you are creating and plotting charts and graphs. The CHAR command is designed especially for this purpose. To display standard characters on a bit map screen, use the CHAR statement as follows:

```
220 CHAR 1, 11,24,"GRAPHICS EXAMPLE"
```

This displays the text "GRAPHICS EXAMPLE" starting at line 25, column 12. The CHAR command can also be used in text mode, however it is primarily designed for the bit map screen.

Changing the Size of Graphic Images—The SCALE Command

The Commodore 128 has another graphics statement which offers additional power to your graphics system. The SCALE statement offers the ability to scale up (enlarge) or scale down (reduce) the size of graphic images on your screen. The SCALE statement also accomplishes another task, which can be explained as follows.

In standard bit-map mode, the 40-column screen has 320 horizontal coordinates and 200 vertical coordinates. In multicolor bit map

mode, the 40-column screen has only half the horizontal resolution of standard bit map mode, that is, 160 by 200. This reduction in resolution is compensated for by the additional capability of using one additional color for a total of three colors, within an 8 by 8 character matrix. Standard bit map mode can only display two colors within an 8 by 8 character matrix.

When you use the SCALE statement, both standard bit map and multicolor bit map modes have coordinates which are proportional to one another. The scale ranges from 0 through a maximum of 1023 horizontal coordinates. This is true regardless of whether you are in standard bit-map or multicolor mode.

To SCALE your screen, type:

SCALE 1, x, y

and the screen coordinates range from 0 to 65535 whether you are in standard or multicolor high-res mode. To turn off SCALEing, type:

SCALE 0

and the coordinates return to their normal values.

Creating a Sample Graphics Program

So far, you have learned several graphics statements. Now tie the program together and see how the statements work at the same time. Here's how the program looks now. The color statements in lines 70, 110, 140, 170, 190 and 210 are added to display each object in a different color.

```
10 COLOR 0,1 :REM SELECT BKGRND COLOR
20 COLOR 1,3 :REM SELECT FORGRND COLOR
30 COLOR 4,1 :REM SELECT BORDER COLOR
40 GRAPHIC1,1:REM SELECT STND HI RES
60 CIRCLE ,150,130,40,40:REM CIRCLE
70 COLOR 1,6 :REM CHANGE FORGRND COLOR
80 BOX,20,100,80,160,90,1:REM BOX
90 COLOR 1,9 :REM CHANGE FORGRND COLOR
100 BOX,220,100,280,160,90,0:REM BOX
110 COLOR 1,8 :REM CHANGE FORGRND COLOR
120 DRAW 1,20,180 TO 280,180:REM DRAW LINE
130 DRAW 1,10,20 TO 300,20 TO150,80 TO 10,20:REM DRAW TRIANGLE
140 COLOR 1,15 :REM CHANGE FORGRND COLOR
150 DRAW 1,150,175:REM DRAW 1 POINT
160 PAINT 1,150,97:REM PAINT CIRCLE
170 COLOR 1,5 :REM CHANGE FORGRND COLOR
180 PAINT 1,50,25:REM PAINT TRIANGLE
190 COLOR 1,7 :REM CHANGE FORGRND COLOR
200 PAINT 1,225,125:REM PAINT BOX
210 COLOR 1,11 :REM CHANGE FORGRND COLOR
220 CHAR,11,24,"GRAPHICS EXAMPLE":REM DISPLAY TEXT
230 FORI=1TO5000:NEXT: GRAPHIC 0,1:COLOR 1,2
```

Here's what the program does:

- Lines 10 through 30 select a COLOR for the background, foreground and border, respectively.
- Line 40 chooses a graphic mode.
- Line 60 displays a CIRCLE.
- Line 80 DRAWs a colored-in BOX.
- Line 100 DRAWs the outline of a box.
- Line 120 DRAWs a straight line at the bottom of the screen.
- Line 130 DRAWs a triangle.
- Line 150 DRAWs a single point below the CIRCLE.
- Line 160 PAINTs the circle.
- Line 180 PAINTs the triangle.
- Line 200 PAINTs the empty box.

- Line 220 prints the CHARacters "GRAPHICS EXAMPLE" at the bottom of the screen.
- Line 230 delays the program so you can watch the graphics on the screen, switches back to text mode and colors the characters black.

If you want the graphics to remain on the screen, omit the GRAPHIC statement in line 230.

Here are some additional example programs using the graphics statements you just learned.

```
10 COLOR 0,1
20 COLOR 1,8
30 COLOR 4,1
40 GRAPHIC1,1
50 FORI=80TO240 STEP10
60 CIRCLE1,I,100,75,75
70 NEXT
80 COLOR 1,5
90 FORI=80TO250 STEP10
100 CIRCLE1,I,100,50,50
110 NEXT
120 COLOR 1,7
130 FORI=50TO280 STEP10
140 CIRCLE1,I,100,25,25
150 NEXT
160 FORI=1TO7500:NEXT:GRAPHIC0,1:COLOR1,2
```

```
10 GRAPHIC 1,1
20 COLOR0,1
30 COLOR4,1
40 FORI=1TO50
50 Z=INT(((RND(1))*16)+1)* 1
60 COLOR1,Z
70 X=INT(((RND(1))*30)+1)*10
80 Y=INT(((RND(1))*20)+1)*10
90 U=INT(((RND(1))*30)+1)*10
100 V=INT(((RND(1))*20)+1)*10
110 DRAW,X,Y TO U,V
120 NEXT
130 SCNCLR
140 GOTO40
```

```
10 COLOR4,7:COLOR0,7:COLOR1,1
20 GRAPHIC1,1
30 FORI= 400TO1 STEP -5
40 DRAW 1,150,100 TO I,1
50 NEXT
60 FORI= 1TO400 STEP 5
70 DRAW 1,150,100 TO 1,I
80 NEXT
90 FORI= 40TO 320 STEP  5
100 DRAW 1,150,100 TO I,320
110 NEXT
120 FORI= 320TO30STEP -5
130 DRAW 1,150,100 TO 320,I
140 NEXT
150 FORI=1TO7500:NEXT:GRAPHIC0,1:COLOR1,1
```

Type the examples into your computer. RUN and SAVE them for future reference. One of the best ways to learn programming is to study program examples and see how the statements perform their functions. You'll soon be able to use graphics statements to create impressive graphics with your Commodore 128.

If you need more information on any BASIC statement or command, consult the Chapter V, BASIC 7.0 Encyclopedia.

You now have a set of graphic commands that allow you to create an almost unlimited number of graphics displays. But Commodore 128 graphics abilities do not end here. The Commodore 128 has another set of statements, known as SPRITE graphics, which make the creation and control of graphic images fast, easy and sophisticated. These high-level statements allow you to create sprites—moveable graphic objects—the C128 has its own built-in SPRite DEFinition ability. These statements represent the new technology for creating and controlling sprites. Read the next section and take your first step in learning computer animation.

Sprites: Programmable, Movable Object Blocks

You already have learned about some of the Commodore 128's exceptional graphics capabilities. You've learned how to use the first set of high-level graphics statements to draw circles, boxes, lines and dots. You have also learned how to color the screen, switch graphic modes, paint objects on the screen and scale them. Now it's time to take the next step in graphics programming—sprite animation.

If you have worked with the Commodore 64, you already know something about sprites. For those of you who are not familiar with the subject, a sprite is a movable object that you can form into any shape or image. You can color sprites in 16 colors. Sprites can even be multicolor. The best part is that you can move them on the screen. Sprites open the door to computer animation.

Here is the set of statements you will learn about in this section:

MOVSPR
SPRDEF
SPRITE
SPRSAV
SSHAPE

Sprite Creation

The first step in programming sprites is designing the way the sprite looks. For example, suppose you want to design a rocket ship or a racing car sprite. Before you can color or move the sprite, you must first design the image. In C128 mode, you can create sprites in these three ways:

1. Using the new SPRITE statements within a program
2. Using SPRite DEFinition mode (SPRDEF)
3. Using the same method as the Commodore 64.

Using Sprite Statements in a Program

This method uses built-in statements so you don't have to use any aids outside your program to design your sprite as the other two methods require. This method uses some of the graphics statements you learned in the previous section. Here's the general procedure. The details will be added as you progress.

1. Draw a picture with the graphics statements you learned in the last section, such as DRAW, CIRCLE, BOX and PAINT. Make the dimensions of the picture 24 pixels wide by 21 pixels tall in standard bit map mode or 12 pixels wide by 21 tall in multicolor bit map mode.
2. Use the SSHAPE statement to store the picture data into a string variable.
3. Transfer the picture data from the string variable into a sprite with the SPRSAV statement.
4. Turn on the sprite, color it, select either standard or multicolor mode and expand it, all with the SPRITE statement.
5. Move the sprite with the MOVSPR statement.

Drawing the Sprite Image

Here are the actual statements that perform the sprite operations. When you are finished with this section, you will have written your first sprite program. You'll be able to RUN the program as much as you like, and SAVE if for future reference.

The first step is to draw a picture (24 by 21 pixels) on the screen using DRAW, CIRCLE, BOX or PAINT. This example is performed in standard bit map mode, using a black background. Here's the statements that set the graphic mode and color the screen background black.

```
5 COLOR 0,1 :REM COLOR BACKGROUND BLACK
10 GRAPHIC 1,1 :REM SET STND BIT MAP MODE
```

The following statements DRAW a picture of a racing car in the upper-left corner of the screen. You already learned these statements in the last section.

```
5 COLOR 0,1:COLOR 4,1
10 GRAPHIC 1,1
15 BOX 1,2,2,45,45
20 DRAW 1,17,10 TO 28,10 TO 26,30 TO 19,30 TO 17,10 :REM CAR BODY
22 DRAW 1,11,10 TO 15,10 TO 15,18 TO 11,18 TO 11,10: REM UP LEFT WHEEL
24 DRAW 1,30,10 TO 34,10 TO 34,18 TO 30,18 TO 30,10:REM RGHT WHEEL
26 DRAW 1,11,20 TO 15,20 TO 15,28 TO 11,28 TO 11,20:REM LOW LFT WHEEL
28 DRAW 1,30,20 TO 34,20 TO 34,28 TO 30,28 TO 30,20:REM LO RGHT WHEEL
30 DRAW 1,26,28 TO 19,28
32 BOX 1,20,14,26,18,90,1
35 BOX 1,150,35,195,40,90,1:REM STREET
37 BOX 1,150,135,195,140,90,1:REM STREET
40 BOX 1,150,215,195,220,90,1:REM STRT
42 DRAW 1,50,180 TO 300,180:DRAW 1,50,180 TO 50,190:DRAW 1,300,180 TO 300,190
43 DRAW 1,50 ,190 TO 300,190
44 CHAR 1,18,23,"FINISH"
```

RUN the program. You have just drawn a white racing car, enclosed in a box, in the upper-left corner of the screen. You have also drawn a raceway with a finish line at the bottom of the screen. At this point, the racing car is still only a stationary picture. The car isn't a sprite yet, but you have just completed the first step in sprite programming—creating the image.

Storing the Sprite Data with SSHAPE

The next step is to save the picture into a text string. Here's the SSHAPE statement that does it:

```
45 SSHAPE A$,10,11,34,31:REM SAVE THE PICTURE IN A
   STRING
```

The SSHAPE command stores the screen image (bit pattern) into a string variable for later processing, according to the specified screen coordinates.

The numbers 10, 11, 34, 31 are the coordinates of the picture. You must position the coordinates in the correct place or the SSHAPE statement can't store your picture data correctly into the string variable A$. If you position the SSHAPE statement on an empty screen location, the data string is empty. When you later transfer it into a sprite, you'll realize there is no data present. Make sure you position the SSHAPE statement directly on the correct coordinate. Also, be sure to create the picture with the dimensions 24 pixels wide by 21 pixels tall, the size of a single sprite.

The SSHAPE statement transfers the picture of the racing car into a data string that the computer interprets as picture data. The data string, A$, stores a string of zeroes and ones in the computer's memory that make up the picture on the screen. As in all computer graphics, the computer has a way it can represent visual graphics with bits in its memory. Each dot on the screen, called a pixel, has a bit in the computer's memory that controls it. In standard bit-map mode, if the bit in memory is equal to a 1 (on), then the pixel on the screen is turned on. If the controlling bit in memory is equal to a 0 (off), then the pixel is turned off.

Saving the Picture Data in a Sprite

Your picture is now stored in a string. The next step is to transfer the picture data from the data string (A$) into the sprite data area so you can turn it on and animate it. The statement that does this is SPRSAV. Here are the statements:

```
50 SPRSAV A$,1:REM STORE DATA STRING IN SPRITE 1
55 SPRSAV A$,2:REM STORE DATA STRING IN SPRITE 2
```

Your picture data is transferred into sprite 1 and sprite 2. Both sprites have the same data, so they look exactly the same. You can't see the sprites yet, because you have to turn them on.

Turning on Sprites

The SPRITE statement turns on a specific sprite (numbered 1 through 8), colors it, specifies its screen priority, expands the sprite's size and determines the type of sprite display. The screen priority refers to whether the sprite passes in front of or behind the objects on the screen. Sprites can be expanded to twice their original size in either the horizontal or vertical directions. The type of sprite display determines whether the sprite is a standard bit map sprite, or a multicolor bit mapped sprite. Here are the two statements that turn on sprites 1 and 2.

```
60 SPRITE 1,1,7,0,0,0,0:REM TURN ON SPR 1
65 SPRITE 2,1,3,0,0,0,0:REM TURN ON SPR 2
```

Here's what each of the numbers in the SPRITE statements mean:

SPRITE #,O,C,P,X,Y,M

#— Sprite number (1 through 8)
O— Turn On (O = 1) or Off (O = 0)
C— Color (1 through 16)
P— Priority— If P = 0, sprite is in front of objects on the screen
If P = 1, sprite is in back of objects on the screen
X— If X = 1, expand sprite in horizontal (X) direction
If X = 0, sprite is normal horizontal size
Y— If Y = 1, expand sprite in vertical (Y) direction
If Y = 0, sprite is normal vertical size
M— If M = 1, sprite is multicolor
If M = 0, sprite is standard

As you can see, the SPRITE statement is powerful, giving you control over many sprite qualities.

Moving Sprites with MOVSPR

Now that your sprite is on the screen, all you have to do is move it. The MOVSPR statement controls the motion of a sprite and allows you to animate it on the screen. The MOVSPR statement can be used in two ways. First, the MOVSPR statement can place a sprite at an absolute location on the screen, using vertical and horizontal coordinates. Add the following statements to your program:

```
70 MOVSPR 1,240,70:REM POSITION SPRITE 1—X = 240, Y = 70
80 MOVSPR 2,120,70:REM POSITION SPRITE 2—X = 120, Y = 70
```

Line 70 positions sprite 1 at sprite coordinate 240,70. Line 80 places sprite 2 at sprite coordinate 120,70. You can also use the MOVSPR statement to move sprites relative to their original positions. For example, place sprites 1 and 2 at the coordinates as in lines 70 and 80. You want to move them from their original locations to another location on the screen. Use the following statements to move sprites along a specific route on the screen:

```
85 MOVSPR 1,180 # 6:REM MOVE SPRITE 1 FROM THE TOP
TO THE BOTTOM
87 MOVSPR 2,180 # 7:REM MOVE SPRITE 2 FROM THE TOP
TO THE BOTTOM
```

The first number in this statement is the sprite number. The second number is the direction expressed as the number of degrees to move in the clockwise direction, relative to the original position of the sprite. The pound sign (#) signifies that the sprite is moved at the specified angle and speed relative to a starting position, instead of an absolute location, as in lines 70 and 80. The final number specifies the speed in which the sprite moves along its route on the screen, which ranges from 0 through 15.

The MOVSPR command has two alternative forms. See Chapter V, BASIC 7.0 Encyclopedia for these notations.

Sprites use an entirely different coordinate plane than bit-map coordinates. The bit-map coordinates range from points 0,0 (the top left corner) to 319,199 (bottom right corner). The visible sprite coordinates start at point 50,24 and end at point 250,344. The rest of the sprite coordinates are off the screen and are not visible, but the sprite still moves according to them. The OFF-screen locations allow sprites to move smoothly onto and off of the screen. Figure 6-7 illustrates the sprite coordinate plane and the visible sprite positions.

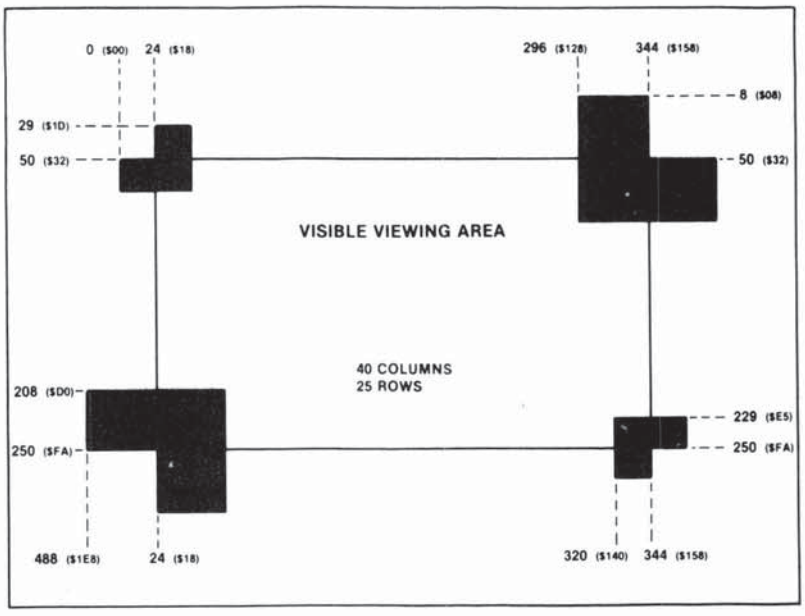


Figure 6-7. Visible Sprite Coordinates

Now RUN the entire program with all the steps included. You have just written your first sprite program. You have created a raceway with two racing cars. Try adding more cars and more objects on the screen. Experiment by drawing other sprites and include them in the raceway. You are now well on the way in sprite programming. Use your imagination and think of other scenes and objects you can animate. Soon you can create all kinds of animated computer "movies."

To stop the sprites, press RUN/STOP and RESTORE at the same time.

Creating a Sprite Program

You now have a working sprite program example. Here's the complete program listing:

```
5 COLOR 0,1:COLOR 4,1
10 GRAPHIC 1,1
15 BOX 1,2,2,45,45
20 DRAW 1,17,10 TO 28,10 TO 26,30 TO 19,30 TO 17,10 :REM CAR BODY
22 DRAW 1,11,10 TO 15,10 TO 15,18 TO 11,18 TO 11,10: REM UP LEFT WHEEL
24 DRAW 1,30,10 TO 34,10 TO 34,18 TO 30,18 TO 30,10:REM RGHT WHEEL
26 DRAW 1,11,20 TO 15,20 TO 15,28 TO 11,28 TO 11,20:REM LOW LFT WHEEL
28 DRAW 1,30,20 TO 34,20 TO 34,28 TO 30,28 TO 30,20:REM LO RGHT WHEEL
30 DRAW 1,26,28 TO 19,28
32 BOX 1,20,14,26,18,90,1
35 BOX 1,150,35,195,40,90,1:REM STREET
37 BOX 1,150,135,195,140,90,1:REM STREET
40 BOX 1,150,215,195,220,90,1:REM STRT
42 DRAW 1,50,180 TO 300,180:DRAW 1,50,180 TO 50,190:DRAW 1,300,180 TO 300,190
43 DRAW 1,50 ,190 TO 300,190
44 CHAR 1,18,23,"FINISH"
45 SSHAPE A$,11,10,34,31:REM SAVE SPR IN A$
50 SPRSAV A$,1:REM SPR0 DATA
55 SPRSAV A$,2:REM SPR1 DATA
60 SPRITE 1,1,7,0,0,0,0:REM SPR1 ATTRIB
65 SPRITE 2,1,3,0,0,0,0:REM SPR2 ATTRIB
70 MOVSPR 1,240,70
80 MOVSPR 2,120,70
85 MOVSPR 1,180 # 6
90 MOVSPR 2,180 # 7
95 FOR I=1TO5000:NEXT
99 GRAPHIC 0,1
```

Here's what the program does:

- Line 5 COLORs the screen black.
- Line 10 sets standard high-resolution GRAPHIC mode.
- Line 15 DRAWs a box in the top-left corner of the screen.
- Lines 20 through 32 DRAW the racing car.
- Lines 35 through 44 DRAW the racing lanes and a finish line.
- Line 45 transfers the picture data from the racing car into a string variable.
- Lines 50 and 55 transfer the contents of the string variable into sprites 1 and 2.
- Lines 60 and 65 turn on sprites 1 and 2.
- Lines 70 and 80 position the sprites at the top of the screen.
- Lines 85 and 87 animate the sprites as though the two cars are racing each other across the finish line.

In this section, you have learned how to create sprites, using the built-in C128 graphics statements such as DRAW and BOX. You learned how to control the sprites, using the Commodore 128 sprite statements. The Commodore 128 has two other ways of creating sprites. The first is with the built-in SPRite DEFinition ability, as described in the following paragraphs. The other method of creating sprites is the same as that used for the Commodore 64; see the C64 Programmer's Reference Guide for details on this sprite-creation technique.

Sprite Definition Mode—The SPRDEF Command

The Commodore 128 has a built-in SPRite DEFinition mode which enables you to create sprites on your Commodore 128. You may be familiar with the Commodore 64 method of creating sprites, in which you required to either have an additional sprite editor, or design a sprite on a piece of graph paper and then READ in the coded sprite DATA and POKE it into an available sprite block. With the new Commodore 128 sprite definition command SPRDEF, you can construct and edit your own sprites in a special sprite work area.

To enter SPRDEF mode, type:

SPRDEF

and press RETURN. The Commodore 128 displays a sprite grid on the screen. In addition, the computer displays the prompt:

SPRITE NUMBER ?

Enter a number between 1 and 8. The computer fills the grid and displays the corresponding sprite in the upper right corner of the screen. From now on, we will refer to the sprite grid as the work area. The work area has the dimensions of 24 characters wide by 21 characters tall. Each character position within the work area corresponds to 1 pixel within the sprite, since a sprite is 24 pixels wide by 21 pixels tall. While within the work area in SPRDEF mode, you have several editing commands available to you. Here's a summary of the commands on the following page:

Sprite Definition Mode Command Summary

CLR key—Erases the entire work area
M key—Turns on/off multicolor sprite
CTRL 1-8—Selects sprite foreground color 1-8
C= 1-8—Selects sprite foreground color 9-16
1 key—Turns on pixels in the background color
2 key—Turns on pixels in the foreground color
3 key—Turns on areas in multicolor1
4 key—Turns on areas in multicolor2
A key—Turns on/off automatic cursor movement
CRSR keys—Moves the cursor (+) within the work area
RETURN—moves cursor to the start of the next line
HOME key—Moves cursor to the top left corner of work area.
X key—Expands sprite horizontally
Y key—Expands sprite vertically
Shift RETURN—Saves sprite from work area and returns to SPRITE NUMBER prompt
C key—copies one sprite to another
STOP key—Turns off displayed sprite and returns to SPRITE NUMBER prompt without changing the sprite
RETURN key—(at SPRITE NUMBER prompt) Exits SPRDEF mode

Sprite Creation Procedure in SPRite DEFinition Mode

Here's the general procedure to create a sprite in SPRite DEFinition mode:

1. Clear the work area by pressing the shift and CLR/HOME keys at the same time.
2. If you want a multicolor sprite, press the M key and an additional cursor appears next to the original one. Two cursors appear since multicolor mode actually turns on two pixels for every one in standard sprite mode. This is why multicolor mode is only half the horizontal resolution of standard high-res mode.
3. Select a color for your sprite. For colors between 1 and 8, hold down the CONTROL key and press a key between 1 and 8. To select color codes between 9 and 16, hold down the Commodore (C=) key and press a key between 1 and 8.
4. Now you are ready to start creating the shape of your sprite. The numbered keys 1 through 4 fill in the sprite and give it shape. For a single color sprite, use the 2 key to fill a character position within the work area. Press the 1 key to erase what you have drawn with the 2 key. If you want to fill one character position at a

time, press the A key. Now you have to move the cursor manually with the cursor keys. If you want the cursor to move automatically to the right while you hold it down, do not press the A key since it is already set to automatic cursor movement. As you fill in a character position within the work area, you can see the corresponding pixel in the displayed sprite turn on. Sprite editing occurs as soon as you edit the work area.

In multicolor mode, the 3 key fills two character positions in the work area with the multicolor 1 color, the 4 key fills two character positions with the multicolor 2.

You can turn off (color the pixel in the background color) filled areas within the work area with the 1 key. In multicolor mode, the 1 key turns off two character positions at a time.

5. While constructing your sprite, you can move freely in the work area without turning on or off any pixels using the RETURN, HOME and cursor keys.
6. At any time, you may expand your sprite in both the vertical and horizontal directions. To expand vertically, press the Y key. To expand horizontally, press the X key. To return to the normal size sprite display, press the X or Y key again.

   When a key turns on AND off of the same control, it is referred to as toggling, so the X and Y keys toggle the vertical and horizontal expansion of the sprite.
7. When you are finshed creating your sprite and are happy with the way it looks, save it by holding down the SHIFT key and pressing the RETURN key. The Commodore 128 SAVEs the sprite data in the appropriate sprite storage area. The displayed sprite in the upper right corner of the screen is turned off and control is returned to the SPRITE NUMBER prompt. If you want to create another sprite enter another sprite number and edit the new sprite just as you did with the first one. If you want to display the original sprite in the work area again, enter the original sprite number. If you want to exit SPRITE DEFinition mode, simply press RETURN at the SPRITE NUMBER prompt.
8. You can copy one sprite into another with the "C" key.
9. If you do not want to SAVE your sprite, press the STOP key. The Commodore 128 turns off the displayed sprite and returns to the SPRITE NUMBER prompt.
10. To EXIT SPRite DEFinition mode, press the RETURN key while the SPRITE NUMBER prompt is displayed on the screen when no

sprite number follows it. You can exit under either of the following conditions:

Immediately after you SAVE your sprite (shift RETURN),
Immediately after you press the STOP key

Once you have created a sprite and have exited SPRite DEFinition mode, your sprite data is stored in the appropriate sprite storage area in the Commodore 128's memory. Since you are now back in the control of the BASIC language, you have to turn on your sprite in order to see it on the screen. To turn it on again, use the SPRITE command you learned previously. For example, you created sprite 1 in SPRDEF mode. To turn it on in BASIC, color it blue and expand it in both the X and Y directions enter this command:

```
SPRITE 1,1,7,0,1,1,0
```

Now use the MOVSPR command to move it at a 90-degree angle at a speed of 5, as follows:

```
MOVSPR 1, 90 # 5
```

Now you know all about SPRDEF mode. First, create the sprite, save the sprite data and exit from SPRDEF mode to BASIC. Next turn on your sprite with the SPRITE command. Move it with the MOVSPR command. When you're finshed programming, SAVE your sprite data in a binary file with the BSAVE command as follows:

```
BSAVE "filename", B0, P3584 TO 4096
```

When you want to use the sprite data again from disk, load the previously BSAVEd binary file with the BLOAD command as follows:

```
BLOAD "filename"[, B0, P3584 TO P4096]
```

The portion in brackets is optional. BLOAD loads data into the address from which it was saved.

Now you know the new methods for creating sprites: 1) SSHAPE, SPRSAV, SPRITE, MOVSPR, 2) SPRDEF MODE. Experiment with all these methods and master sprite animation.

See **Storing Sprite Data in Binary Files** later in this section for more information.

Adjoining Sprites

You have learned how to create, color, turn on and animate a sprite. An occasion may arise when you want to create a picture that is too

detailed or too large to fit into a single sprite. In this case, you can join two or more sprites so the picture is larger and more detailed than with a single sprite. By joining sprites, each one can move independently of one another. This gives you much more control over animation than a single sprite.

This section includes an example using two adjoining sprites. Here's the general procedure (algorithm) for writing a program with two or more adjoining sprites.

1. Draw a picture on the screen with Commodore 128 graphics statements, such as DRAW, BOX and PAINT, just as you did in the raceway program in the last section. This time, make the picture twice as large as a single sprite with the dimensions 48 pixels wide by 21 pixels tall.
2. Use two SSHAPE statements to store the sprites into two separate data strings. Position the first SSHAPE statement coordinates over the 24 by 21 pixel area of the first half of the picture you drew. Then position the second SSHAPE statement coordinates over the second 24 by 21 pixel area. Make sure you store each half of the picture data in a different string. For example, the first SSHAPE statement stores the first half of the picture into A$, and the second SSHAPE statement stores the second half of the picture in B$.
3. Transfer the picture data from each data string into a separate sprite with the SPRSAV statement.
4. Turn on each sprite with the SPRITE statement.
5. Position the sprites so the beginning of one sprite starts at the pixel next to where the first sprite ends. This is the step that actuallly joins the sprites. For example, draw a picture 48 by 42 pixels. Position the first sprite (1, for example) at location 10,10 with this statement:

```
100 MOVSPR 1,10,10
```

where the first number is the sprite number, the second number is the horizontal (X) coordinate and the third number is the vertical (Y) coordinate. Position the second sprite 24 pixels to the right of sprite 1 with this statement:

```
200 MOVSPR 2,34,10
```

At this point, the two sprites are displayed directly next to each other. They look exactly like the picture you drew in the beginning of the program, using the DRAW, BOX and PAINT statements.

6. Now you can move the sprites any way you like, again using the MOVSPR statement. You can move them together along the same path or in different directions. As you learned in the last section, the MOVSPR statement allows you to move sprites to a specific location on the screen, or to a location relative to the sprite's original position.

   The following program is an example of adjoining sprites. The program creates an outer space environment. It draws stars, a planet and a spacecraft similiar to Apollo. The spacecraft is drawn, then stored into two data strings, A$ and B$. The front of the spaceship, the capsule, is stored in sprite 1. The back half of the spaceship, the retro rocket, is stored in sprite 2. The spacecraft flies slowly across the screen twice. Since it is traveling so slowly and is very far from Earth, it needs to be launched earthward with the retro rockets. After the second trip across the screen, the retro rockets fire and propel the capsule safely toward Earth.

   Here's the program listing:

```
5 COLOR 4,1:COLOR 0,1:COLOR 1,2:REM SELECT BLACK BORDER & BKGRND, WHITE FRGRD
10 GRAPHIC 1,1:REM SET HI RES MODE
17 FOR I=1TO40
18 X=INT(RND(1)*320)+1:REM DRAW STARS
19 Y=INT(RND(1)*200)+1:REM DRAW STARS
21 DRAW 1,X,Y:NEXT    :REM DRAW STARS
22 BOX 0,0,5,70,40,,1:REM CLEAR BOX
23 BOX 1,1,5,70,40:REM BOX-IN SPACESHIP
24 COLOR 1,8:CIRCLE 1,190,90,35,25:PAINT 1,190,95:REM DRAW & PAINT PLANET
25 CIRCLE 1,190,90,65,10:CIRCLE 1,190,93,65,10:CIRCLE 1,190,95,65,10:COLOR 0,1
26 DRAW 1,10,17 TO 16,17 TO 32,10 TO 33,20 TO 32,30 TO 16,23 TO 10,23 TO 10,17
28 DRAW 1,19,24 TO 20,21 TO 27,25 TO 26,28:REM BOTTOM WINDOW
35 DRAW 1,20,19 TO 20,17 TO 29,13 TO 30,18 TO 28,23 TO 20,19:REM TOP WINDOW
38 PAINT 1,13,20:REM PAINT SPACESHIP
40 DRAW 1,34,10 TO 36,20 TO 34,30 TO 45,30 TO 46,20 TO 45,10 TO 34,10:REM SP1
42 DRAW 1,45,10 TO 51,12 TO 57,10 TO 57,17 TO 51,15 TO 46,17:REM ENG1
43 DRAW 1,46,22 TO 51,24 TO 57,22 TO 57,29 TO 51,27 TO 45,29:REM ENG2
44 PAINT 1,40,15:PAINT 1,47,12:PAINT 1,47,26:DRAW 0,45,30 TO 46,20 TO 45,10
45 DRAW 0,34,14 TO 44,14 :DRAW 0,34,21 TO 44,21:DRAW 0,34,28 TO 44,28
47 SSHAPE A$,10,10,33,32:REM SAVE SPRITE IN A$
48 SSHAPE B$,34,10,57,32:REM SAVE SPRITE IN B$
50 SPRSAV A$,1:REM SPR1 DATA
55 SPRSAV B$,2:REM SPR2 DATA
60 SPRITE 1,1,3,0,0,0,0:REM SET SPR1 ATTRIBUTES
65 SPRITE 2,1,7,0,0,0,0:REM SET SPR2 ATTRIBUTES
82 MOVSPR 1,150 ,150:REM ORIGINAL POSITION OF SPR1
83 MOVSPR 2,172 ,150:REM ORIGINAL POSITION OF SPR2
85 MOVSPR 1,270 # 5 :REM MOVE SPR1 ACROSS SCREEN
87 MOVSPR 2,270 # 5 :REM MOVE SPR2 ACROSS SCREEN
90 FOR I=1TO 5950:NEXT:REM DELAY
92 MOVSPR 1,150,150:REM POSITION SPR1 FOR RETRO ROCKET LAUNCH
93 MOVSPR 2,174,150:REM POSITION SPR2 FOR RETRO ROCKET LAUNCH
95 MOVSPR 1,270 # 10 :REM SPLIT ROCKET
96 MOVSPR 2, 90 # 5  :REM SPLIT ROCKET
97 FOR I=1TO 1200:NEXT:REM DELAY
98 SPRITE 2,0:REM TURN OFF RETRO ROCKET (SPR2)
99 FOR I=1TO 20500:NEXT:REM DELAY
100 GRAPHIC 0,1:REM RETURN TO TEXT MODE
```

Here's an explanation of the program:

- Line 5 COLORs the background black and the foreground white.
- Line 10 selects standard high-resolution mode and clears the high-res screen.
- Line 23 BOXes in a display area for the picture of the spacecraft in the top-left corner of the screen.
- Lines 17 through 21 DRAW the stars.
- Line 24 DRAWs and PAINTs the planet.
- Line 25 DRAWs the CIRCLEs around the planet.
- Line 26 DRAWs the outline of the capsule portion of the spacecraft.

- Line 28 DRAWs the bottom window of the space capsule.
- Line 35 DRAWs the top window of the space capsule.
- Line 38 PAINTs the space capsule white.
- Line 40 DRAWs the outline of the retro rocket portion of the spacecraft.
- Lines 42 and 43 DRAW the retro rocket engines on the back of the spacecraft.
- Line 44 PAINTs the retro rocket engines and DRAWs an outline of the back of the retro rocket in the background color.
- Line 45 DRAWs lines on the retro rocket portion of the spacecraft in the background color. (At this point, you have displayed only pictures on the screen. You have not used any sprite statements, so your rocketship is not yet a sprite.)
- Line 47 positions the SSHAPE coordinates above the first half (24 by 21 pixels), of the capsule of the spacecraft and stores it in a data string, A$.
- Line 48 positions the SSHAPE coordinates above the second half (24 by 21 pixels) of the spacecraft and stores it in a data string, B$.
- Line 50 transfers the data from A$ into sprite 1.
- Line 55 transfers the data from B$ into sprite 2.
- Line 60 turns on sprite 1 and colors it red.
- Line 65 turns on sprite 2 and colors it blue.
- Line 82 positions sprite 1 at coordinate 150,150.
- Line 83 positions sprite 2, 24 pixels to the right of the starting coordinate of sprite 1.
- Lines 82 and 83 actually join the two sprites.
- Lines 85 and 87 move the joined sprites across the screen.
- Line 90 delays the program. This time, delay is necessary for the sprites to complete the two trips across the screen. If you leave out the delay, the sprites do not have enough time to move across the screen.
- Lines 92 and 93 position the sprites in the center of the screen, and prepare the spacecraft to fire the retro rockets.
- Line 95 propels sprite 1, the space capsule, forward. The number 10 in line 95 specifies the speed in which the sprite moves. The speed ranges from 1, which is stop, to 15, which is lightning fast.
- Line 96 moves the expired retro rocket portion of the spacecraft backwards and off the screen.
- Line 97 is another time delay so the retro rocket, sprite 2, has time to move off the screen.
- Line 98 turns off sprite 2, once it is off the screen.
- Line 99 is another delay so the capsule can continue to move across the screen.
- Line 100 returns you to text mode.

Working with adjoining sprites can be more interesting than working with a single sprite. The main points to remember are: (1) Make sure you position the SSHAPE coordinates at the correct locations on the screen, so you save the picture data properly; and (2) be certain to position the sprite coordinates in the correct location when you are joining them with the MOVESPR statement. In this example, you positioned sprite 2 at a location 24 pixels to the right of sprite 1.

Once you master the technique of adjoining two sprites, try more than two. The more sprites you join, the better the detail and animation will be in your programs.

The C128 has two additional SPRITE commands, SPRCOLOR and COLLISION, which are not covered in this chapter. To learn about these commands, refer to Chapter V, the BASIC 7.0 Encyclopedia.

Storing Sprite Data in Binary Files

> **NOTE:** The following explanation assumes some knowledge of machine language, memory locations, binary files and object code files.

The Commodore 128 has two new commands, BLOAD and BSAVE, which make handling sprite data neat and easy. The "B" in BLOAD and BSAVE stand for BINARY. The BSAVE and BLOAD commands save and load binary files to and from disk. A binary file consists of either a portion of a machine language program, or a collection of data within a specified address range. You may be familiar with the SAVE command within the built-in machine language monitor. When you use this SAVE command, the resulting file on disk is considered a binary file. A binary file is easier to work with than an object code file since you can load a binary file without any further preparation. An object code file must be loaded with a loader, as in the Commodore 64 Assembler Development System; then the SYSTEM command (SYS) must be used to execute it. When loading binary files, remember to load them in either of these two ways:

LOAD "binary filename",8,1

or

BLOAD"binary filename",B0,Pstart

where start is 3584 if you are loading sprite data files.

In the first method you must specify the ",1" at the end or else the computer treats it as a BASIC program file and loads it at the beginning of BASIC text. The ",1" tells the computer to load the binary file into the same place from which it was stored.

You're probably wondering what this has to do with sprites. Here's the connection. The Commodore 128 has a dedicated portion of memory ranging from decimal address 3584 ($0E00) through 4095 ($0FFF), where sprite data is stored. This portion of memory takes up 512 bytes. As you know, a sprite is 24 pixels wide by 21 pixels tall. Each pixel requires one bit of memory. If the bit in a sprite is off (equal to 0), the corresponding pixel on the screen is considered off and it takes on the color of the background. If a pixel within a sprite is on (equal to 1), the corresponding pixel on the screen is turned on in the foreground color. The combination of zeroes and ones produces the image you see on the screen.

Since a sprite is 24 by 21 pixels and each pixel requires one bit of storage in memory, one sprite uses up 63 bytes of memory. See Figure 6-8 to understand the storage requirements for a sprite's data.

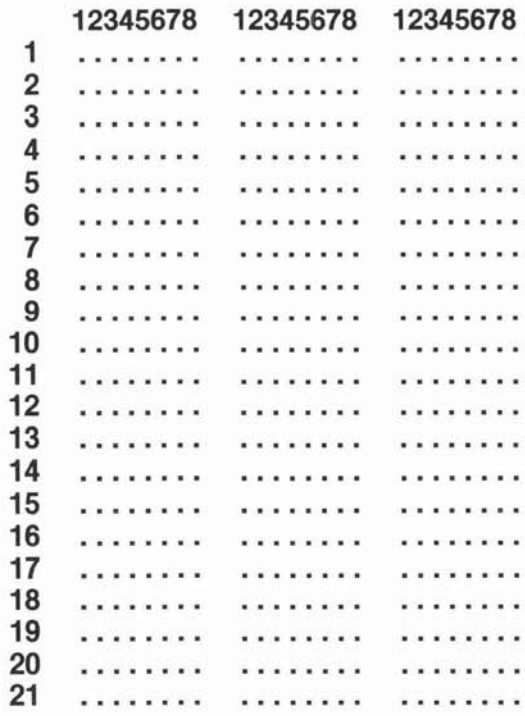


Figure 6-8. Sprite Data Requirements

A sprite requires 63 bytes of data. Each sprite block is actually made up of 64 bytes; the extra byte is not used. Since the Commodore 128 has eight sprites and each one consists of a 64-byte sprite block, the computer needs 512 (8 × 64) bytes to represent the data of all eight sprite images.

The entire area where all eight sprite blocks reside starts at memory location 3584 ($0E00) and ends at location 4095 ($0FFF). Figure 6-9 lists the memory address ranges where each individual sprite stores its data.

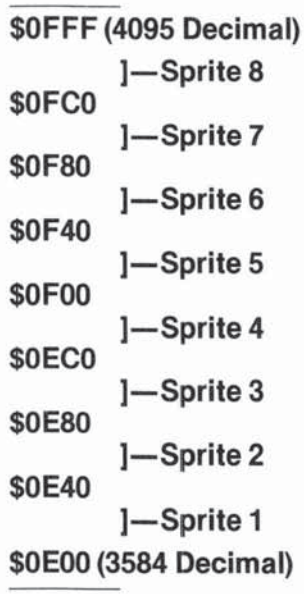


Figure 6-9. Memory Address Ranges for Sprite Storage

BSAVE

Once you exit from the SPRDEF mode, you can save your sprite data in binary sprite files. This way, you can load any collection of sprites back into the Commodore 128 neatly and easily. Use this command to save your sprite data into a binary file:

BSAVE "filename", B0, P3584 TO P4096

The binary filename is a name you give to the file. The "B0" specifies that you are saving the sprite data from bank 0. The parameters "P3584 TO P4096" signify you are saving the address range 3584 ($0E00) through 4095 ($0FFF), which is the range where all the sprite data is stored.

You do not have to define all of the sprites when you BSAVE them. The sprites you do define are BSAVEd from the correct sprite block. The undefined sprites are also BSAVEd in the binary file from the appropriate sprite block, but they do not matter to the computer. It is

easier to BSAVE the entire 512 bytes of all eight sprites, regardless if all the sprites are used, rather than BSAVE each sprite block individually.

BLOAD

Later on, when you want to use the sprites again, just BLOAD the entire 512 bytes for all of the sprites into the range starting at 3584 ($0E00) and ending at 4095 ($0FFF). Here's the command to accomplish this:

BLOAD "filename"[, B0, P3584]

Use the same filename you entered when you BSAVEd your original sprite data. The "B0" stands for the bank number 0 and the P3584 specifies the starting location where the binary sprite data file is loaded. The last two parameters are optional.

****

In this section you have seen how much the new Commodore 7.0 BASIC commands can simplify the usually complex process of creating and animating graphic images. The next section describes some other new BASIC 7.0 commands that do the same for music and sound.

SECTION 7
Sound and Music in C128 Mode

INTRODUCTION 131

THE SOUND STATEMENT 133
- Writing a SOUND Program 134
- Random Sounds 138

ADVANCED SOUND AND MUSIC IN C128 MODE 140
- A Brief Background: The Characteristics of Sound 140
- Making Music on the Commodore 128 142
  - The ENVELOPE Statement 142
  - The TEMPO Statement 145
  - The PLAY Statement 145
  - The SID Filter 149
  - The FILTER Statement 152
- Tying your Music Program Together 153
- Advanced Filtering 154

CODING A SONG FROM SHEET MUSIC 156

Introduction

The Commodore 128 has one of the most sophisticated built-in sound synthesizers available in a microcomputer. The synthesizer, called the Sound Interface Device (SID), is a chip dedicated solely to generating sound and music. The SID chip is capable of producing three independent **voices** (sounds) simultaneously. Each of the voices can be played in one of four types of sounds, called waveforms. The SID chip also has programmable Attack, Decay, Sustain and Release (ADSR) parameters. These parameters define the quality of a sound. In addition, the synthesizer has a filter you can use to choose certain sounds, eliminate others, or modify the characters of a sound or sounds. In this section you will learn how to control these parameters to produce almost any kind of sound.

To make it easy for you to select and manipulate the many capabilities of the SID chips, Commodore has developed new and powerful BASIC music statements.

Here are the new sound and music statements available on the Commodore 128:

SOUND
ENVELOPE
VOL
TEMPO
PLAY
FILTER

This section explains these sound statements, one at a time, in the process of constructing a sample musical program. When you are finished with this section, you will know the ingredients that go into a musical program. You'll be able to expand on the example and write

programs that play intricate musical compositions. Eventually, you'll be able to program your own musical scores, make your own sound effects and play works of the great classical masters such as Beethoven and contemporary artists like the Beatles. You can even add computer-generated music to your graphics programs to create your own "videos."

The SOUND Statement

The SOUND statement is designed primarily for quick and easy sound effects in your programs. You will learn a more intricate way of playing complete musical arrangements with the other sound statements later in this section.

The format for the SOUND statement is as follows:

SOUND VC, FREQ, DUR[, DIR[, MIN[, SV[, WF[, PW]]]]]

Here's what the parameters mean:

VC —Select VoiCe 1, 2 or 3

FREQ —Set the FREQuency level of sound (0-65535)

DUR —Set DURation of the sound (in 60ths of a second)

DIR —Set the DIRection in which the sound is incremented/decremented

- 0 = Increment the frequency upward
- 1 = Decrement the frequency downward
- 2 = Oscillate the frequency up and down

MIN —Select the MINimum frequency (0-65535) if the sweep (DIR) is specified

SV —Choose the Step Value for the sweep (0-32767)

WF —Select the Wave Form (0-3)

- 0 = Triangle
- 1 = Sawtooth
- 2 = Variable Pulse
- 3 = White Noise

PW —Set the Pulse Width, the width of the variable pulse waveform

Note that the DIR, MIN, SV, WF and PW parameters are optional.

The first parameter (VC) in the SOUND statement selects which voice will be played. The second parameter (FREQ) determines the frequency of the sound, which ranges from 0 through 65535. The third setting (DUR) specifies the amount of time the sound is played. The duration is measured in 60ths of a second. If you want to play a sound for one second, set the duration to 60, since 60 times 1/60 equals 1. To play the sound for two seconds, specify the duration to be 120. To play the sound 10 seconds, make the duration 600, and so on.

The fourth parameter (DIR) selects the direction in which the frequency of the sound is incremented or decremented. This is referred to as the sweep. The fifth setting (MIN) sets the minimum frequency where the sweep begins. The sixth setting (SV) is the step value of the sweep. It is similar to the step value in a FOR . . . NEXT loop. If the DIR, MIN and SV values are specified in the SOUND command, the sound is played first at the original level specified by the FREQ parameter. Then the synthesizer sweeps through and plays each level of the entire range of frequency values starting at the MIN frequency. The sweep is incremented or decremented by the step value (SV) according to the direction specified by the DIR parameter and the frequency is played at the new level.

The seventh parameter (WF) in the SOUND command selects the waveform for the sound. (Waveforms are explained in detail in the paragraph titled, **Advanced Sound and Music in C128 Mode.**)

The final setting in the SOUND command determines the width of the pulse width waveform if it is selected as the waveform parameter. (See the **Advanced Sound** discussion for an illustration of the pulse width waveform.)

Writing a SOUND Program

Now it's time to write your first SOUND program. Here's an example of the SOUND statement:

```
10 VOL 5
20 SOUND 1, 4096, 60
```

RUN this program. The Commodore 128 plays a short, high-pitched beep. You must set the volume before you can play the sound statement, so line 10 sets the VOLume of the sound chip. Line 20 plays voice 1 at a frequency of 4096 for a duration of 1 second (60 times 1/60). Change the frequency with this statement:

```
30 SOUND 1, 8192, 60
```

Notice line 30 plays a higher tone than line 20. This shows the direct relationship between the frequency setting and the actual frequency of the sound. As you increase the frequency setting, the Commodore 128 increases the pitch of the tone. Now try this statement:

```
40 SOUND 1, 0, 60
```

This shows that a FREQ value of 0 plays the lowest frequency (which is so low it is inaudable). A FREQ value of 65535 plays the highest possible frequency.

Now try placing the sound statement within a FOR . . . NEXT loop. This allows you to play the complete range of frequencies within the loop. Add these statements to your program:

```
50 FOR I = 1 TO 65535 STEP 100
60 SOUND 1, I, 1
70 NEXT
```

This program segment plays the variable pulse waveform in the range of frequencies from 1 through 65535, in increments of 100, from lowest frequency to highest. If you don't specify the waveform, the computer selects the default value of waveform 2, the variable pulse waveform.

Now change the waveform with the following program line (60) and try the program again:

```
60 SOUND 1, I, 1, 0, 0, 0, 0, 0
```

Now the program plays voice 1, using the triangle waveform, for the range of frequencies between 1 and 65535 in increments of 100. This sounds like a typical sound effect in popular arcade games. Try waveform 1, the sawtooth waveform, and see how it sounds with this line:

```
60 SOUND 1, I, 1, 0, 0, 0, 1, 0
```

The sawtooth waveform sounds similar to the triangle waveform though it has less buzz. Finally, try the white noise waveform (3). Substitute line 60 for this line:

```
60 SOUND 1, I, 1, 0, 0, 0, 3, 0
```

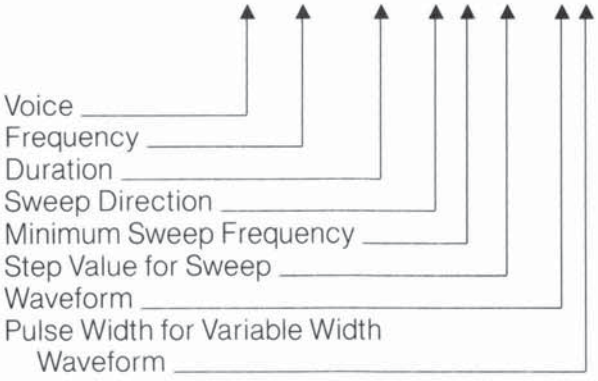
Now the program loop plays the white noise generator for the entire range of frequencies. At first, there is a low-pitched rumbling sound. As the frequency increases in the loop, the pitch increases and sounds like a rocket taking off.

Notice that so far, we have not specified all of the parameters in the SOUND statement. Take line 60, for example:

```
60 SOUND 1, I, 1, 0, 0, 0, 3, 0
```

The three zeros following 1, I, 1 pertain to the sweep parameters within the SOUND statement. Since none of the parameters is specified, the SOUND does not sweep. Add this line to your program:

```
100 SOUND 1, 49152, 240, 1, 0, 100, 1, 0
```



Line 100 starts the sweep frequency at 49152 and decrements the sweep by 100 in the downward direction, until it reaches the minimum sweep frequency at 0. Voice 1, using the sawtooth waveform (#1), plays each SOUND for four seconds (240 * 1/60 sec.). Line 100 sounds like a bomb dropping, as in many "shoot 'em up" arcade games.

Now try changing some of the parameters in line 100. For instance, change the direction of the sweep to 2 (oscillate); change the minimum frequency of the sweep to 32768; and increase the step value to 3000. Your new SOUND command looks like this:

```
110 SOUND 1, 49152, 240, 2, 32768, 3000, 1
```

Line 110 makes a siren sound as though the police were right on your tail. For a more pleasant sound, try this:

```
110 SOUND 1, 65535, 250, 0, 32768, 3000, 2, 2600
```

This should remind you of a popular space-age TV show, when the space crew unleashed their futuristic weapons on the unsuspecting aliens.

Until now, you have been programming in only one voice. You can produce interesting sound effects with the SOUND statement using

up to three voices. Experiment and create a program which utilizes all three voices.

Here's a sample program that will help you understand how to program the Commodore 128 synthesizer chip. The program, when run, asks for each parameter, and then plays the sound. Here's the program listing. Type it into your computer and RUN it.

```
10 PRINT:PRINT:PRINT:PRINT"♥            SOUND PLAYER":PRINT:PRINT:PRINT
20 PRINT"    INPUT SOUND PARAMETERS TO PLAY":PRINT:PRINT
30 INPUT "VOICE (1-3)";V
40 INPUT "FREQUENCY (0-65535)";F
50 INPUT "DURATION (0-32767)";D:PRINT
60 INPUT"WANT TO SPECIFY OPTIONAL PARAMETERS Y/N";B$:PRINT
70 IF B$="N" THEN 130
80 INPUT "SWEEP DIRECTION 0=UP,1=DOWN,2=OSCILL";DIR
90 INPUT "MINIMUM SWEEP FREQUENCY (0-65535)";M
100 INPUT "SWEEP STEP VALUE (0-32767)";S
110 INPUT "WAVEFORM (0=TRI,1=SAW,2=VAR PUL,3=NOISE";W
120 IF W=2 THEN INPUT "PULSE WIDTH (0-4095)";P
130 SOUND V, F, D, DIR, M, S, W, P
140 INPUT"DO YOU WANT TO HEAR THE SOUND AGAIN Y/N";A$
150 IF A$="Y"THEN 130
160 GOTO10
```

Here's a quick explanation of the program. Lines 10 and 20 PRINT the introductory messages on the screen. Lines 30 through 50 INPUT the voice, frequency and duration parameters. Line 60 asks if you want to enter the optional SOUND parameters, such as the sweep settings and waveform. If you don't want to specify these parameters, press the "N" key and the program jumps to line 120 and plays the sound. If you do want to specify the optional SOUND settings, press the "Y" key and the program continues with line 80. Lines 80 through 110 specify the sweep direction, minimum sweep frequency, sweep step value and waveform. Line 120 INPUTs the pulse width of the variable pulse waveform only if waveform 2 (variable pulse) is selected. Finally, line 130 plays the SOUND according to the parameters that you specified earlier in the program.

Line 140 asks if you want to hear the SOUND again. If you do, press the "Y" key; otherwise, press the "N" key. Line 150 checks to see if you pressed the "Y" key. If you did, program control is returned to line 130 and the program plays the SOUND again. If you do not press

the "Y" key, the program continues with line 160, which returns program control to line 10 and the program repeats. To stop the Sound Player program, press the RUN/STOP and RESTORE keys at the same time.

Random Sounds

The following program generates random sounds using the RND function. Each SOUND parameter is calculated randomly. Type the program into your computer, SAVE it and RUN it. This program illustrates how many thousands of sounds you can produce by specifying various combinations of the SOUND parameters. Here's the listing:

```
10 PRINT"VC  FREQ  DIR  MIN   SV   WF   PW "
20 PRINT"_____________________________________"
30 V=INT(RND(1)*3)+1:REM VOICE
40 F=INT(RND(1)*65535)  :REM FREQ
50 D=INT(RND(1)*32767)  :REM DURATION
60 DIR=INT(RND(1)*3)  :REM STEP DIR
70 M=INT(RND(1)*65535)  :REM MIN FREQ
80 S=INT(RND(1)*32767)  :REM STEP VAL
90 W=INT(RND(1)*4)  :REM WAVEFORM
100 P=INT(RND(1)*4095)  :REM PULSE W
110 PRINTV; F;DIR;M;S;W;P:PRINT:PRINT
120 SOUND V, F, D, DIR, M, S, W, P
130 SLEEP 4
140 SOUND V, 0, 0, DIR, 0, 0, W, P
150 GOTO10
```

Lines 10 and 20 PRINT parameter column headings and the underline. Lines 30 through 100 calculate each SOUND parameter within its specific range. For example, line 30 calculates the voice number as follows:

30 V = INT(RND(1)*3)+1

The notation RND (1) specifies the **seed** value of the random number. The **seed** is the base number generated by the computer. The 1 tells the computer to generate a new seed each time the command is encountered. Since the Commodore 128 has three voices, the notation * 3 tells the computer to generate a random number within the range 0 through 3. Notice, however, there is no voice 0, so the

+ 1 in line 30 tells the computer to generate a random number in the range between 1 and 3. The procedure for generating a random number in a specific range is to multiply the given random number times the maximum value of the parameter (in this case, 3). If the minimum value of the parameter is greater than zero, add to the random number a value that will specify the minimum value of the range of numbers you want to generate (in this case, 1). For instance, line 40 generates a random number in the range between 0 and 65535. Since the minimum value is zero in this case, you do not need to add a value to the generated random number.

Line 110 PRINTs the values of the parameters. Line 120 plays the SOUND specified by the random numbers generated in lines 30 through 100. Line 130 delays the program for 4 seconds while the sound is playing. Line 140 turns off the SOUND after the 4 second delay. All sounds generated by this program play for the same amount of time, since they are all turned off after 4 seconds with line 140. Finally, line 150 returns control to line 10, and the process is repeated until you press the RUN/STOP and RESTORE keys at the same time.

So far you have experimented with sample programs using only the SOUND statement. Although you can use the SOUND statement to play musical scores, it is best suited for quick and easy sound effects like the ones in the dogfight program. The Commodore 128 has other statements designed specifically for song playing. The following paragraphs describe the advanced sound and music statements that enable you to play complex musical scores and arrangements with your Commodore 128 synthesizer.

A Brief Background: The Characteristics of Sound

Every sound you hear is actually a sound wave traveling through the air. Like any wave, a sound (sine) wave can be represented graphically and mathematically (see Figure 7-1).

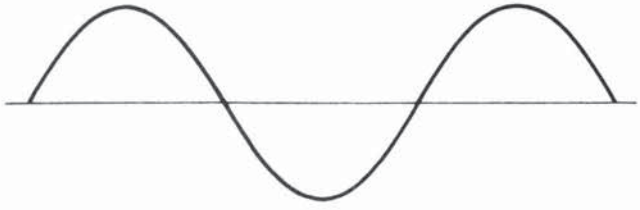


Figure 7-1. Sine Wave

The sound wave moves (oscillates) at a particular rate (frequency) which determines the overall pitch (the highness or lowness of the sound).

The sound is also made up of harmonics, which are accompanying multiples of the overall frequency of the sound or note. The combination of these harmonic sound waves give the note its qualities, called timbre. Figure 7-2 shows the relationship of basic sound frequencies and harmonics.

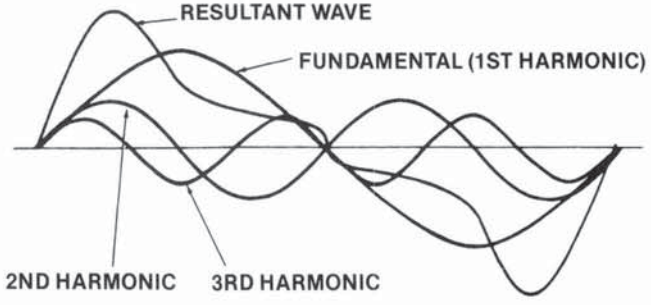
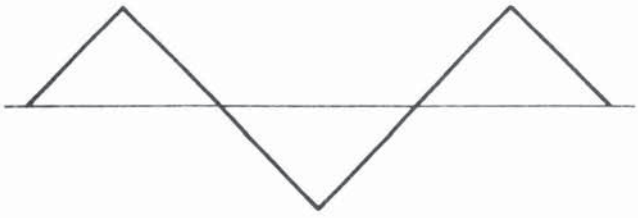
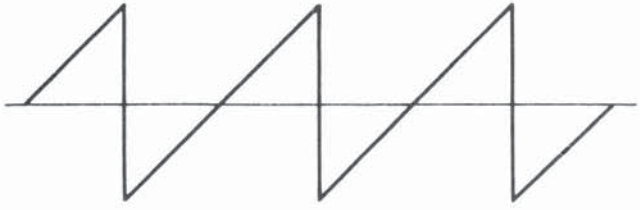


Figure 7-2. Frequency and Harmonics

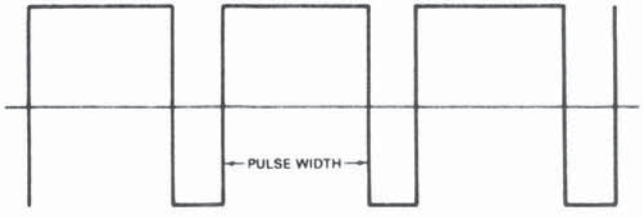
The timbre of a musical tone, (i.e., the way a tone sounds,) is determined by the tone's waveform. The Commodore 128 can generate four types of waveforms: triangle, sawtooth, variable pulse and noise. See Figure 7-3 for a graphic representation of these four waveforms.



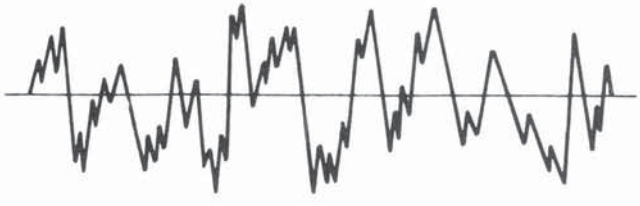
TRIANGLE



SAWTOOTH



VARIABLE
PULSE



NOISE

Figure 7-3. Sound Waveforms Types

The ENVELOPE Statement

The volume of a sound changes throughout the duration of the note, from when you first hear it until it is no longer audible. These volume qualities are referred to as Attack, Decay, Sustain and Release (ADSR). **Attack** is the rate at which a musical note reaches its peak volume. **Decay** is the rate at which a musical note decreases from its peak volume to its midranged (sustain) level. **Sustain** is the level at which a musical note is played at its midranged volume. **Release** is the rate at which a musical note decreases from its sustain level to zero volume. The ENVELOPE generator controls the ADSR parameters of sound. See Figure 7-4 for a graphical representation of ADSR. The Commodore 128 can change each ADSR parameter to 16 different rates. This gives you absolute flexibility over the ENVELOPE generator and the resulting properties of the volume when the sound is originated.

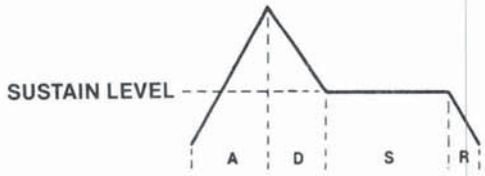


Figure 7-4. ADSR Phases

One of the most powerful Commodore 128 sound statements—the one that controls the ADSR and waveform—is the ENVELOPE statement. The ENVELOPE statement sets the different controls in the synthesizer chip which makes each sound unique. The ENVELOPE gives you the power to manipulate the SID synthesizer. With ENVELOPE, you can select particular ADSR settings and choose a waveform for your own music and sound effects. The format for the ENVELOPE statement is as follows:

ENVELOPE e[,a[,d[,s[,r[,wf[,Pw]]]]]]

Here's what the letters mean:

e — envelope number (0-9)
a — attack rate (0-15)
d — decay rate (0-15)
s — sustain level (0-15)
r — release rate (0-15)
wf — waveform—0 = triangle
1 = sawtooth
2 = pulse (square)
3 = noise
4 = ring modulation
pw — pulse width (0-4095)

Here are the definitions of the parameters not previously defined:

Envelope –The properties of a musical note specified by the waveform and the attack, decay, sustain and release settings of the note. For example, the envelope for a guitar note has a different ADSR and waveform than a flute.

Waveform –The type of sound wave created by the combination of accompanying musical harmonics of a tone. The accompanying harmonic sound waves are multiples of, and are based on the overall frequency of the tone. The qualities of the tone generated by each waveform are recognizably different from one another and are represented graphically in Figure 7-3.

Pulse Width –The length of time between notes, generated by the pulse waveform.

Now you can realize the power of the ENVELOPE statement. It controls most of the musical qualities of the notes being played by the sound

synthesizer. The Commodore 128 has 10 predefined envelopes for 10 different musical instruments. In using the predefined envelopes you do not have to specify the ADSR parameters, waveform and pulse width settings—this is already done for you. All you have to do is specify the envelope number. The rest of the parameters are chosen automatically by the Commodore 128. Here are the preselected envelopes for different types of musical instruments:

Envelope Number	Instrument	Attack	Decay	Sustain	Release	Waveform	Width
0	Piano	0	9	0	0	2	1536
1	Accordion	12	0	12	0	1	
2	Calliope	0	0	25	0	0	
3	Drum	0	5	5	0	3	
4	Flute	9	4	4	0	0	
5	Guitar	0	9	2	1	1	
6	Harpsichord	0	9	0	0	2	512
7	Organ	0	9	9	0	2	2048
8	Trumpet	8	9	4	1	2	512
9	Xylophone	0	9	0	0	0	

Figure 7-5. Default Parameters for ENVELOPE Statement

Now that you have a little background on the ENVELOPE statement, begin another example by entering this statement into your Commodore 128.

```
10 ENVELOPE 0, 5, 9, 2, 2, 2, 1700
```

This ENVELOPE statement redefines the default piano envelope (0) to the following: Attack = 5, Decay = 9, Sustain = 2, Release = 2, waveform remains the same (2) and the pulse width of the variable pulse waveform is now 1700. The piano envelope will not take on these properties until it is selected by a PLAY statement, which you will learn later in this section.

The next step in programming music is setting the volume of the sound chip as follows:

```
20 VOL 8
```

The VOL statement sets the volume of the sound chip between 0 and 15, where 15 is the maximum and 0 is off (no volume).

The TEMPO Statement

The next step in Commodore 128 music programming is controlling the tempo, or speed of your tune. The TEMPO statement does this for you. Here's the format:

```
TEMPO n
```

where n is a digit between 0 and 255 (and 255 is the fastest tempo). If you do not specify the TEMPO statement in your program, the Commodore 128 automatically sets the tempo to 8. Add this statement to your musical example program:

```
30 TEMPO 10
```

The PLAY Statement

Now it's time to learn how to play the notes in your song. You already know how the PRINT statement works. You play the notes in your tune the same way as PRINTing a text string to the screen, except you use the PLAY statement in place of PRINT. PRINT outputs text, PLAY outputs musical notes.

Here's the general format for the play statement:

```
PLAY"string of synthesizer control
characters and musical notes"
```

The total number of characters (including musical notes and synthesizer control characters) that can be put into a PLAY command is 255. However, since this exceeds the maximum number of characters (160) allowed for a single program line in BASIC 7.0, you have to concatenate (that is, add together) at least two strings to reach this length. You can avoid the need to concatenate strings by making sure your PLAY commands do not exceed 160 characters, i.e.,

one program line in length. (This is equivalent to four screen lines in 40-column mode, and two screen lines in 80-column mode.) By doing this, you will produce PLAY command strings that are easier to understand and use.

To play musical notes, enclose the letter of the note you want to play within quotes. For example, here's how to play the musical scale:

```
40 PLAY "C D E F G A B"
```

This plays the notes C, D, E, F, G, A and B in the piano envelope, which is envelope 0. After each time you RUN this example program you are creating, hold down the RUN/STOP key and press the RESTORE key to reset the synthesizer chip.

You have the option of specifying the duration of the note by preceding it in quotes with one of the following letters:

W–Whole note
H–Half note
Q–Quarter note
I –Eighth note
S –Sixteenth note

The default setting, if the duration is not specified, is for Whole (W) notes.

You can PLAY a rest by including the following in the PLAY string:

R–Rest

You can instruct the computer to wait until all voices currently playing reach the end of a measure by including the following in quotes:

M–Wait for end of measure

The Commodore 128 also has synthesizer control characters you can enclose within quotes in a PLAY string. This gives you absolute control over each note and allows you to change synthesizer controls within a string of notes. Follow

the control character with a number in the allowable range for that character. The control characters and the range of numbers for each are shown in Figure 7-6. The "n" following the control character refers to the number you select from the specified range.

Control Character	Description	Range	Default Setting
V n	Voice	1-3	1
O n	Octave	0-6	4
T n	Envelope	0-9	0
U n	Volume	0-15	9
X n	Filter	0 = off, 1 = on	0

Figure 7-6. Sound Synthesizer Control Characters

Although the SID chip can process these control characters in any order, for the best results, place the control characters in your string in the order that they appear in Figure 7-6.

You don't absolutely have to specify any of the control characters, but you should to maximize the power from your synthesizer. The Commodore 128 automatically sets the synthesizer controls to the default settings in Figure 7-6. If you don't assign special control characters, the SID chip can PLAY only one envelope, one voice and one octave without any FILTERing. Specify the control characters to exercise the most control over the notes within your PLAY string.

If you specify an ENVELOPE statement and select your own settings instead of using the default parameters from Figure 7-5, the envelope control character number in your PLAY string must match the envelope number in your ENVELOPE statement in order to assume the parameters you assigned. You don't have to specify the ENVELOPE statement at all if you just want to PLAY the default envelope settings from Figure 7-6. In this case, simply select an envelope number with the (T) control character in the PLAY statement.

Here's an example of the PLAY statement using the SID chip control characters within a string. Add this line to your program and notice the difference between this statement and the PLAY statement in line 40.

```
50 PLAY "V2 O5 T7 U5 X0 C D E F G A B"
```

This statement PLAYS the same notes as in line 40, but voice 2 is selected, the notes are played one octave higher (5) than line 40, the volume setting is turned down to 5 and the FILTER is specified as off. For now, leave the filter off. When you learn about FILTERing in the next section, you can come back and turn the filter on to see how it affects the notes being played. Notice line 50 selects a new instrument, the organ envelope, with the T7 control character. Now your program PLAYS two different instruments in two of the independent voices. Add this statement to PLAY the third voice:

```
60 PLAY "V3 O6 U7 T6 X0 C D E F G A B"
```

Here's how line 60 controls the synthesizer. The V3 selects the third voice, O6 places voice 3 one octave higher (6) than voice two, T6 selects the harpsichord envelope, U7 sets the volume to 7 and X0 leaves the filter off for all three voices. Now your program PLAYS all three voices, each one octave higher than the other, in three separate instruments, piano, organ and harpsichord.

So far, your PLAY statements only played whole notes. Add notes of different duration by placing duration control characters in your PLAY string as follows:

```
70 PLAY "V2 O6 T0 U7 X0 H C D Q E F I G
A S B"
```

Line 70 PLAYs voice 2 in octave 6 at volume level 7 with the redefined piano envelope (0) on and filter turned off. This statement PLAYs the notes C and D as half notes, E and F as quarter notes, G and A as eighth notes and B as a sixteenth note. Notice the difference between the

piano envelope in line 40 and the redefined piano envelope in line 70. Line 40 actually sounds more like a piano than line 70.

You can PLAY sharp, flat and dotted notes by preceding the notes within quotes with the following characters:

– Sharp
$ – Flat
. – Dotted

A dotted note plays one-and-a-half times longer than a note that is not dotted.

Now try adding sharp, flat and dotted notes with this statement:

```
80 PLAY "V1 O4 T4 U8 X0 .H C D Q # E F I
$ G A .S # B"
```

Line 80 PLAYS voice 1 in octave 4 at volume level 8 with the flute envelope turned on and the filter turned off. It also PLAYS C and D as dotted half notes, E and F as sharp quarter notes, G and A as flat eighth notes and B as a sharp dotted sixteenth note. You can add rests (R) at any place within your PLAY string. The spaces in the new PLAY statement examples are not necessary. They are used only for readability.

Up until now your statement examples have left the filter off within the sound synthesizer and have not realized the true power behind it. Now that you have digested most of the sound and music statements and the SID control characters, move on to the next section to learn how to enhance your musical quality with the FILTER statement.

The SID Filter

Once you have selected the ENVELOPE, ADSR, VOLume and TEMPO, use the FILTER to perfect your synthesized sounds. In your program, the FILTER statement will precede the PLAY statement. First you should become comfortable with generating the sound and worry about FILTERing last. Since the SID chip has only one

filter, it applies to all three voices. Your computerized tunes will play without FILTERing, but to take full advantage of your music synthesizer, use the FILTER statement to increase the sharpness and quality of the sound.

In the first paragraph of this section, *The Characteristics of Sound,* we defined a sound as a wave traveling (oscillating) through the air at a particular rate. The rate at which a sound wave oscillates is called the wave's frequency. Recall that a sound wave is made up of an overall frequency and accompanying harmonics, which are multiples of the overall frequency. See Figure 7-2. The accompanying harmonics give the sound its timbre, the qualities of the sound which are determined by the waveform. The filter within the SID chip gives you the ability to accent and eliminate the harmonics of a waveform and change its timbre.

The SID chip filters sounds in three ways: low-pass, band-pass and high-pass filtering. These filtering methods are additive, meaning you can use more than one filter at a time. This is discussed in the next section. Low-pass filters out frequencies above a certain level you specify, called the cutoff frequency. The cutoff frequency is the dividing line that marks the boundary of which frequency level will be played and which will not. In low-pass filtering, the SID chip plays all frequencies below the cutoff frequency and filters out the frequencies above it. As the name implies, the low frequencies are allowed to pass through the filter and the high ones are not. The low-pass filter produces full, solid sounds. See Figure 7-7.

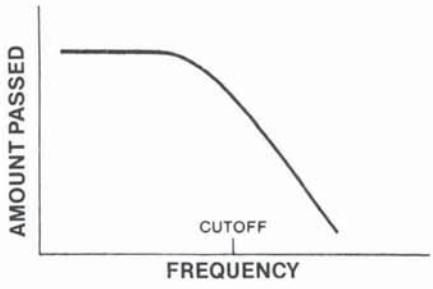


Figure 7-7. Low-pass Filter

Conversely, the high-pass filter allows all the frequencies above the cutoff frequency to pass through the chip. All the ones below it are filtered out. See Figure 7-8. The pass filter produces tinny, hollow sounds.

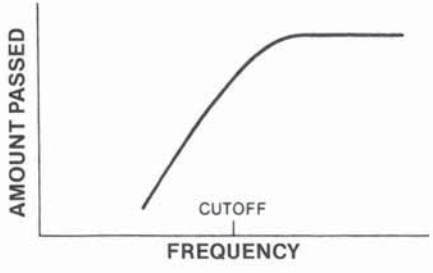


Figure 7-8. High-pass Filter

The band-pass filter allows a range of frequencies partially above and below the cutoff frequency to pass through the SID chip. All other frequencies above and below the band surrounding the cutoff frequency are filtered out. See Figure 7-9.

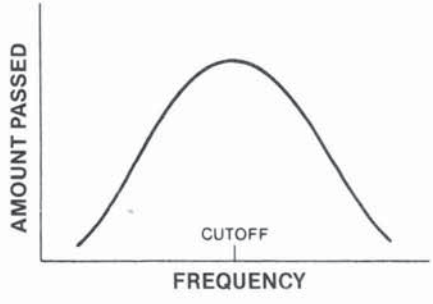


Figure 7-9. Band-pass Filter

The FILTER Statement

The FILTER statement specifies the cutoff frequency, the type of filter being used and the resonance. The resonance is the peaking effect of the sound wave frequency as it approaches the cutoff frequency. The resonance determines the sharpness and clearness of a sound: the higher the resonance, the sharper the sound.

This is the format of the FILTER statement:

FILTER cf, lp, bp, hp, res

Here's what the parameters mean:

cf –Cutoff frequency (0–2047)
lp –Low-pass filter 0 = off, 1 = on
bp –Band-pass filter 0 = off, 1 = on
hp –High-pass filter 0 = off, 1 = on
res –Resonance (0–15)

You can specify the cutoff frequency to be any value between 0 and 2047. Turn on the low-pass filter by specifying a 1 as the second parameter in the FILTER statement. Turn on the band-pass filter by specifying a 1 as the third parameter and enable the high-pass filter with a 1 in the fourth parameter position. Turn off any of the three filters by placing a 0 in the respective position of the filter you want to disable. You can enable or disable one, two or all three of the filters at the same time.

Now that you have some background on the FILTER statement, add this line to your sound program, but do not RUN the program yet.

45 FILTER 1200, 1, 0, 0, 10

Line 45 sets the cutoff frequency at 1024, turns on the low-pass filter, disables the high-pass and band-pass filters and assigns a 10 as the resonance level. Now go back and turn the filter on in your PLAY statements by changing all the X0 filter control characters to X1. Reset the sound chip by pressing the RUN/STOP and RESTORE keys and RUN your sound program again. Notice the differences between the way the

notes sound and how they sounded without the filter. Change line 45 to:

45 FILTER 1200, 0, 1, 0, 10

The new line 45 turns off the low-pass filter and enables the band-pass filter. Press RUN/STOP and RESTORE and RUN your sound program again. Notice the difference between the low-pass and band-pass filters. Change line 45 again to:

45 FILTER 1200, 0, 0, 1, 10

Reset the sound chip and RUN your example program again. Notice the difference between the high-pass filter and the low-pass and band-pass filters. Experiment with different cutoff frequencies, resonance levels and filters to perfect the music and sound in your own programs.

Tying Your Music Program Together

Your first musical program is complete. Now you can program your favorite songs. Let's tie all the components together. Here's the program listing. Don't be alarmed, this is the same program you built in this section except the print statements are added so you know which program lines are being played.

```
10 ENVELOPE 0,5,9,2,2,2,1700
15 VOL 8
20 TEMPO 10
25 PRINT"LINE 30"
30 PLAY "C D E F G A B M"
35 FILTER 1200,0,0,1,10
40 PRINT"LINE 45 - FILTER OFF"
45 PLAY"V2 O5 T7 U5 X0 C D E F G A B M"
50 PRINT"SAME AS LINE 45 - FILTER ON"
55 PLAY"V2 O5 T7 U5 X1 C D E F G A B M"
60 PRINT"LINE 65 - FILTER OFF"
65 PLAY "V3 O6 U7 T6 X0 C D E F G A B M"
70 PRINT"SAME AS LINE 65 - FILTER ON"
75 PLAY "V3 O6 U7 T6 X1 C D E F G A B M"
80 PRINT"LINE 85 - FILTER OFF"
85 PLAY "V2 O6 T0 U7 X0 H CD Q EF I GA S B M"
90 PRINT"SAME AS LINE 85 - FILTER ON"
95 PLAY "V2 O6 T0 U7 X1 H CD Q EF I GA S B M"
100 PRINT"LINE 105 - FILTER OFF"
105 PLAY "V1 O4 T4 U8 X0 H .C D Q # EF I $ GA S .B M"
110 PRINT"SAME AS LINE 105 - FILTER ON"
115 PLAY "V1 O4 T4 U8 X1 H .C D Q # EF I $ GA S .B M"
```

Line 10, the ENVELOPE statement, specifies the envelope for piano (0), which sets the attack to 0, decay to 9, sustain to 0 and release to 0. It also selects the variable pulse waveform with a pulse width of 1700. Line 15 sets the VOLume to 8. Line 20 chooses the TEMPO to be 10.

Line 35 FILTERs the notes that are played in lines 30 through 115. It sets the FILTER cutoff frequency to 1200. In addition, line 35 turns off the low-pass and band-pass filters with the two zeros following the cutoff frequency (1200). The high-pass filter is turned on with the 1 following the two zeros. The resonance is set to 10 by the last parameter in the FILTER statement.

Line 30 PLAYS the notes C, D, E, F, G, A, B in that order. Line 45 PLAYS the same notes as line 30, but it specifies the SID control characters U5 as volume level 5, V1 as voice 1 and 05 as octave 5. Remember, the SID control characters allow you to change the synthesizer controls within a string and exercise the most control over the synthesizer. Line 65 specifies the control characters U7 for volume level 7, V3 for voice 06 for octave 6 and X0 to turn off the filter. Line 65 PLAYS the same notes as lines 30 and 45, but in a different volume, voice and octave.

Line 85 has the same volume, voice and octave as line 65, and it specifies half notes for the notes C and D, quarter notes for the notes E and F, eighth notes for notes G and A and a sixteenth note for the B note. Line 105 sets the volume at 7, voice 1, octave 4 and turns off the filter. It also specifies the C note as a dotted half note, E as a sharp quarter note, G and A as flat eighth notes and B as a dotted sharp sixteenth note.

Advanced Filtering

Each of the previous FILTERing examples used only one filter at a time. You can combine the SID chip's three filters with each other to achieve different filtering effects. For example, you can enable the low-pass and high-pass filters at the same time to form a notch reject filter. A notch reject filter allows the frequencies below and above the cutoff to pass through the SID chip, while the frequencies close to the cutoff frequency are filtered. See Figure 7-10 for a graphic representation of a notch reject filter.

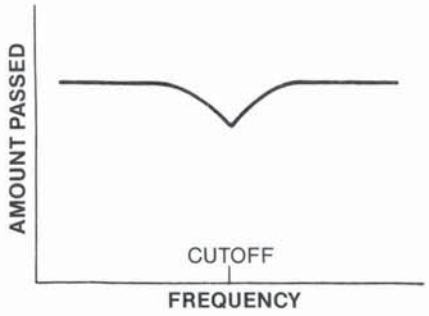


Figure 7-10. Notch Reject Filter

You can also add either the low-pass or high-pass filter to the band-pass filter to obtain interesting effects. By mixing the band-pass filter with the low-pass filter, you can select the band of frequencies beneath the cutoff frequency and below. The rest are filtered out.

By mixing the band-pass and the high-pass filters, you can select the band of frequencies above the cutoff frequency and higher. All the frequencies below the cutoff are filtered out.

Experiment with the different combinations of filters to see all the different types of accents you can place on your musical notes and sound effects. The filters are designed to perfect the sounds created by the other components of the SID chip. Once you have created the musical notes or sound effects with the SID chip, go back and add the FILTERing to your programs to make them as crisp and clean as possible.

Now you have all the information you need to write your own musical programs in Commodore 128 BASIC. Experiment with the different waveforms, ADSR settings, TEMPOs and FILTERing. Look in a book of sheet music and enter the notes from a musical scale in sequence within a play string. Accent the notes in the string with the SID control characters. You can combine your Commodore 128 music synthesizer with C128 mode graphics to make your own videos or "movies," complete with sound tracks.

Coding A Song from Sheet Music

This section provides a sample piece of sheet music and illustrates how to decode notes from a musical staff and translate them into a form the Commodore 128 can understand. This exercise is substantially faster and easier if you know how to read music. However, you don't have to be a musician to be able to play the tune on your Commodore 128. For those of you who cannot read music, Figure 7-11 shows how a typical musical staff is arranged and how the notes on the staff are related to the keys on a piano.

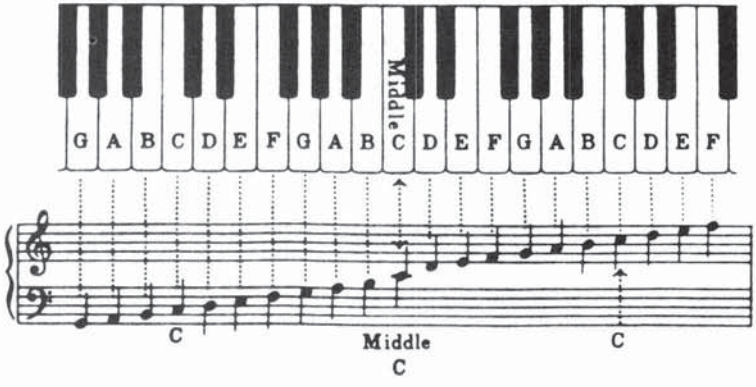


Figure 7-11. Musical Staff

Figure 7-12 is an excerpt from a composition titled *Invention 13 (Inventio 13 in Italian),* by Johann Sebastian Bach. Although this composition was written a few hundred years ago, it can be played and enjoyed on the most modern of computer synthesizers, such as the SID chip in the Commodore 128. Here are the opening measures of *Invention 13.*

© COPYRIGHT
SHEET MUSIC COURTESY
OF C.F. PETERS, CORP.,
NEW YORK
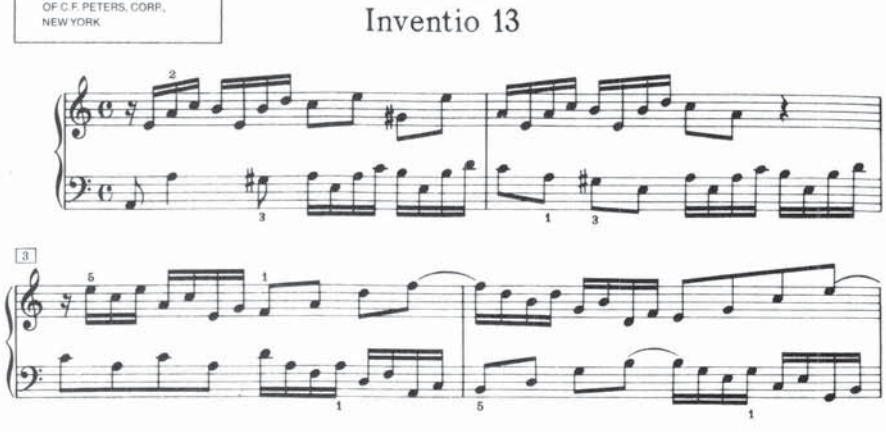


Figure 7-12. Part of Bach's *Invention 13*

The best way to start coding a song into your Commodore 128 is by breaking the notes down into an intermediate code. Write down the upper staff notes on a piece of paper. Now write down the notes for the lower staff. Precede the note values with a duration code. For instance, precede an eighth note with an 8, precede a sixteenth note with a 16, and so on. Next, separate the notes so the notes on the upper staff for one measure are proportional in time with the notes for one measure on the lower staff.

If the musical composition had a third staff, you would separate it so the duration is proportional to the two other upper staffs. Once the notes for all the staffs are separated into equal durations, a separate dedicated voice would play each note for a particular staff. For example, voice 1 would play the upper staff, voice 2 will play the 2d staff and voice 3 would play the lowest staff if it existed.

Let's say the upper staff begins with a string of four eighth notes. In addition, say the lower staff begins with a string of eight sixteenth notes. Since an eighth note is proportional in time to two sixteenth notes, separate the notes as shown in Figure 7-13.

```
V1 =  8A      8B      8C      8D
V2 =  16D 16E 16F 16G 16A 16B 16C 16D
```

Figure 7-13. Synchronizing Notes for Two Voices

Since the synchronization and timing in a musical composition is critical, you must make sure the notes in the upper staff for voice 1, for example, are in time agreement with the notes in the lower staff for voice 2. The first note in the upper staff in Figure 7-12 is an A eighth note. The first two notes for voice 2 are D and E sixteenth notes. In this case, you must enter the voice 1 eighth note in the PLAY string first, then follow the voice 2 sixteenth notes immediately after it. To continue the example, the second note in Figure 7-12 for voice 1 (the upper staff) is a B eighth note. The B eighth note is equal in time to the two sixteenth notes, F and G, which appear in the bottom staff for voice 2. In order to coordinate the timing, enter the B eighth note in the string for voice 2 and follow it with the two sixteenth notes, F and G, for voice 2.

As a rule, always start with the note with the longer duration. For example, if a bar starts with a series of two sixteenth notes on the lower staff for voice 2 and the upper staff starts with an eighth note for voice 1, enter the eighth note in the string first since it must play for the duration while the two sixteenth notes are being fetched by the Commodore 128. You must give the computer time to play the longer note first, and then PLAY the notes of shorter duration, or else the composition will not be synchronized.

Here's the program that plays *Invention 13*. Enter it into your C128, SAVE it for future use, and then RUN it.

```
10 REM INVENTION 13 BY J S BACH
20 TEMPO 6
30 PLAY"V1O4T7U8X0":REM VOICE 1=ORGAN
40 PLAY"V2O4T7U8X0":REM VOICE 2=PIANO
50 REM FIRST MEASURE
60 A$="V2O1IAV1O3IEV2O2QAV1O3SAO4CO3BEV2O2I#GV1O3SBO4DV1O4ICV2O2SAEM"
70 B$="V1O4IEV2O2SAO3CV1O3I#GV2O2SBEV1O4IEV2O2SBO3D"
80 REM SECOND MEASURE
90 C$="V2O3ICV1O3SAEV2O2IAV1O3SAO4CV2O2I#GV1O3SBEV2O2IEV1O3SBO4D"
100 D$="V1O4ICV2O2SAEV1O3IAV2O2SAO3CV1O4QRV2O2SBEBO3D"
110 REM   REM THIRD MEASURE
120 E$="V2O3ICV1O4SREV2O2IAV1O4SCEV2O3ICV1O3SAO4CV2O2IAV1O2SEG"
130 F$="V1O3IFV2O3SDO2AV1O3IAV2O2SFAV1O4IDV2O2SDFV1O4IFV2O1SAO2C"
140 REM  FOURTH MEASURE
150 G$="V2O1IBV1O4SFDV2O2IDV1O3SBO4DV2O2IGV1O3SGBV2O2IBV1O3SDF"
160 H$="V1O3IEV2O2SGEV1O3IGV2O2SEGV1O4ICV2O2SCEV1O4IEV2O1SGB"
170 REM  FIFTH MEASURE
180 I$="V2O1IAV1O4SECV2O2ICV1O3SAO4CV1O3IFV2O2SDFV1O4IDV2O1SBO2D"
190 J$="V2O1IGV1O3SDBV2O1IBV1O3SGBV1O3IEV2O2SCEV1O4ICV2O1SAO2C"
200 REM SIXTH MEASURE
210 K$ = "V2O1IFV1O4SCO3AV2O1IDV1O3SFAV1O3IDV2O1SGO2GV1O3IBV2O2SFG"
220 M$="V2O1IAV1O4SCO3AV2O2I#FV1O4SCEV2O1IBV1O4SDO3BV2O2I#GV1O4SDF"
230 REM SEVENTH MEASURE
240 N$="V2O2ICV1O4SECV2O2IAV1O4SEGV2O2IDV1O4SEEV2O2I$BV1O4SDC"
250 O$="V2O2I#GV1O3SBO4CV2O2IFV1O4SDEV2O2IDV1O4SFDV2O1IBV1O4S#GD"
260 REM EIGHTH MEASURE
270 P$="V2O2I#GV1O4SBDV2O2IAV1O4SCAV2O2IDV1O4SFDV2O2IEV1O3SBO4D"
280 Q$="V2O2IFV1O3S#GBV2O2I#DV1O4SCO3AV2O2IEV1O3SEAV2O2IEV1O3SB#G"
290 REM NINTH MEASURE
300 R$="V2O1HAV1O3SAECEO2QA"
310 PLAY A$:PLAY B$:PLAY C$:PLAY D$:PLAY E$
320 PLAY F$:PLAY G$:PLAY H$:PLAY I$:PLAY J$
330 PLAY K$:PLAY M$:PLAY N$:PLAY O$:PLAY P$
340 PLAY Q$:PLAY R$
```

You can use the techniques described in this section to code your favorite sheet music and play it on your Commodore 128.

****

You now have been introduced to most of the powerful new commands of the BASIC 7.0 language that you can use in C128 mode. In the following section you will learn to use both 40- and 80-column screen displays with the Commodore 128.

SECTION 8
Using 80 Columns

INTRODUCTION 163

THE 40/80 KEY 163

VIDEO PORTS AND MONITORS 164
- **Connecting a Monitor** 164
- **Types of Monitors** 164
  - Composite Monitors 164
  - RGBI Monitors 165
  - Dual Monitors 165

USING PREPACKAGED 80-COLUMN SOFTWARE 165

CREATING 80-COLUMN PROGRAMS 165

USING 40 AND 80 COLUMNS TOGETHER 166

Introduction

In C128 and CP/M modes, you can choose between a 40- and 80-column screen display. You could even use both in a single program.

Each screen size has special uses. The 40-column screen is the same size screen the Commodore 64 uses. With the 40-column screen you can use the Commodore 128's full graphics capabilities. You can draw circles, graphs, sprite characters, boxes and other shapes in high-resolution or multicolor graphic modes. You can also use sprites.

If you are using 80-columns, you get twice the number of characters per program line. In 80-column mode, you can use the standard graphic characters and colors available through the keyboard.

You can also write programs using two monitors to take advantage of both screen display formats with each monitor screen performing different aspects of the program. For example, text output could be displayed on the 80-column monitor while graphics output could be seen on the 40-column monitor.

The 40/80 Key

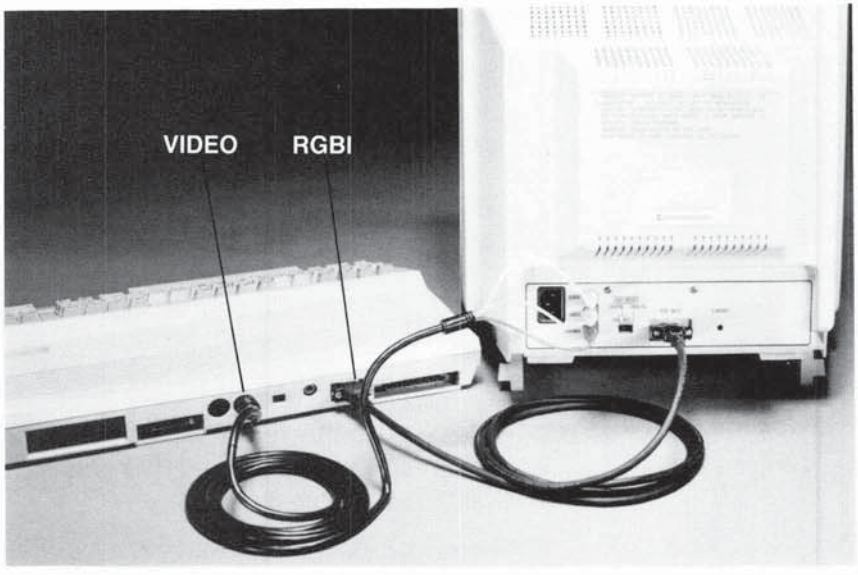
You can use the 40/80 key to set the screen width as either 40 or 80 columns. Pressing this key will only have an effect when one of the following actions is taken:

1. Power is turned ON.
2. The RESET button is pressed.
3. The RUN/STOP and RESTORE keys are pressed simultaneously.

The 40/80 key acts like a SHIFT/LOCK key: it locks when you press it, and does not release until you press it again. If this key is up (not pressed) when one of the three conditions above occurs, the screen is set to 40 columns. If *before power-up* you press the key down, causing it to lock, and one of the three conditions listed above then occurs, the screen is set to 80 columns. Once the computer is running in one screen format (40 or 80 columns), you cannot switch to the other format using the 40/80 key. In this case you must press and release the ESC key and then press the X key.

Connecting a Monitor

Make sure that you connect your monitor properly to the ports on the back of your computer. There are two openings: one is labeled VIDEO and one is labeled RGBI.



VIDEO is the connecting port for 40-column composite video monitor while RGBI is used for 80-column monitors. Dual monitors like the Commodore 1902, which can display either 40-column composite or 80 column RGBI screens, are connected to both ports.

Types of Monitors

Composite Monitors

Composite monitors are designed to display 40-column output on their screens. Examples of composite monitors are the Commodore 1701 and 1702 monitors. These monitors can be used for all 40-column programs and programming in all three modes. However, they cannot be used for 80-column work.

RGBI Monitors

RGBI monitors are specially designed to display 80-column output. Although RGBI stands for Red Green Blue Intensity, RGBI monitors can be either color or monochrome (single color). The most popular monochrome monitors use green or amber displays. An RGBI monitor connected to the RGBI port can handle 80-column output in both C128 and CP/M modes.

Dual Monitors

Dual monitors like the Commodore 1902 can provide either a composite video (40-column) or RGBI (80-column) display. A dual monitor connects to both video ports. A switch on the monitor lets you select either screen output. The 40/80 key on your computer determines the type of screen display upon power-up. Make sure sure the 40/80 key setting corresponds to the 40/80 column slide switch setting on the front control panel of the monitor. NOTE: You can still switch back and forth between 40 and 80 column output by pressing and releasing the ESC key and then pressing the X key, regardless of which position the 40/80 key is in.

Using Prepackaged 80-Column Software

Most CP/M programs utilize an 80-column screen, as do many of the other buisness application packages you can use in C128 mode. Since the width of a normal printed page is 80 columns, an 80-column wordprocessor can display information on the screen exactly as that information will appear on paper. Spreadsheet programs often specify an 80-column format, in order to provide enough space for the necessary columns and categories of information. Many database packages and telecommunications programs also require or can use an 80-column screen.

Creating 80-Column Programs

In addition to running prepackaged software, the 80-column screen width can be useful in designing your own programs. You've probably noticed what happens when you type a line that is wider than 40 columns on a 40-column screen. The lines "wrap around"—that is, they continue onto the next screen line. This may cause confusion in reading the line, and can even lead to programming errors. An 80-column screen helps eliminate these problems. In general, an 80-column screen allows for a clearer screen and better organization.

Using 40 and 80 Columns Together

The main advantage of 40-column composite video output is the availability of bit mapped graphics, while 80 columns gives you output for word processing and other business applications. If you have two monitors, you can write programs that are "shared", using the text features 80 columns affords you and the graphics of 40 columns. A special command, (GRAPHIC 1,1) can be used within a program to transfer the execution of graphics commands to the 40-column display. If you have a dual monitor (one that can display both 40- and 80-column formats) you can place GRAPHIC 1,1 statements in your program so that graphics will be output in 40-column screen format. In order to view the graphic output, however, you will need to change the video switch on the monitor to 40 columns. If you write a program like this, it might be a good idea to include on-screen directions to the user to change the video switch.

For example, you might write a program which asked the user to input data, then created a bar graph based on the user's input. The message "CHANGE TO 40 COLUMN TO VIEW GRAPH" would tell the user to switch modes and see the results.

As noted previously, you can switch between the 80- and 40-column formats after power up, with the ESCape/X sequence.

The following example shows how dual screens can be used within a program:

```
10 GRAPHIC 5,1 :REM THIS STATEMENT SWITCHES TO 80 COLUMN TEXT MODE
20 PRINT "■START IN 40 COLUMN OUTPUT":PRINT
30 PRINT"SLIDE THE SWITCH ON THE FRONT OF THE 1902 DUAL MONITOR TO THE MIDDLE"
40 PRINT:PRINT"PRESS RETURN WHEN READY"
50 GRAPHIC 0,1
60 PRINT:PRINT"PRESS RETURN WHEN READY":GETKEY A$:IF A$<> CHR$(13)THEN 60
70 COLOR 1,5: COLOR 4,1:COLOR 0,1
80 GRAPHIC 2,1 :CHAR 1,8,18,"BIT MAP/TEXT SPLIT SCREEN":REM SELECT SPLIT SCREEN
90 FOR I=70 TO 220 STEP 20 :CIRCLE 1,I,50,30,30:NEXT
100 PRINT"  SWITCH TO 80 COLUMN OUTPUT"
110 PRINT"  SLIDE THE MONITOR SWITCH ON THE FRONT    TO THE EXREME RIGHT"
120 PRINT"  PRESS THE RETURN KEY WHEN READY":GETKEY A$:IF A$<> CHR$(13)THEN 120
130 GRAPHIC 5,1 :REM THIS STATEMENT SWITCHES TO 80 COLUMN TEXT MODE
140 FOR J=1TO 10
150 PRINT "NOW YOU ARE IN 80 COLUMN TEXT OUTPUT"
160 NEXT :PRINT
170 PRINT"NOW SWITCH BACK TO 40 COLUMN OUTPUT":PRINT
180 PRINT "SLIDE THE SWITCH ON THE FRONT OF THE MONITOR TO THE MIDDLE" :PRINT
190 PRINT"PRESS THE RETURN KEY WHEN READY":GETKEY A$:IF A$<> CHR$(13)THEN 190
200 GRAPHIC 0,1:REM THIS STATEMENT SWITCHES TO 40 COLUMN TEXT MODE
210 FOR J=1TO 70
220 PRINT "NOW YOU ARE IN 40 COLUMN TEXT OUTPUT"
230 NEXT
```

Each screen display format offers certain advantages; yet the two types of displays can be combined in a program to complement each other. Using a 40-column screen, you get the full power of advanced BASIC graphics. The 80-column display gives you more space for your own programs. In addition, it lets you run the wide variety of software designed to run on an 80-column screen.

****

The sections of this chapter have introduced you to the many features and capabilities provided by the Commodore 128 in C128 mode. The following chapter tells you how to use the Commodore 128 in C64 mode.

CHAPTER

3

USING C64 MODE



SECTION 9
Using the Keyboard In C64 Mode

USING BASIC 2.0	173
KEYBOARD CHARACTER SETS	173
USING THE TYPEWRITER-STYLE KEYS	173
USING THE COMMAND KEYS	173
MOVING THE CURSOR IN C64 MODE	173
PROGRAMMING FUNCTION KEYS IN C64 MODE	174

USING BASIC 2.0

The entire BASIC 2.0 language built into the Commodore 64 computer has been incorporated into the BASIC 7.0 language of the Commodore 128. You can use the BASIC 2.0 commands in both C128 and C64 modes. Refer to Sections 3 and 4 in Chapter II for a description of these commands.

Keyboard Character Sets

In the keyboard illustration in Section 3, the **shaded** keys are the ones that can be used in C64 mode. The keyboard in C64 mode has the same two character sets as in C128 mode:

—Upper-case/graphic character set
—Upper/lower-case character set

When you enter C64 mode, the keyboard is in the upper-case/graphic character set, so that everything you type is in capital letters. In C64 mode you can only use one character set at a time. To switch back and forth between character sets, press the SHIFT key and the C= key (the COMMODORE key) at the same time.

Using The Typewriter-Style Keys

As in C128 mode, you can use the typewriter-style keys in C64 mode to type both upper-case letters (capitals) and lower-case letters (small letters). You can also type the numerals shown on the top row of the main keyboard. In addition, you can type the graphics symbols on the fronts of the keys.

Using The Command Keys

Most COMMAND keys (i.e., the keys that send messages to the computer, like RETURN, SHIFT, CTRL, etc.) work the same in C64 mode as they do in C128 mode.

The only difference is that in C64 mode, you can only move the cursor by using the two CRSR keys at the bottom-right corner of the main keyboard. (In C128 mode, you can also use the four arrow keys located just above the top right side of the main keyboard.)

Moving The Cursor In C64 Mode

In C64 mode, you use two CRSR keys on the main keyboard and the SHIFT key to move the cursor, as described in Section 3.

Programming Function Keys In C64 Mode

The four keys to the right side of the keyboard, just above the numeric keypad, are called **function keys.** The keys are marked F1, F3, F5 and F7 on the tops and F2, F4, F6 and F8 on the fronts. These keys can be **programmed**—that is, they can be instructed to perform a specific task or function. For this reason, these keys are often called **programmable function keys.**

You must hold down the SHIFT key to perform the functions associated with the markings on the front of the keys—that is, F2, F4, F6 and F8. Therefore, these keys are sometimes called the **SHIFTed programmable function keys.**

The function keys in C64 mode do not have a printed character assigned to them. They do, however, have CHR$ codes assigned. In fact, each of them has two CHR$ codes—one for when you press the key by itself, and one for when you press the key while holding down the SHIFT key. To get the even-numbered function keys, hold down the SHIFT key while pressing the function key. For example, to get F2, hold down SHIFT and press F1.

The CHR$ codes for the F1-F8 keys range from 133 to 140. However, the codes are not assigned to the keys in numerical order. The keys and their corresponding CHR$ codes are as follows:

F1 CHR$(133)
F2 CHR$(137)
F3 CHR$(134)
F4 CHR$(138)
F5 CHR$(135)
F6 CHR$(139)
F7 CHR$(136)
F8 CHR$(140)

You can use the function keys in your program in several ways. To do this, you'll need to use the GET statement. (See Section 4 for a description of the GET statement.) As an example, 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, 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.

SECTION 10
Storing And Reusing Your Programs In C64 Mode

FORMATTING A DISK IN C64 MODE 179

THE SAVE COMMAND 179
- SAVEing on Disk 179
- SAVEing on Cassette 180

THE LOAD AND RUN COMMANDS 180
- LOADing and RUNning from Disk 180
- LOADing and RUNning from Cassette 180

OTHER DISK-RELATED COMMANDS 181
- Verifying a Program 181
- Displaying Your Disk Directory 181
- Initializing a Disk Drive 181

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

Formatting a Disk in C64 Mode

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 in C64 mode, you type this command:

OPEN 15,8,15: PRINT# 15,"N0:NAME,ID" RETURN

In place of NAME, type a disk name of your choice; you can use up to 16 characters to identify the disk. In place of ID, type a two-character code of your choice (such as W2 or 10).

The cursor disappears during the formatting process. When the cursor blinks again, type the following command:

CLOSE 15 RETURN

NOTE: Once a disk is formatted in C64 or C128 mode, that disk can be used in either mode.

The SAVE Command

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

SAVEing on 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 disk or tape. 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 cursor on your computer disappears while the program is being SAVEd, but it returns when the process is completed.

SAVEing on Cassette

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 LOAD And RUN Commands

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

LOADing and RUNning from 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.

To RUN the program, type RUN and press ⟨RETURN⟩.

LOADing and RUNning from Cassette

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 the programs. Then, when you want to retrieve a program, simply wind the tape forward from 000 to the program'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, the program being LOADed is not erased from the tape; it is simply copied into the computer. However, LOADing a program automatically erases any BASIC program that may have been in the computer's memory.

Other Disk-Related Commands

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."

The VERIFY command also works for tape programs. You type:

VERIFY"PROGRAM NAME" RETURN

Note that you do not need to enter the comma and the number 8, since 8 indicates that you are working with a disk program.

Displaying Your Disk Directory

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

LOAD"$",8 RETURN

The cursor disappears during this process. When the cursor re-appears, type:

LIST RETURN

A list of the programs on your disk will then be displayed. **Note that when you load the directory, any program that was in memory is erased.**

Initializing A Disk Drive

If the disk drive's ready 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, you type:

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

If the 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 disk drive or Datassette manual. Also consult the LOAD and SAVE command descriptions in Chapter V, BASIC 7.0 Encyclopedia.

CHAPTER

4

USING CP/M MODE



SECTION 11
Introduction To CP/M 3.0

WHAT CP/M 3.0 IS 187

WHAT YOU NEED TO RUN CP/M 3.0 187

GETTING STARTED WITH CP/M 3.0 188
- Loading or Booting CP/M 3.0 188
- The Opening CP/M Screen Display 188

THE COMMAND LINE 190
- Types of Commands 190
- How CP/M Reads Command Lines 191

What CP/M 3.0 Is

CP/M is a product of Digital Research, Inc. The version of CP/M used on the Commodore 128 is CP/M Plus Version 3.0. In this chapter, CP/M is generally referred to as CP/M 3.0, or simply CP/M. This chapter summarizes CP/M on the Commodore 128. For detailed information on CP/M 3.0, fill out and return the postage-paid card included in this chapter.

CP/M 3.0 is a popular operating system for microcomputers. As an operating system, CP/M 3.0 manages and supervises your computer's resources, including memory and disk storage, the console (screen and keyboard), printer, and communication devices. CP/M 3.0 also manages information stored in disk files. CP/M 3.0 can copy files from a disk to your computer's memory, or to a peripheral device such as a printer. To do this, CP/M 3 places various programs in memory and executes them in response to commands you enter at your console. Once in memory, a program executes through a set of steps that instructs your computer to perform a certain task.

You can use CP/M to create your own programs, or you can choose from the wide variety of available CP/M 3.0 application programs.

What You Need to Run CP/M 3.0

The general hardware requirements for CP/M 3.0 are a computer containing a Z80 microprocessor, a console consisting of a keyboard and a display screen, and at least one floppy disk drive. For CP/M 3.0 on the Commodore 128 Personal Computer, the Z80 microprocessor is built-in; the console consists of the full Commodore 128 keyboard and an 80-column monitor; and the disk drive is the new Commodore 1571 fast disk drive. In addition, there are two CP/M disks packed in the computer carton—one containing the CP/M 3.0 system and an extensive HELP utility program, and the other containing a number of other utility programs.

NOTE: Although CP/M can be used with a 40-column monitor, only 40 columns can be displayed at one time. To view all 80 columns of the display, you must scroll the screen horizontally by pressing the CONTROL key and the appropriate cursor key (left or right).

Getting Started With CP/M 3.0

The following paragraphs tell you how to start or "boot" CP/M 3.0, how to enter and edit the command line, and how to make back-up copies of your CP/M 3.0 disks.

Loading Or Booting CP/M 3.0

Loading or "booting" CP/M 3.0 means reading a copy of the operating system from your CP/M 3.0 system disk into your computer's memory.

You can boot CP/M 3.0 in several ways. If your computer is off, you can boot CP/M by first turning on your disk drive and inserting the CP/M 3.0 system disk, and then turning on the computer. CP/M 3.0 will load automatically. If you are already in C128 BASIC mode, you can boot CP/M 3.0 by inserting the CP/M system disk into the drive and then typing the BASIC command BOOT. CP/M 3.0 will then load. In C128 mode, you can also boot CP/M by inserting the system disk and pressing the RESET button.

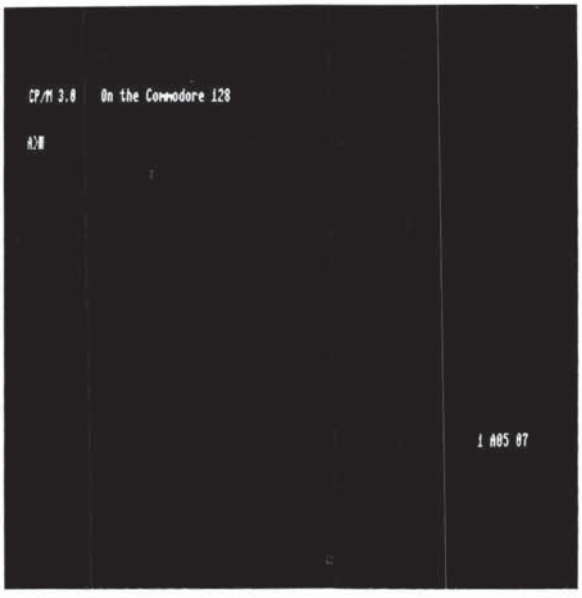
If you are in C64 mode, and you want to enter CP/M mode, first turn off the computer. Then load the CP/M system disk in the drive and turn on the computer.

Caution: Always make sure that the disk is fully inserted in the 1571 drive before you close the drive door.

In CP/M 3.0 on the Commodore 128, the user has a 59K TPA (Transient Program Area), which in effect is user RAM.

The Opening CP/M Screen Display

After CP/M 3 is loaded into memory, a message similar to the following is displayed on your screen:



An important part of the opening display is the following two-character message:

A〉

This is the CP/M 3.0 **system prompt.** The system prompt tells you that CP/M is ready to read a command entered by you from your keyboard. The prompt also tells you that drive A is your default drive. This means that until you tell CP/M to do otherwise, it looks for program and data files on the disk in drive A. It also tells you that you are logged in as user 0, simply by the absence of any user number other than 0.

NOTE: In CP/M a single disk drive is identified as drive A. This is equivalent to unit number 8, drive 0 in C128 and C64 modes. Usually, the maximum number of drives in CP/M 3.0 is four. Additional drives are identified as drives B, C, etc.

The Command Line

CP/M 3.0 performs tasks according to specific commands that you type at your keyboard. These commands appear on the screen in what is called a **command line.** A CP/M 3.0 command line is composed of a **command keyword** and an optional **command tail.** The command keyword identifies a command (program) to be executed. The command tail can contain extra information for the command, such as a filename or parameters. The following example shows a command line.

A>*DIR MYFILE*

Throughout this chapter, the characters that a user would type are in slanted (italic) bold face type to distinguish them from characters that the system displays. In this example, DIR is the command keyword and MYFILE is the command tail. To send the command line to CP/M 3.0 for processing, press the RETURN key, as indicated in this book by the **RETURN** symbol.

As you type characters at the keyboard, they appear on your screen. The cursor moves to the right as you type. If you make a typing error, press either the INST DEL key or CTRL-H to move the cursor to the left and correct the error. CTRL is the abbreviation for the CONTROL key. To specify a control character, hold down the CTRL key and press the appropriate letter key. (A list of control characters and their uses is given in Section 13.)

You can type the keyword and command tail in any combination of upper-case and lower-case letters. CP/M 3.0 interprets all letters in the command line as uppercase.

Generally, you must type a command line directly after the system prompt. However, CP/M 3.0 does allow spaces between the prompt and the command keyword.

Types Of Commands

CP/M 3.0 recognizes two different types of commands: built-in commands and transient utility commands. **Built-in** commands execute programs that reside in memory as a part of the CP/M operating system. Built-in commands can be executed immediately. **Transient utility** commands are stored on disk as program files. They must be loaded from disk to perform their task. You can recognize transient utility program files when a directory is displayed on the screen because their filenames are followed by a period and COM (.COM). Section 14 presents lists of the CP/M built-in and transient utility commands.

For transient utilities, CP/M 3.0 checks only the command keyword. Many utilities require unique command tails. If you include a command tail, CP/M 3.0 passes it to the utility without checking it. A command tail cannot contain more than 128 characters.

How CP/M Reads Command Lines

Let's use the DIR command to demonstrate how CP/M reads command lines. DIR, which is an abbreviation for directory, tells CP/M to display a directory of disk files on your screen. Type the DIR keyword after the system prompt, and press RETURN:

A>*DIR* **RETURN**

CP/M responds to this command by displaying the names of all the files that are stored on whatever disk is in drive A. For example, if the CP/M system disk is in disk drive A, a list of filenames like this appears on your screen:

```
A: PIP COM : ED COM : CCP COM : HELP COM : HELP HLP
A: DIR COM : CPM SYS
```

CP/M 3.0 recognizes only correctly spelled command keywords. If you make a typing error and press RETURN before correcting your mistake, CP/M 3.0 repeats or "echoes" the command line, followed by a question mark. For example, suppose you mistype the DIR command, as in the following example:

A>*DJR* **RETURN**

CP/M replies with:

```
DJR?
```

This tells you that CP/M cannot find a command keyword spelled DJR. To correct typing errors like this, you can use the INST/DEL key to delete the incorrect letters. Another way to delete characters is to hold down the CTRL key and press H to move the cursor to the left. CP/M provides a number of other control characters that help you edit command lines. Section 13 tells how to use control characters to edit command lines and other information you enter at your console.

DIR accepts a filename as a command tail. You can use DIR with a filename to see if a specific file is on the disk. For example, to check that the file program MYFILE is on your disk, type:

A>*DIR MYFILE* **RETURN**

CP/M 3.0 performs this task by displaying either the name of the file you specified, or the message:

No File

Be sure you type at least one space after DIR to separate the command keyword from the command tail. If you do not, CP/M 3.0 responds as follows:

A>*DIRMYFILE* **RETURN**
DIRMYFILE?

SECTION 12
Files, Disks and Drives In CP/M 3.0

WHAT IS A FILE? 195

CREATING A FILE 195

NAMING A FILE 195
File Specification 195
Drive Specifier 196
Filename 196
Filetype 196
Password 197
Sample File Specification 197
User Number 197
Using Wildcard Characters to Access More Than One File 198
Reserved Characters 198
Reserved Filetypes 199

HOW TO MAKE COPIES OF YOUR CP/M 3.0 DISKS AND FILES 200

Making Copies With a Single Disk Drive 200
Making Copies With a Dual Disk Drive 200

What Is A File?

One of CP/M's most important tasks is to access and maintain files on your disks. Files in CP/M are fundamentally the same as in C128 or C64 modes—that is, they are collections of information. However, CP/M handles files somewhat differently than do C128 and C64 modes. This section defines the two types of files used in CP/M; tells how to create, name and access a file; and describes how files are stored on your CP/M disks.

As noted above, a CP/M 3.0 file is a collection of information. Every file must have a unique name by which CP/M identifies the file. A directory is also stored on each disk. The directory contains a list of the filenames stored on that disk and the locations of each file on the disk.

There are two kinds of CP/M files: **program** (command) files, and **data** files. A **program** file contains a series of instructions that the computer follows step-by-step to achieve some desired result. A **data** file is usually a collection of related information (e.g., a list of names and addresses, the inventory of a store, the accounting records of a business, the text of a document).

Creating A File

There are several ways to create a CP/M file. One way is to use a text editor. The CP/M text editor ED is used to create and name a file. You can also create a file by copying an existing file to a new location; you can rename the file in the process. Under CP/M, you can use the PIP command to copy and rename files. Finally, some programs (such as MAC, a CP/M machine language program) create output files as they process input files.

The ED and PIP commands are summarized in Section 14, together with other commonly used CP/M commands. Details on these and all other CP/M 3.0 commands may be found in the CP/M Plus User's Guide, which you can obtain by responding to the offer on the card inserted in this chapter.

Naming A File

File Specification

CP/M identifies every file by a unique **file specification.** A file specification can have four parts: a **drive specifier,** a **filename,** a **filetype** and a **password.** The only mandatory part is the filename.

Drive Specifier

The drive specifier is a single letter (A-P) followed by a colon. Each disk drive in your system is assigned a letter. When you include a drive specifier as part of the file specification, you are telling CP/M to look for the file on the disk currently in the specified drive. For example, if you enter:

B:*MYFILE* **RETURN**

CP/M looks in drive B for the file MYFILE. If you omit the drive specifier, CP/M 3.0 looks for the file in the default drive (usually A).

Filename

A filename can be from one to eight characters long, such as:

MYFILE

A file specification can consist simply of a filename. When you make up a filename, try to let the name tell you something about what the file contains. For example, if you have a list of customer names for your business, you could name the file:

CUSTOMER

so that the name gives you some idea of what is in the file.

Filetype

To help you identify files belonging to the same category, CP/M allows you to add an optional one- to three-character extension, called a filetype, to the filename. When you add a filetype to the filename, separate the filetype from the filename with a period. Try to use letters that tell something about the file's category. For example, you could add the following filetype to the file that contains a list of customer names:

CUSTOMER.NAM

When CP/M displays file specifications, it adds blanks to short filenames so that you can compare filetypes quickly. The **program** files that CP/M loads into memory from a disk have the filetype COM.

Password

In the Commodore 128's CP/M 3.0 you can include a password as part of the file specification. The password can be from one to eight characters. If you include a password, separate it from the filetype (or filename, if no filetype is included) with a semicolon, as follows:

CUSTOMER.NAM;ACCOUNT

A password is optional. However, if a file has been protected with a password, you MUST enter the password as part of the file specification to access the file.

Sample File Specification

A file specification containing all four possible elements consists of a drive specification, a primary filename, a filetype and a password, all separated by the appropriate characters or symbols as in the following example:

A:DOCUMENT.LAW;SUSAN **RETURN**

User Number

CP/M 3.0 further identifies all files by assigning each one a user number which ranges from 0 to 15. CP/M 3.0 assigns the user number to a file when the file is created. User numbers allow you to separate your files into 16 file groups.

The user number always precedes the drive identifier except for user 0, which is the default user number and is not displayed in the prompt. Here are some examples of user numbers and their meanings.

4A〉	User number 4, drive A
A〉	User number 0, drive A
2B〉	User number 2, drive B

You can use the built-in command USER to change the current user number like this:

A〉 ***USER 3*** **RETURN**
3A〉

You can change both the user number and the drive by entering the new user number and drive specifier together at the system prompt:

```
A>3B: RETURN
3B>
```

Most commands can access only those files that have the current user number. However, if a file resides in user 0 and is marked with a system file attribute, the file can be accessed from any user number.

Using Wildcard Characters to Access More Than One File

Certain CP/M 3.0 built-in and transient commands can select and process several files when special wildcard characters are included in the filename or filetype. A wildcard is a character that can be used in place of some other characters. CP/M 3.0 uses the asterisk (*) and the question mark (?) as wildcards. For instance, if you use a ? as the third character in a filename, you are telling CP/M to let the ? stand for **any** character that may be encountered in that position. Similarly, an * tells CP/M to fill the filename with ? question marks as indicated. A file specification containing wildcards is called an ambiguous filespec and can refer to more than one file, because it gives CP/M 3.0 a pattern to match. CP/M 3.0 searches the disk directory and selects any file whose filename or filetype matches the pattern. For example, if you type:

```
?????TAX.LIB
```

then CP/M 3.0 selects all files whose filename end in TAX and whose filetype is .LIB.

Reserved Characters

The characters in Table 12-1 have special meaning in CP/M 3.0, so do not use these characters in file specifications except as indicated.

Table 12-1. CP/M 3.0 Reserved Characters

Character	Meaning
< $, !] > [ ] tab space carriage return	file specification delimiters
:	drive delimiter in file specification
.	filetype delimiter in file specification
;	password delimiter in file specification
;	comment delimiter at the beginning of a command line
* ?	wildcard characters in an ambiguous file specification.
< > & !] ^-	option list delimiters
[ ]	option list delimiters for global and local options.
( )	delimiters for multiple modifiers inside square brackets for options that have modifiers.
/ $	option delimiters in a command line.

Reserved Filetypes

CP/M 3.0 has already established several file groups. Table 12-2 lists some of their filetypes with a short description of each.

Table 12-2. CP/M 3.0 Reserved Filetypes

Filetype	Meaning
ASM	Assembler source file
BAS	BASIC source program
COM	Z80 or equivalent machine language program
HEX	Output file from MAC (used by HEXCOM)
HLP	HELP message file
$$$	Temporary file
PRN	Print file from MAC or RMAC
REL	Output file from RMAC (used by LINK)
SUB	List of commands to be executed by SUBMIT
SYM	Symbol file from MAC, RMAC or LINK
SYS	System file

How To Make Copies Of Your CP/M 3.0 Disks And Files

You can back up your CP/M 3.0 disks, using either one or two disk drives. The back-up disks can be new or used. You might want to format new disks, or reformat used disks with an appropriate CP/M disk formatting program. If the disks have been used previously, *be sure that there are no other files on the disks.*

To make backups use the copysys and PIP utility programs found on your CP/M system disk. PIP can copy all program and data files. Copysys can only copy the operating system.

Making Copies With a Single Disk Drive

You may copy the contents of a disk to another disk with a single Commodore disk drive (1541 or 1571). The PIP command copies the contents of one file to another. To check the format for PIP type:

A> *HELP PIP*

In response, the computer gives the syntax for the PIP command. Use drive A as the source drive and drive E as the destination drive. Drive E is referred as a *virtual* drive—that is, it does not exist as an actual piece of hardware. During the copying process, you will be prompted to remove the source disk and replace it with the destination (new) disk.

Making Copies With A Dual Disk Drive

This section shows how to make distribution disk back-ups on a system that has two drives: drive A and drive B. Your drives might be named with other letters from the range A through P. To make a copy of your CP/M 3.0 distribution system disk, first use the COPYSYS utility to copy the operating system loader. Make sure that your distribution system disk is in drive A, the default drive, *and the blank disk is in drive B.* Then enter the following command at the system prompt:

A>COPYSYS

CP/M 3 loads COPYSYS into memory and runs it. COPYSYS displays the following output on your screen. When the program prompts you, press RETURN only when you have verified that the correct disk is in the correct drive.

COPYSYS VER 3.0
SOURCE DRIVE NAME (OR RETURN FOR DEFAULT) ?A
SOURCE ON A THEN TYPE RETURN
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) ?B
DESTINATION ON B THEN TYPE RETURN
FUNCTION COMPLETE
DO YOU WISH TO COPY CPM.SYS? YES
(CP/M 3.0 REPEATS THE ABOVE PROMPTS TO COPY CPM.SYS.)
A⟩

You now have a copy of the operating system only. To copy the remaining files from the system disk, enter the following PIP command:

A⟩***PIP B:*** = ***A:*.***

This PIP command copies all the files in you disk directory *to drive B from drive A.* PIP displays the message COPYING followed by each filename as the copy operation proceeds. When PIP finishes copying, CP/M 3 displays the system prompt.

Now you have an exact copy of the system disk in *drive B.* Remove the original system disk from drive A and store it in a safe place. As long as you retain the original in an unchanged condition, you will be able to restore your CP/M program files if something happens to your working copy.

SECTION 13
Using the Console and Printer in CP/M 3.0

CONTROLLING CONSOLE OUTPUT 205

CONTROLLING PRINTER OUTPUT 205

CONSOLE LINE EDITING 205

USING CONTROL CHARACTERS FOR LINE EDITING 206

This section describes how CP/M 3.0 communicates with your console and printer. It tells how to start and stop console and printer output, and edit commands you enter at your console.

Controlling Console Output

Sometimes CP/M 3.0 displays information on your screen too quickly for you to read it. To ask the system to wait while you read the display, hold down the CONTROL (CTRL) key and press S. A CTRL-S keystroke sequence causes the display to pause. When you are ready, press CTRL-Q to resume the display. Pressing the NO SCROLL key will also pause the system and place a **pause** window on the status line at the bottom of the screen (line 25). To resume the display, press NO SCROLL again. If you press any key besides CTRL-Q or NO SCROLL during a display pause, CP/M 3.0 sounds the console bell.

Some CP/M 3.0 utilities (like DIR and TYPE) support automatic paging at the console. This means that if the program's output is longer than the screen can display at one time, the display automatically halts when the screen is filled. When this occurs, CP/M 3.0 prompts you to press RETURN to continue. This option can be turned on or off using the SETDEF command.

Controlling Printer Output

You can also use a control command to **echo** (that is, display) console output to the printer. To start printer echo, press CTRL-P. A beep occurs to tell you that echo is on. To stop, press CTRL-P again. (There is no beep at this point.) While printer echo is in effect, any characters that appear on your screen are listed at your printer.

You can use printer echo with a DIR command to make a list of files stored on a floppy disk. You can also use CTRL-P with CTRL-S and CTRL-Q to make a hard copy of part of a file. Use a TYPE command to start a display of the file at the console. When the display reaches the part you need to print, press CTRL-S to stop the display, CTRL-P to enable printer echo, and then CTRL-Q to resume the display and start printing. You can use another CTRL-S, CTRL-P, CTRL-Q sequence to terminate printer echo.

Console Line Editing

As noted previously, you can correct simple typing errors by using the INST DEL key or CTRL-H. CP/M 3.0 also supports additional line-editing functions that you perform with control characters. You can use the control characters to edit command lines or input lines to most programs.

Using Control Characters for Line Editing

By using the line-editing control characters listed in Table 13-1, you can move the cursor left and right to insert and delete characters in the middle of a command line. In this way you do not have to retype everything to the right of your correction.

In the following sample example, the user mistypes PIP, and CP/M 3.0 returns an error message. The user recalls the erroneous command line by pressing CTRL-W and corrects the error (the underbar character represents the cursor):

A>POP A:=B:*.*_ **POP?**	(PIP mistyped)
A>POP A:=B:*.*_	(CTRL-W recalls the line)
A>_POP A:=B:*.*	(CTRL-B moves cursor to beginning of line)
A>P_OP A:=B:*.*	(CTRL-F moves cursor to right)
A>PP_ A:=B:*.*	(CTRL-G deletes error)
A>PIP_ A:=B:*.*	(type I corrects the command name)

After the command line is corrected, the user can press RETURN even though the cursor is in the middle of the line. A RETURN keystroke, (or one of the equivalent control characters) not only executes the command, but also stores the command in a buffer so that you can recall it for editing or reexecution by pressing CTRL-W.

When you insert a character in the middle of a line, characters to the right of the cursor move to the right. If the line becomes longer than your screen is wide, characters disapper off the right side of the screen. These characters are not lost. They reappear if you delete characters from the line or if you press CTRL-E when the cursor is in the middle of the line. CTRL-E moves all characters to the right of the cursor to the next line on the screen.

Table 13-1 gives a complete list of line-editing control characters for the CP/M 3.0 system on the Commodore 128.

Table 13-1. CP/M 3.0 Line-editing Control Characters

Character	Meaning
CTRL-A	Moves the cursor one character to the left.
CTRL-B	Moves the cursor to the beginning of the command line without having any effect on the contents of the line. If the cursor is at the beginning, CTRL-B moves it to the end of the line.

Table 13-1. CP/M 3.0 Line-editing Control Characters (Continued)

Character	Meaning
CTRL-E	Forces a physical carriage return but does not send the command line to CP/M 3.0. Moves the cursor to the beginning of the next line without erasing the previous input.
CTRL-F	Moves the cursor one character to the right.
CTRL-G	Deletes the character under by the cursor. The cursor does not move. Characters to the right of the cursor shift left one place.
CTRL-H	Deletes the character to the left of the cursor and moves the cursor left one character position. Characters to the right of the cursor shift left one place.
CTRL-I	Moves the cursor to the next tab stop. Tab stops are automatically set at each eighth column. Has the same effect as pressing the TAB key.
CTRL-J	Sends the command line to CP/M 3.0 and returns the cursor to the beginning of a new line. Has the same effect as a RETURN or a CTRL-M keystroke.
CTRL-K	Deletes to the end of the line from the cursor.
CTRL-M	Sends the command line to CP/M 3.0 and returns the cursor to the beginning of a new line. Has the same effect as a RETURN or a CTRL-J keystroke.
CTRL-R	Retypes the command line. Places a # character at the current cursor location, moves the cursor to the next line, and retypes any partial command you typed so far.
CTRL-U	Discards all the characters in the command line, places a # character at the current cursor position, and moves the cursor to the next line. However, you can use a CTRL-W to recall any characters that were to the left of the cursor when you pressed CTRL-U.
CTRL-W	Recalls and displays previously entered command line both at the operating system level and within executing programs, if the CTRL-W is the first character entered after the prompt. CTRL-J, CTRL-M, CTRL-U and RETURN define the com-

Table 13-1. CP/M 3.0 Line-editing Control Characters (Continued)

Character	Meaning
	mand line you can recall. If the command line contains characters, CTRL-W moves the cursor to the end of the command line. If you press RETURN, CP/M 3.0 executes the recalled command.
CTRL-X	Discards all the characters left of the cursor and moves the cursor to the beginning of the current line. CTRL-X saves any characters right of the cursor.

SECTION 14
Summary Of Major CP/M 3.0 Commands

THE TWO TYPES OF CP/M 3.0 COMMANDS	211
BUILT-IN COMMANDS	211
TRANSIENT UTILITY COMMANDS	212
REDIRECTING INPUT AND OUTPUT	214
ASSIGNING LOGICAL DEVICES	214
FINDING PROGRAM FILES	215
EXECUTING MULTIPLE COMMANDS	215
TERMINATING PROGRAMS	216
GETTING HELP	216

As noted in Section 11, a CP/M 3.0 command line consists of a command keyword, an optional command tail, and a RETURN keystroke. This section describes the two kinds of commands the command keyword can identify, and summarizes individual commands and their functions. The section also gives examples of some commonly used commands. In addition, the section explains the concept of logical and physical devices under CP/M 3.0. This section then tells how CP/M 3.0 searches for a program file on a disk, tells how to execute multiple commands, and how to reset the disk system. Finally, the section explains how to use the HELP command to get information on various CP/M topics including command formats and usage, right at the keyboard.

The Two Types of CP/M 3.0 Commands

There are two types of commands in CP/M 3.0:

- **Built-in commands**—which identify programs in memory
- **Transient utility commands**—which identify program files on a disk

CP/M 3.0 has six built-in commands and over 20 transient utility commands. You can add utilities to your system by purchasing various CP/M 3.0-compatible application programs. If you are an experienced programmer, you can also write your own utilities that operate with CP/M 3.0.

Built-In Commands

Built-in commands are parts of CP/M 3.0 that are always available for your use, regardless of which disk you have in which drive. Built-in commands are entered in the computer's memory when CP/M 3.0 is loaded, and therefore execute more quickly than the transient utilities. Table 14-1, on the next page, lists the Commodore 128 CP/M 3.0 built-in commands.

Some built-in commands have options that require support from a related transient utility. The related transient utiltiy command has the same name as the built-in command and has a filetype of COM.

Table 14-1. Built-in Commands

Command	Function
DIR	Displays filenames of all files in the directory except those marked with the SYS attribute.
DIRSYS	Displays filenames of files marked with the SYS (system) attribute in the directory.
ERASE	Erases a filename from the disk directory and releases the storage space occupied by the file.
RENAME	Renames a disk file.
TYPE	Displays contents of an ASCII (TEXT) file at your screen.
USER	Changes to a different user number.

Transient Utility Commands

The CP/M 3.0 transient utility commands are listed in Table 14-2. When you enter a command keyword that identifies a transient utility, CP/M 3.0 loads the program file from the disk and passes that file any filenames, data or parameters you entered in the command tail.

Table 14-2. Transient Utility Commands

Name	Function
COPYSYS	Creates a new boot disk.
DATE	Sets or displays the date and time.
DEVICE	Assigns logical CP/M devices to one or more physical devices, changes device driver protocol and baud rates, or sets console screen size.
DIR	Displays directory with files and their characteristics.
DUMP	Displays a file in ASCII and hexadecimal format.
ED	Creates and alters ASCII files.
ERASE	Used for wildcard erase.
GET	Temporarily gets console input from a disk file rather than the keyboard.
HELP	Displays information on how to use CP/M 3.0 commands.
INITDIR	Initializes a disk directory to allow time and date stamping.
PIP	Copies files and combines files.
PUT	Temporarily directs printer or console output to a disk file.
RENAME	Changes the name of a file, or a group of files using wildcard characters.
SET	Sets file options including disk labels, file attributes, type of time and date stamping and password protection.
SETDEF	Sets system options including the drive search chain.
SHOW	Displays disk and drive statistics.
SUBMIT	Automatically executes multiple commands.
TYPE	Display contents of text file (or group of files, if wildcard characters are used) on screen (and printer if desired).

Redirecting Input and Output

CP/M 3.0's PUT command allows you to direct console or printer output to a disk file. You can use a GET command to make CP/M 3.0 or a utility program take console input from a disk file. The following examples illustrate some of the capabilities offered by GET and PUT.

You can use a PUT command to direct console output to a disk file as well as to the console. With PUT, you can create a disk file containing a directory of all files on that disk, as shown in Figure 14-1.

```
A>PUT CONSOLE OUTPUT TO FILE DIR.PRN
PUTTING CONSOLE OUTPUT TO FILE: DIR.PRN

A>DIR
A: FILENAME   TEX : FRONT   TEX : FRONT   BAK : ONE    BAK : THREE   TEX
A: FOUR       TEX : ONE     TEX : LINEDIT TEX : EXAMP1 TXT : TWO     BAK
A: TWO        TEX : THREE   BAK : EXAMP2  TXT

A>TYPE DIR.PRN
A: FILENAME   TEX : FRONT   TEX : FRONT   BAK : ONE    BAK : THREE   TEX
A: FOUR       TEX : ONE     TEX : LINEDIT TEX : EXAMP1 TXT : TWO     BAK
A: TWO        TEX : THREE   BAK : EXAMP2  TXT
```

Figure 14-1. PUT Command Example

A GET command can direct CP/M 3.0 or a program to read console input from a disk file instead of from the keyboard. If the file is to be read by CP/M 3.0, it must contain standard CP/M 3.0 command lines. If the file is to be read by a utility program, it must contain input appropriate for that program. A file can contain both CP/M 3.0 command lines and program input if it also includes a command to start a program.

Assigning Logical Devices

The minimal Commodore 128 CP/M 3.0 hardware includes a console consisting of a keyboard and screen display, and a 1571 disk drive. You may want to add another device to your system, such as a printer or a modem. To help keep track of these physically different input and output devices, Table 14-3 gives the names of CP/M 3.0 logical devices. It also shows the physical devices assigned to these logical devices in the Commodore 128 CP/M 3.0 system.

Table 14-3. CP/M 3.0 Logical Devices

Logical Device Name	Device Type	Physical Device Assignment
CONIN:	Console input	Keyboard
CONOUT:	Console output	80-column Screen
AUXIN:	Auxiliary input	Null
AUXOUT:	Auxiliary output	Null
LST:	List output	PTR1 or PTR2

You can change these assignments with a DEVICE command. For example, you can, assign AUXIN and AUXOUT to a modem so that your computer can use telephone lines to communicate with other computer users, with information services like CompuServe, and with computerized bulletin boards.

Finding Program Files

If a command keyword identifies a utility, CP/M 3.0 looks for that program file on the default or specified drive. It looks under the current user number, and then under user 0 for the same file marked with the SYS attribute. At any point in the search process, CP/M 3.0 stops the search if it finds the program file. CP/M 3.0 then loads the program into memory and executes it. When the program terminates, CP/M 3.0 displays the system prompt and waits for your next command. However, if CP/M 3.0 does not find the command file, it repeats the command line followed by a question mark, and waits for your next command.

Executing Multiple Commands

In the examples so far, CP/M 3.0 has executed only one command at a time. CP/M 3.0 can also execute a sequence of commands. You can enter a sequence of commands at the system prompt, or you can put a frequently needed sequence of commands in a disk file, using a filetype of SUB. Once you have stored the sequence in a disk file, you can execute the sequence whenever you need to with a SUBMIT command.

Terminating Programs

You can use the two keystroke command CTRL-C to terminate program execution or reset the disk system. To enter a CTRL-C command, hold down the CTRL key and press C.

Most application programs that run under CP/M and most CP/M transient utilities can be terminated by a CTRL-C. However, if you try to terminate a program while it is sending a display to the screen, you may need to press a CTRL-S to halt the display before you enter CTRL-C.

Getting Help

CP/M 3.0 includes a transient utility command called HELP that will display a summary of the format and use for the most common CP/M commands. To access HELP, simply enter the command:

A)*HELP*

You can press the HELP key instead of typing the word HELP and pressing the RETURN key.

The list of available topics is then displayed, like this:

Topics available:

COMMANDS	**CNTRLCHARS**	**COPYSYS**	**DATE**	**DEVICE**	**DIR**
DUMP	**ED**	**ERASE**	**FILESPEC**	**GENCOM**	**GET**
HELP	**HEXCOM**	**INITDIR**	**LIB**	**LINK**	**MAC**
PATCH	**PIP (COPY)**	**PUT**	**RENAME**	**RMAC**	**SAVE**
SET	**SETDEF**	**SHOW**	**SID**	**SUBMIT**	**TYPE**
USER	**XREF**				

Suppose you type:

HELP) PIP

CP/M then displays the following information:

PIP (COPY)

Syntax:

DESTINATION SOURCE

PIP d: {Gn}filespec {[Gn]} = filespec {[o]} , . . .d: {[o]}

Explanation:

The file copy program PIP copies files, combines files, and transfers files between disks, printers, consoles, or other devices attached to your computer. The first filespec is the destination. The second filespec is the source. Use two or more source filespecs separated by commas to combine two or more files into one file. [o] is any combination of the available options. The [Gn] option in the destination filespec tells PIP to copy your file to that user number.

PIP with no command tail displays an * prompt and awaits your series of commands, entered and processed one line at a time. The source or destination can be any CP/M 3.0 logical device.

The HELP facility provides information like this on all the CP/M 3.0 built-in and transient utility commands. If you want information on a specific area, you can type ***HELP subject*** after the system prompt, where subject is a command tail describing the subject you are interested in. For example:

A〉 *HELP PIP*
At〉 *HELP DIRSYS*

You can refer to HELP any time you need information on a specific command. Or you can just browse through HELP to broaden your knowledge of CP/M 3.0.

SECTION 15
Commodore Enhancements To CP/M 3.0

KEYBOARD ENHANCEMENTS 221
- Defining a Key 222
- Defining a String 222
- Using ALT Mode 223

SCREEN ENHANCEMENTS 223

Commodore has added a number of enhancements to CP/M 3.0. These enhancements tailor the capabilities of the Commodore 128 to those of CP/M 3.0. This section describes these enhancements.

Keyboard Enhancements

Any key on the keyboard can be defined to generate a code or function, **except** the following keys:

Left SHIFT key
Right SHIFT key
Commodore key
CONTROL key
RESTORE key
40-80 key
CAPS LOCK key

In defining a key, the keyboard recognizes the following special functions. To indicate these functions, hold down the CONTROL key and the right SHIFT key, and press the desired function key simultaneously.

Key	*Function*
CURSOR LEFT key	**Defines key**
CURSOR RIGHT key	**Defines string (points to function keys)**
ALT key	**Toggles key filter**

Defining A Key

A user can define the code that a key can produce. Each key has four possible definitions: Normal, Alpha Shift, Shift and Control. The Alpha Shift is toggled on/off by pressing the Commodore key. After entering this mode, a small box will appear on the bottom of the screen. The first key that is pressed is the key to be defined. The current HEX (hexadecimal) value assigned to this key is displayed; the user can then type the new HEX code for the key, or abort by typing a non-HEX key. The following is a definition of the codes that can be assigned to a key. (In ALT mode, codes are returned to the application; see ALT Mode below.)

Code	*Function*
00h	**Null (same as not pressing a key)**
01h to 7Fh	**Normal ASCII codes**
80h to 9Fh	**String assigned**
A0h to AFh	**80-column character color**
B0h to BFh	**80-column background color**
C0h to CFh	**40-column character color**
D0h to DFh	**40-column background color**
E0h to EFh	**40-column border color**
F0h	**Toggle disk status on/off**
F1h	**System Pause**
F2h	**(Undefined)**
F3h	**40-column screen window right**
F4h	**40-column screen window left**
F5h to FFh	**(Undefined)**

Defining A String

This function allows the user to assign more than one key code to a single key. Any key that is typed in this mode is placed in the string. The user can see the results of typing in a long box at the bottom of the screen.

NOTE: Some keys may not display what they are. To provide the user with control over the process of entering data, the following five special key functions, are available. To access these functions, press the CONTROL and right SHIFT keys and the desired function keys.

Key	*Function*
RETURN	**Complete string definition**
+ (on main keyboard)	**Insert space into string**
– (on main keyboard)	**Delete cursor character**
Left arrow	**Cursor left**
Right arrow	**Cursor right**

Using ALT Mode

ALT mode is a toggle function (that is, it can be switched between ON and OFF.) The default value is OFF. This function allows the user to send 8-bit codes to an application.

Screen Enhancements

The default screen in CP/M 3.0 emulates an ADM31 terminal. The following screen functions emulate ADM 3A operation, which is a subset of ADM31 operation.

CTRL G	Sound bell
CTRL H	Cursor left
CTRL J	Cursor down
CTRL K	Cursor up
CTRL L	Cursor right
CTRL M	Move cursor to start of current line (CR)
CTRL Z	Home cursor and clear screen
ESC = RC	Cursor position where R is the row location (with values from space to 8) and C is the column location (next values from space to 0), referenced to the status line

Additional functions in ADM31 mode include:

ESC T ESC t	Clear to end of line
ESC Y ESC y	Clear to end of screen
ESC : ESC *	Home cursor and clear screen (including the status line)
ESC Q	Insert character
ESC W	Delete character
ESC E	Insert line
ESC R	Delete line

* ESC ESC ESC color# sets a screen color from a table of 16 color entries. (These are the same color values listed in Chapter II, Section 6, Figure 6-2.) The color # will be set as follows:

20h to 2Fh	character color
30h to 3Fh	background color
40h to 4Fh	border color (40 column only)

The visual effects associated with following functions are visible only with the 80-column screen format.

	ESC)	Half intensity
	ESC (	Full intensity
	ESC G4	Reverse video ON
*	ESC G3	Turn underline ON
	ESC G2	Blink ON
*	ESC G1	Select the alternate character set
	ESC G0	All ESC G attributes OFF

***NOTE:** This is NOT a normal ADM31 sequence.

****

The sections in this chapter provide a summary of the structure and wide-ranging capabilities of CP/M 3.0 For detailed information on any facet of CP/M 3.0, you should respond to the offer described on the card included in this chapter. In return you will receive a copy of the Digital Research, Inc. book, **CP/M Plus User's Guide.**

CHAPTER

5

BASIC 7.0 ENCYCLOPEDIA



SECTION 16
Introduction

ORGANIZATION OF ENCYCLOPEDIA 229

COMMAND AND STATEMENT FORMAT 229

Organization of Encyclopedia

This chapter lists BASIC 7.0 language elements. It gives a complete list of the rules (syntax) of Commodore 128 BASIC 7.0, along with a concise description of each.

BASIC 7.0 includes all the elements of BASIC 2.0. The new commands, statements, functions and operators provided in BASIC 7.0 are highlighted in color.

The different types of BASIC operations are listed in individual sections, as follows:

1. **COMMANDS and STATEMENTS:** the commands used to edit, store and erase programs; and the BASIC program statements used in the numbered lines of a program.
2. **FUNCTIONS:** the string, numeric and print functions.
3. **VARIABLES AND OPERATORS:** the different types of variables, legal variable names, arithmetic operators and logical operators.
4. **RESERVED WORDS AND SYMBOLS:** the words and symbols reserved for use by the BASIC 7.0 language, which cannot be used for any other purpose.

Command and Statement Format

The command and statement definitions in this encyclopedia are arranged in the following format:

Command name→	**AUTO**
Brief definition→	—Enable/disable automatic line numbering
Command format→	**AUTO [line#]**
Discussion of format and use→	This command turns on the automatic line-numbering feature. This eases the job of entering programs, by automatically typing the line numbers for the user. As each program line is entered by pressing RETURN, the next line number is printed on the screen, and the cursor is positioned two spaces to the right of the line number. The line number argument refers to the desired increment between line numbers. AUTO without an argument turns off the auto line numbering, as does RUN. This statement can be used only in direct mode (outside of a program).
Example(s)→	**EXAMPLES:** **AUTO 10** Automatically numbers program lines in increments of 10. **AUTO 50** Automatically numbers lines in increments of 50. **AUTO** Turns off automatic line numbering.

The boldface line that defines the format consists of the following elements:

```
      DLOAD "program name" [D0,U8]
      →                    →
keyword          argument    additional arguments
                             (possibly optional)
```

The parts of the command or statement that must be typed exactly as shown are in capital letters. Words the user supplies, such as the name of a program, are not capitalized.

When quote marks (" ") appear (usually around a program name or filename), the user should include them in the appropriate place, according to the format example.

KEYWORDS, also called reserved words, appear in upper-case letters. Keywords may be typed using the full word or the approved abbreviation. (A full list of abbreviations is given in Appendix K). The keyword or abbreviation must be entered correctly or an error will result. The BASIC and DOS error messages are defined in Appendices A and B, respectively.

Keywords are words that are part of the BASIC language. They are the central part of a command or statement, and they tell the computer what kind of action to take. These words cannot be used as variable names. A complete list of reserved words and symbols is given in Section 20.

ARGUMENTS, also called parameters, appear in lower-case letters. Arguments complement keywords by providing specific information to the command or statement. For example, the keyword load tells the computer to load a program while the argument tells the computer which specific program to load. A second argument specifies from which drive to load the program. Arguments include filenames, variables, line numbers, etc.

SQUARE BRACKETS [] show optional arguments. The user selects any or none of the arguments listed, depending on requirements.

ANGLE BRACKETS ⟨ ⟩ indicate the user MUST choose one of the arguments listed.

A VERTICAL BAR | separates items in a list of arguments when the choices are limited to those arguments listed. When the vertical bar appears in a list enclosed in SQUARE BRACKETS, the choices are limited to the items in the list, but the user still has the option not to use any arguments.

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

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

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

VARIABLE refers to any valid BASIC variable names, such as X, A$, T%, etc.

EXPRESSION refers to any valid BASIC expressions, such as A + B + 2, .5*(X + 3), etc.

SECTION 17
Basic Commands and Statements

APPEND

APPEND #logical file number,"filename"[,Ddrive number] [〈ON ,〉Udevice]

This command opens the file having the specified filename, and positions the pointer at the end of the file. Subsequent PRINT# (write) statements will cause data to be appended to the end of this logical file number. Default values for drive number and device number are 0 and 8 respectively.

Variables or expressions used as filenames must be enclosed within parentheses.

EXAMPLES: Append # 8, "MYFILE"	OPEN logical file 8 called "MYFILE" for appending with subsequent PRINT# statements.
Append # 7, (A$),D0,U9	OPEN logical file named by the variable in A$ on drive 0, device number 9, and prepare to APPEND.

AUTO

—Enable/disable automatic line numbering

AUTO [line#]

This command turns on the automatic line-numbering feature. This eases the job of entering programs, by automatically typing the line numbers for the user. As each program line is entered by pressing RETURN, the next line number is printed on the screen, and the cursor is positioned two spaces to the right of the line number. The line number argument refers to the desired increment between line numbers. AUTO without an argument turns off the auto line numbering, as does RUN. This statement can be used only in direct mode (outside of a program).

EXAMPLES:

AUTO 10 Automatically numbers program lines in increments of 10.

AUTO 50 Automatically numbers lines in increments of 50.

AUTO Turns off automatic line numbering.

BACKUP —Copy the entire contents from one disk to another on a dual disk drive

BACKUP source Ddrive number TO destination Ddrive number [⟨ON ,⟩Udevice]

This command copies all the files from the source diskette onto the destination diskette, using a dual disk drive. With the BACKUP command, a new diskette can be used without first formatting it. This is because the BACKUP command copies all the information on the diskette, including the format. Because of this, the BACKUP command destroys any information already on the destination disk. Therefore, when backing up onto a previously used diskette, make sure it contains no programs you mean to keep. As a precaution the computer asks "ARE YOU SURE?" before it starts the operation. Press the "Y" key to perform the BACKUP, or any other key to stop it. You should always create a backup of all your disks, in case the original diskette is lost or damaged. Also see the COPY command. The default device number is unit 8.

NOTE: This command can be used only with a dual-disk drive.

EXAMPLES:

BACKUP D0 to D1 Copies all files from the disk in drive 0 to the disk in drive 1, in dual disk drive unit 8.

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

BANK —Select one of the 16 banks, numbered 0–15

BANK bank number

This statement specifies the bank number and corresponding memory configuration for the Commodore 128 memory. The default bank is 15. Here is a table of available BANK configurations in the Commodore 128 memory:

BANK	CONFIGURATION
0	RAM(0) only
1	RAM(1) only
2	RAM(2) only
3	RAM(3) only
4	Internal ROM , RAM(0), I/O
5	Internal ROM , RAM(1), I/O
6	Internal ROM , RAM(2), I/O
7	Internal ROM , RAM(3), I/O
8	External ROM , RAM(0), I/O
9	External ROM , RAM(1), I/O
10	External ROM , RAM(2), I/O
11	External ROM , RAM(3), I/O
12	Kernal and Internal ROM (LOW), RAM(0), I/O
13	Kernal and External ROM (LOW), RAM(0), I/O
14	Kernal and BASIC ROM, RAM(0), Character ROM
15	Kernal and BASIC ROM, RAM(0), I/O

To look at a particular bank (in the machine language monitor, for example), type BANK n (n = 0–15) and enter the monitor. From within the monitor, precede the four-digit hexidecimal number of the address range you are viewing with a hexadecimal digit (0–F).

This procedure is described in detail in the **Commodore 128 Programmer's Reference Guide.**

BEGIN/BEND

A conditional statement like IF . . . THEN: ELSE structured so that you can include several program lines between the start (BEGIN) and end (BEND) of the structure. Here's the format:

IF Condition THEN BEGIN : statement

statement
statement BEND : ELSE BEGIN
statement
statement BEND

For EXAMPLE:

```
10 IF X=1 THEN BEGIN: PRINT "X=1 is True"
20 PRINT "So this part of the statement is performed"
30 PRINT "When X equals 1"
40 BEND: PRINT "End of BEGIN/BEND structure":GO to 60
50 PRINT "X does not equal 1":PRINT "The statements
between BEGIN/BEND are skipped"
60 PRINT "Rest of Program"
```

If the conditional (IF..THEN) statement in line 10 is true, the statements between the keywords BEGIN and BEND are performed, including all the statements on the same line as BEND. If the (IF...THEN) conditional statement in line 10 is FALSE, all statements between the BEGIN and BEND, including the ones on the same program line as BEND are skipped, and the program resumes with the first program line immediately following the line containing BEND. The BEGIN/BEND essentially treats lines 10 through 40 as one long line.

The same rules are true if the ELSE:BEGIN clause is specified. If the condition is true, all statements between ELSE:BEGIN and BEND are performed, including all statements on the same line as BEND. If false, the program resumes with the line immediately following the line containing BEND.

BLOAD

—Load a binary file starting at the specified memory location

BLOAD "filename"[,Ddrive number][,Udevice number] [,Bbank number][,Pstart address]

where:

- filename is the name of your file
- bank number lets you select one of the 16 banks
- start address is the memory location where loading begins

A binary file is a file, whether a program or data, that has been SAVEd either within the machine language monitor or by the BSAVE command. The BLOAD command loads the binary file into the location specified by the start address.

EXAMPLES:

BLOAD "SPRITES", B0, P3584 — LOADS the binary file "SPRITES" starting in location 3584 (in BANK 0).

BLOAD "DATA1", D0, U8, B1, P4096	LOADS the binary file "DATA 1" into location 4096 (BANK 1) from Drive 0, unit 8.

BOOT

—Load and execute a program which was saved as a binary file

BOOT "filename" [,Ddrive number][〈ON ,〉Udevice]

The command loads an executable binary file and begins execution at the predefined starting address. The default device number is 8 (drive 0).

EXAMPLE:

BOOT	BOOT an executable program (CP/M Plus for example).
BOOT "GRAPHICS 1", D0, U9	"GRAPHICS 1", D0, U9 BOOTS the program "GRAPHICS 1" from unit 9, drive 0, and executes it.

BOX

—Draw box at specified position on screen

BOX [color source], X1, Y1[,X2,Y2][,angle][,paint]]

where:

color source . . . . . 0 = Background color
1 = Foreground color
2 = Multicolor 1
3 = Multicolor 2

x1, y1 . . . . . Top left corner coordinate (scaled)

x2, y2 . . . . . Bottom right corner opposite x1, y1, (scaled); default is the PC location.

angle . . . . . Rotation in clockwise degrees; default is 0 degrees

paint . . . . . Paint shape with color
0 = Do not paint
1 = Paint
(default = 0)

This statement allows the user to draw a rectangle of any size on the screen. Rotation is based on the center of the rectangle. The pixel cursor (PC) is located at x2, y2 after the BOX statement is executed. The color source number must be a zero (0) or one (1) if in standard

bit map, or a 2 or 3 if in multicolor bit map mode. Also see the GRAPHIC command for selecting the appropriate graphic mode to be used with the BOX color source number.

The x and y values can place the pixel cursor at absolute coordinates such as (100,100) or at coordinates relative to previous position (+ / – x and + / – y) of the pixel cursor such as (+ 20, – 10). The coordinate of one axis (x or y) can be relative and the other can be absolute. Here are the possible combinations of ways to specify the x and y coordinates:

x,y	absolute x, absolute y
+/-x,y	relative x, absolute y
x, +/-y	absolute x, relative y
+/-x, +/-y	relative x, relative y

Also see the LOCATE command for information on the pixel cursor.

EXAMPLES:

BOX 1, +10, +10	Draw a box 10 pixels to the right and 10 down from the current pixel cursor location.
BOX 1, 10, 10, 60, 60	Draws the outline of a rectangle.
BOX , 10, 10, 60, 60, 45, 1	Draws a painted, rotated box (a diamond).
BOX , 30, 90, , 45, 1	Draws a filled, rotated polygon.

Any parameter can be omitted but you must include a comma in its place, as in the last two examples.

NOTE: Wrapping occurs if the degree is greater than 360.

BSAVE

—Save a binary file from the specified memory locations

BSAVE "filename"[,Ddrive number][,Udevice number] [,Bbank number],Pstart address TO Pend address

where:

- filename is the name you give the file
- drive number is either 0 or 1 on a dual drive (0 is the default for a single drive)
- device number is the number of disk drive unit (default is 8)
- bank number is the number of the bank you specify (0-15)
- start address is the starting address where the program is SAVEd from

- end address is the last address in memory which is SAVEd

This is the same as the SAVE command in the machine language MONITOR.

EXAMPLES:

BSAVE "SPRITE DATA",B0,P3584 TO P4096 Saves the binary file named "SPRITE DATA", starting at location 3584 through 4096 (BANK 0).

BSAVE "PROGRAM.SCR",D0,U9,B0,P3182 TO P8000 Saves the binary file named "PROGRAM.SCR" in the memory address range 3182 through 8000 (BANK 0) on drive 0, unit 9.

CATALOG

—Display the disk directory

CATALOG [Ddrive number][〈ON ,〉Udevice number][,wildcard string]

The CATALOG command displays the directory on the specified drive, just like the DIRECTORY command. See the DIRECTORY command.

EXAMPLE:

CATALOG Displays the disk directory on drive 0.

CHAR

—Display characters at the specified position on the screen

CHAR [color source],x,y[,string][,RVS]

This is primarily designed to display characters on a bit mapped screen, but it can also be used on a text screen. Here's what the parameters mean:

color source . . . . . .0 = Background
1 = Foreground

x.....Character column (0–79)
(wraps around to the next line in 40-column mode)

y.....Character row (0–24)

string.....String to print

reverse.....Reverse field flag (0 = off, 1 = on)

Text (alphanumeric strings) can be displayed on any screen at a given location by the CHAR statement. Character data is read from the Commodore 128 character ROM area. The user supplies the x and y coordinates of the starting position and the text string to be displayed. Color source and reverse imaging are optional.

In 40-column format the string is continued on the next line if it attempts to print past the (40th column) right edge of the screen. When used in TEXT mode, the string printed by the CHAR command works just like a PRINT string, including cursor and color control. These control functions inside the string do not work when the CHAR command is used to display text in bit map mode. Upper/lower case controls (CHR$ (14) or CHR$ (142)) also operate in bit map mode.

Multicolor characters are handled differently than standard characters. First select multicolor1 and multicolor 2 with the COLOR command. Set the GRAPHIC mode to multicolor. To display the foreground on multicolor 1, characters set the color source in the CHAR command to zero and the reverse flag to zero. To display the foreground on multicolor 2, set the color source to 0 and the reverse flag to 1. The following example displays the foreground character color using a red background. Change the reverse flag to 1 and the characters are displayed in multicolor 2 (blue).

```
10 Color 2,3: REM multicolor 1 = Red
20 Color 3,7: REM multicolor 2 = Blue
30 GRAPHIC 3,1
30 CHAR 0,10,10"TEXT",0
```

CIRCLE

—Draw circles, ellipses, arcs, etc. at specified positions on the screen

CIRCLE [color source],X,Y[,Xr][,Yr]
[,sa][,ea][,angle][,inc]

where:

color source . . . . . .0 = background color
1 = foreground color
2 = multicolor 1
3 = multicolor 2

x,y . . . . . .Center coordinate of the CIRCLE

xr . . . . . .X radius (scaled)

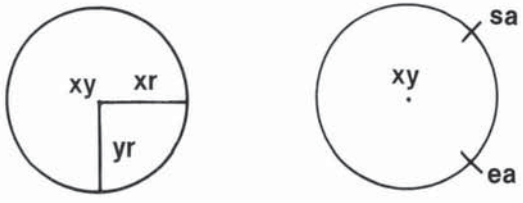
yr . . . . . .Y radius (default is xr)

sa. . . . . .Starting arc angle (default 0 degrees)

ea. . . . . .Ending arc angle (default 360 degrees)

angle . . . . . .Rotation is clockwise degrees (default is 0 degrees)

inc . . . . . .Degrees between segments (default is 2 degrees)



With the CIRCLE statement, the user can draw a circle, ellipse, arc, triangle, octagon or other polygon. The pixel cursor (PC) is left at the circumference of the circle at the ending arc angle. Any rotation is relative to the center. Setting the y radius equal to the x radius does not draw a perfect circle, since the x and y coordinates are scaled differently (x = 0 –320 and y = 0 –200). Arcs are drawn from the starting angle clockwise to the ending angle. The increment controls the smoothness of the shape; using lower values results in more nearly circular shapes. Specifying the inc greater than 2 creates a rough-edged, boxed-in shape.

The x and y values can place the pixel cursor at absolute coordinates such as (100,100) or at coordinates relative to the previous position (+ / – x and + / – y) of the pixel cursor such as (+ 20, – 10). The coordinate of one axis (x or y) can be relative and the other can be absolute. Here are the possible combinations of ways to specify the x and y coordinates:

x,y	absolute x, absolute y
+/−x,y	relative x, absolute y
x,+/−y	absolute x, relative y
+/−x,+/−y	relative x, relative y

Also see the LOCATE command for information on the pixel cursor.

EXAMPLES:

CIRCLE1, 160,100,65,10	Draws an ellipse.
CIRCLE1, 160,100,65,50	Draws a circle.
CIRCLE1, 60,40,20,18,,,,45	Draws an octagon.
CIRCLE1, 260,40,20,,,,,90	Draws a diamond.
CIRCLE1, 60,140,20,18,,,,120	Draws a triangle.
CIRCLE 1,+2,+2,50,50	Draws a circle (two pixels down and two to the right) relative to the original coordinates of the pixel cursor.

You may omit a parameter, but you must still place a comma in the appropriate position. Omitted parameters take on the default values.

CLOSE

—Close logical file

CLOSE file number

This statement closes any files used by the DOPEN or OPEN statements. The number following the word CLOSE is the file number to be closed.

EXAMPLE:

CLOSE 2	Logical file 2 is closed.

CLR

—Clear program variables

CLR

This statement erases any variables in memory, but leaves the program intact. This statement is automatically executed when a RUN or NEW command is given. There is no need to use CLR after editing, because variables and text no longer share memory.

CMD

—Redirect screen output

CMD logical file number [,write list]

This command sends the output, which normally goes to the screen (i.e., PRINT statement, LIST, but not POKES into the screen) to another device, such as a disk data file or printer. This device or file must be OPENed first. The CMD command must be followed by a number or numeric variable referring to the file. The write list can be any alphanumeric string or variable. This command is useful for printing headings at the top of program listings.

EXAMPLE:

Open 1,4	OPENS device #4, which is the printer.
CMD 1	All normal output now goes to the printer.
LIST	The LISTing goes to the printer, not the screen—even the word READY.
PRINT#1	Sends output back to the screen.
CLOSE 1	Closes the file.

COLLECT

—Free inaccessible disk space

COLLECT [Ddrive number][〈ON ,〉Udevice]

Use this command to make available any disk space that has been allocated to improperly closed (splat) files, and to delete references to these files from the directory. Splat files are files that appear on the directory with an asterisk next to them. Defaults to device number 8.

EXAMPLE:

COLLECT D0	Free all available space which has been incorrectly allocated to improperly closed files.

COLLISION

—Define handling for sprite collision interrupt

COLLISION type [,statement]

type. . . . . . Type of interrupt, as follows:
1 = Sprite-to-sprite collision
2 = Sprite-to-display data collision
3 = Light pen (40 columns only)

statement . . . . . BASIC line number of a subroutine

When the specified situation occurs, BASIC will finish processing the currently executing instruction and perform a GOSUB to the line number given. When the subroutine terminates (it must end with a RETURN), BASIC will resume processing where it left off. Interrupt action continues until a COLLISION of the same type without a line number is specified. More than one type of interrupt may be enabled at the same time, but only one interrupt can be handled at a time (i.e., there can be no recursion and no nesting of interrupts). The cause of an interrupt may continue causing interrupts for some time unless the situation is altered or the interrupt disabled.

When a sprite is completely off-screen and not visible, it cannot generate an interrupt. To determine which sprites have collided since the last check, use the BUMP function.

EXAMPLE:

Collision 0, 5000	Detects a sprite-to-sprite collision and program control sent to subroutine at line 5000.
Collision 0	Stops interupt action which was initiated in above example.
Collision 1, 1000	Detects sprite-to-data collision and program control directed to subroutine in line 1000.

COLOR

—Define colors for each screen area

COLOR source number, color number

This statement assigns a color to one of the seven color areas:

Area	Source
0	40-column (VIC) background
1	40-column (VIC) foreground
2	multicolor 1
3	multicolor 2
4	40-column (VIC) border
5	character color (40- or 80-column screen)
6	80-column background color

Colors that are usable are in the range 1-16.

Color Code	Color	Color Code	Color
1	Black	9	Orange
2	White	10	Brown
3	Red	11	Light Red
4	Cyan	12	Dark Gray
5	Purple	13	Medium Gray
6	Green	14	Light Green
7	Blue	15	Light Blue
8	Yellow	16	Light Gray

Color Numbers in 40-Column Format

1	Black	9	Dark Purple
2	White	10	Dark Yellow
3	Dark Red	11	Light Red
4	Light Cyan	12	Dark Cyan
5	Light Purple	13	Medium Gray
6	Dark Green	14	Light Green
7	Dark Blue	15	Light Blue
8	Light Yellow	16	Light Gray

Color Numbers in 80-Column Format

EXAMPLE:

Color 0, 1: Changes background color of 40 column screen to black.

Color 5, 8: Changes character color to yellow.

CONCAT

—Concatenate two data files

CONCAT "file 2" [,Ddrive number] TO "file 1" [,Ddrive number][〈ON ,〉Udevice]

The CONCAT command attaches file 2 to the end of file 1 and retains the name of file 1. The device number defaults to 8 and the drive number defaults to 0.

EXAMPLE:

Concat "File B" to "File A" FILE B is attached to FILE A, and the combined file is designated FILE A.

Concat (A$) to (B$), D1, U9 The file named by B$ becomes a new file with the same name with the file named by A$ attached to the end of B$—This is performed on Unit 9, drive 1 (a dual disk drive).

Whenever a variable is used as a filename, as in the last example, the filename variable must be within parentheses.

CONT

—Continue program execution

CONT

This command is used to restart a program that has been stopped by either using the STOP key, a STOP statement, or an END statement. The program resumes execution where it left off. CONT will not resume with the program if lines have been changed or added to the program or if any editing of the program is performed on the screen. If the program stopped due to an error; or if you have caused an error before trying to restart the program, CONT will not work. The error message in this case is CAN'T CONTINUE ERROR.

COPY

—Copy files from one drive to another in a dual disk drive, or within a single drive

COPY "source filename"[,Ddrive number]TO"destination filename"[,Ddrive number][〈ON ,〉Udevice]

This command copies files from one disk (the source file) to another (the destination file) on a dual-disk drive. It can also create a copy of a file on the same disk within a single drive, but the filename must be different. When copying from one drive to another, the filename may be the same.

The COPY command can also COPY all the files from one drive to another on a dual disk drive. In this case the drive numbers are specified and the source and destination filenames are omitted.

The default parameters for the COPY command are device number 8, drive 0.

NOTE: Copying between two single or double disk drive units cannot be done. See BACKUP.

EXAMPLES:

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

DATA —Define data to be used by a program

DATA list of constants

This statement is followed by a list of data items to be input into the computer's memory by READ statements. The items may be numeric or string and are separated by commas. String data need not be inside quote marks, unless they contain any of the following characters: space, colon, or comma. If two commas have nothing between them, the value is READ as a zero if numeric, or as an empty string. Also see the RESTORE statement, which allows the Commodore 128 to reread data.

EXAMPLE:

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

DCLEAR —Clear all open channels on disk drive

DCLEAR [Ddrive number][⟨ON ,⟩Udevice]

This statement closes all files and clears all open channels on the specified device number. Default is D0, U8. This command is analagous to OPEN 10,8,15, "I0" CLOSE 10.

EXAMPLES:

DCLEAR D0	Clears all open files on drive 0, device number 8.
DCLEAR D1,U9	Clears all open files (channels) on drive 1, device number 9.

DCLOSE

—Close disk file

DCLOSE [#logical file number][〈ON ,Udevice]

This statement closes a single file or all the files currently open on a disk unit. If no logical file number is specified, all currently open files are closed. The default device number is 8. Note the following examples:

EXAMPLES:

DCLOSE	Closes all files currently open on unit 8.
DCLOSE #2	Closes the file associated with the logical file number on unit 8.
DCLOSE ON U9	Closes all files currently open on unit 9.

DEF FN

—Return the value of a user-defined function

DEF FN name (variable) = expression

This statement allows definition of a complex calculation as a function. In the case of a long formula that is used several times within a program, this keyword can save valuable program space. The name given to the function begins with the letters FN, followed by any alphanumeric name beginning with a letter. First, define the function by using the statement DEF, followed by the name given to the function. Following the name is a set of parentheses () with a dummy numeric variable name (in this case, X) enclosed. Next is an equal sign, followed by the formula to be defined. The function can be performed by substituting any number for X, using the format shown in line 20 of the example below:

EXAMPLE:

```
10 DEF FNA(X) = 12*(34.75-X/.3) + X
20 PRINT FNA(7)
```

The number 7 is inserted each place X is located in the formula given in the DEF statement. In the example above, the answer returned is 144.

DELETE

—Delete lines of a BASIC program in the specified range

DELETE [first line] [-last line]

This command can be executed only in direct mode.

EXAMPLES:

DELETE 75	Deletes line 75.
DELETE 10-50	Deletes lines 10 through 50, inclusive.
DELETE-50	Deletes all lines from the beginning of the program up to and including line 50.
DELETE 75-	Deletes all lines from 75 to the end of the program, inclusive.

DIM

—Declare number of elements in an array

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

Before arrays of variables can be used, the program must first execute a DIM statement to establish DIMensions of the array (unless there are 11 or fewer elements in the array). The DIM statement is followed by the name of the array, which may be any legal variable name. Then, enclosed in parentheses, put the number (or numeric variable) of elements in each dimension. An array with more than one dimension is called a matrix. Any number of dimensions may be used, but keep in mind the whole list of variables being created takes up space in memory, and it is easy to run out of memory if too many are used. Here's how to calculate the amount of memory used by an array:

5 bytes for the array name
2 bytes for each dimension
2 bytes/elements for integer variables
5 bytes/elements for normal numeric variables
3 bytes/elements for string variables
1 byte for each character in each string element

Integer arrays take up two-fifths the space of floating-point arrays (e.g., DIM A% (100) requires 209 bytes; DIM A (100) requires 512 bytes.)

More than one array can be dimensioned in a DIM statement by separating the array variable name by commas. If the program executes a DIM statement for any array more than once, the message "RE'DIMed ARRAY ERROR" is posted. It is good programming practice to place DIM statements near the beginning of the program.

EXAMPLE:

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

41 elements, 16 elements, 64 elements

DIRECTORY

—Display the contents of the disk directory on the screen

DIRECTORY [Ddrive number][,〈ON ,〉 Udevice][,wildcard]

The F3 function key in C128 mode displays the DIRECTORY for device number 8, drive 0. Use CONTROL S or NO SCROLL to pause the display; any key restarts the display after a pause. Use the COMMODORE key to slow down the display. The DIRECTORY command cannot be used to print a hard copy. The disk directory must be loaded (LOAD"$",8) destroying the program currently in memory in order to print a hard copy. The default device number is 8, and the default drive number is 0.

EXAMPLES:

DIRECTORY	Lists all files on the disk in unit 8.
DIRECTORY D1, U9, "work"	Lists the file named "work," on drive 1 of unit 9.
DIRECTORY "AB*"	Lists all files starting with the letters "AB" like ABOVE, ABOARD, etc. on all drives of unit 8. The asterisk specifies a wild card, where all files starting with "AB" are displayed.
DIRECTORY D0, "file ?.BAK"	The ? is a wild card that matches any single character in that position. For example: file 1.BAK, file 2.BAK, file 3.BAK all match the string.
DIRECTORY D1,U9 (A$)	LISTS the filename stored in the variable A$ on device number 9, drive 1. Remember, whenever a variable is used as a filename, put the variable in parentheses.

NOTE: To print the DIRECTORY of the disk in drive 0, unit 8, use the following example:

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

DLOAD

—Load a BASIC program from disk

DLOAD "filename" [,Ddrive number][,Udevice number]

This command loads a BASIC program from disk into current memory. (Use LOAD to load programs from tape.) The program must be specified by a filename of up to 16 characters. DLOAD assumes device number 8, drive 0.

EXAMPLES:

DLOAD "BANKRECS"	Searches the disk for the program "BANKRECS" and LOADs it.
DLOAD (A$)	LOADS a program from disk whose name is stored in the variable A$. An error message is given if A$ is empty. Remember, when a variable is used as a filename, it must be enclosed in parentheses.

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

DO/LOOP/WHILE/UNTIL/EXIT

—Define and control program loop

DO [UNTIL condition / WHILE condition] statements [EXIT] LOOP [UNTIL condition / WHILE condition]

This loop structure performs the statements between the DO statement and the LOOP statement. If no UNTIL or WHILE modifies either the DO or the LOOP statement, execution of the statements in between continues indefinitely. If an EXIT statement is encountered in the body of a DO loop, execution is transferred to the first statement following the LOOP statement. DO loops may be nested, following the rules defined by the FOR-NEXT structure. If the UNTIL parameter is specified, the program continues looping until the condition is satisfied (becomes true). The WHILE parameter is basically the opposite of the UNTIL parameter: the program continues looping as long as the condition is TRUE. As soon as the condition is no longer true, program control resumes with the statement immediately following the LOOP statement. An example of a condition (boolean argument) is A = 1, or G>65.

EXAMPLE:

```
10 X=25
20 DO UNTIL X=0
30 X=X-1
40 PRINT "X=";X
50 LOOP
60 PRINT "End of Loop"
```

This example performs the statments X=X-1 and PRINT "X=";X until X=0. When X=0 the program resumes with the PRINT "End of Loop" statement immediately following LOOP.

```
10 DO WHILE A$=" ":GETKEY A$:LOOP
20 PRINT "The";A$;"key has been pressed"
```

A$ remains null as long as no key is pressed. As soon as a key is pressed, program control passes to the statement immediately following LOOP, Print "The";A$; "Key has been pressed". The example performs GETKEY A$ as long as A$ is a null character. This loop constantly checks to see if a key on the keyboard is being pressed.

```
10 DOPEN #8,"SEQFILE"
20 DO
30 GET #8,A$
40 PRINT A$;
50 LOOP UNTIL ST
60 DCLOSE #8
```

This program opens file "SEQFILE" and gets data until the ST system variable indicates all data is input.

DOPEN

—Open a disk file for a read and/or write operation

DOPEN #logical file number,"filename[,⟨S/P⟩]"[,Lrecord length][,Ddrive number][⟨ON ,⟩Udevice number][,w]

where:

S = Sequential File Type
P = Program File Type
L = Record Length = the length of records in a relative file only
W = Write Operation (if not specified a read operation occurs)

This statement opens a sequential, relative or random access file for a read or write operation. The record length (L) pertains to a relative file, which can be as long as 255. The "W" parameter is specified only during a write (PRINT#) operation in a sequential file. If it is not specified, the disk drive assumes the disk operation to be a read operation.

The logical file number associates a number to the file for future disk operatons such as a read (input#) or write (print#) operation. The logical file number can range from 1 to 255. Logical file numbers greater than 128 automatically send a carriage return and linefeed with each write (print#) command. Logical file numbers less than 128 send only a carriage RETURN, which can be suppressed with a semicolon at the end of the print# command. The default device number is 8, and the default drive is 0.

EXAMPLES:

DOPEN#1, "ADDRESS",W	Open the sequential file number 1 (ADDRESS) for a write operation
DOPEN"2 "RECIPES",D1,U9	Open the sequential file number 2 (RECIPES) for a read operation on device number 9, drive 1

DRAW

—Draw dots, lines and shapes at specified positions on screen

DRAW [color source], X1, Y1[TO X2, Y2] . . .

This statement draws individual dots, lines, and shapes. Here are the parameter values:

where:

Color source	0 Bit map background 1 Bit map foreground 2 Multicolor 1 3 Multicolor 2
X1,Y1	Starting coordinate (0,0 through 320,200)
X2,Y2	Ending coordinate (0,0 through 320,200)

The X and Y values can place the pixel cursor at absolute coordinates such as (100,100) or at coordinates relative to the previous position (+ / − x and + / − y) of the pixel cursor such as (+ 20, − 10). The coordinate of one axis (x or y) can be relative and the other can be absolute. Here are the possible combinations of ways to specify the x and y coordinates:

x,y	absolute x, absolute y
+/−x,y	relative x, absolute y
x, +/−y	absolute x, relative y
+/−x, +/−y	relative x, relative y

Also see the LOCATE command for information on the pixel cursor.

EXAMPLES:

DRAW 1, 100, 50 Draw a dot.

DRAW , 10,10 TO 100,60 Draw a line.

DRAW , 10,10 TO 10,60 TO 100,60 TO 10,10 Draw a triangle.

You may omit a parameter but you still must include the comma that would have followed the unspecified parameter. Omitted parameters take on the default values.

DSAVE

—Save a BASIC program file to disk

DSAVE "filename" [,Ddrive number][〈ON ,〉Udevice number]

This command stores (SAVEs) a BASIC program on disk. (See SAVE to store programs on tape.) A filename up to 16 characters long must be supplied. The default device number is 8, while the default drive number is 0.

EXAMPLES:

DSAVE "BANKRECS"	SAVEs the program "BANKRECS" to disk.
DSAVE (A$)	SAVEs the disk program named in the variable A$.
DSAVE "PROG 3",D1,U9	SAVEs the program "PROG 3" to disk on unit number 9, drive 1.

DVERIFY

—Verify the program in memory against the one on disk

DVERIFY "filename"[,Ddrive number][〈ON ,〉Udevice number]

This command causes the Commodore 128 to check the program on the specified drive against the program in memory. The default drive number is 0 and the default device number is 8.

NOTE: If a graphic area is allocated or reallocated after a SAVE, an error occurs. Technically this is correct. Because BASIC text is moved from its original (SAVEd) location when a bit mapped graphics area is allocated or deallocated, the original location where the C128 verified the SAVEd program changes. Hence, VERIFY, which performs byte-to-byte comparisons, fails, even though the program is valid.

To verify **Binary** data, see VERIFY "filename",8,1 format, under VERIFY command description.

EXAMPLES:

DVERIFY "C128"	Verifies program "C128" on drive 0, unit 8.
DVERIFY "SPRITES",D0,U9	Verifies program "SPRITES" on drive 0, device 9.

END

—Define the end of program execution

END

When the program encounters the END statement, it stops RUNning immediately. The CONT command can be used to restart the program at the next statement (if any) following the END statement.

ENVELOPE

—Define a musical instrument envelope

ENVELOPE n,[,atk] [,dec] [,sus] [,rel][,wf] [,pw]

where:

n. . . . . Envelope number (0-9)
atk . . . . . Attack rate (0-15)
dec. . . . . Decay rate (0-15)
sus. . . . . Sustain (0-15)
rel . . . . . Release rate (0-15)
wf. . . . . Waveform: 0 = triangle
1 = sawtooth
2 = variable pulse (square)
3 = noise
4 = ring modulation
pw . . . . . Pulse width (0-4095)

A parameter that is not specified will retain its predefined or currently redefined value. Pulse width applies to the width of the variable pulse waveform (wf = 2) only and is determined by the formula pwout = pw/40.95. The Commodore 128 has initialized the following 10 envelopes:

	n	A	D	S	R	wf	pw	instrument
ENVELOPE	0,	0,	9,	0,	0,	2,	1536	piano
ENVELOPE	1,	12,	0,	12,	0,	1		accordion
ENVELOPE	2,	0,	0,	15,	0,	0		calliope
ENVELOPE	3,	0,	5,	5,	0,	3		drum
ENVELOPE	4,	9,	4,	4,	0,	0		flute
ENVELOPE	5,	0,	9,	2,	1,	1		guitar
ENVELOPE	6,	0,	9,	0,	0,	2,	512	harpsichord
ENVELOPE	7,	0,	9,	9,	0,	2,	2048	organ
ENVELOPE	8,	8,	9,	4,	1,	2,	512	trumpet
ENVELOPE	9,	0,	9,	0,	0,	0		xylophone

To play predefined musical instrument envelopes,you can simply specify the envelope number and omit the rest of the parameters since they retain their predefined values.

FAST

—Put machine in 2 MHz mode of operation

FAST

This command initiates 2MHz mode, causing the VIC 40-column screen to be turned off. All operations (except I/O) are speeded up considerably. Graphics may be used, but will not be visible until a SLOW command is issued.

FETCH

—Get data from expansion (RAM module) memory

FETCH #bytes, intsa, expb, expsa

where bytes = number of bytes to get from expansion memory (1-65536)
intsa = starting address of host ram (0-65535)
expb = 64k expansion RAM bank number (0–3)
expsa = starting address of expansion RAM (0-65535)

FILTER

—Define sound (SID chip) filter parameters

FILTER [freq] [,lp] [,bp] [,hp] [,res]

where:

freq .....Filter cut-off frequency (0-2047)
lp .....Low-pass filter on (1), off (0)
bp.....Bank-pass filter on (1), off (0)
hp.....High-pass filter on (1), off (0)
res .....Resonance (0-15)

Unspecified parameters result in no change to the current value.

You can use more than one type of filter at a time. For example, both low-pass and high-pass filters can be used together to produce a notch-(or band-reject) filter response. For the filter to have an audible effect, at least one type of filter must be selected and at least one voice must be routed through the filter.

EXAMPLES:

FILTER 1024,0,1,0,2	Set the cutoff frequency at 1024, select the band pass filter and a resonance level of 2.
FILTER 2000,1,0,1,10	Set the cutoff frequency at 2000, select both the low pass and high pass filters (to form a notch reject) and set the resonance level at 10.

FOR/TO/STEP/ NEXT

—Define a repetitive program loop structure

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

This statement works with the NEXT statement to set up a section of the program (i.e., a loop) that repeats for a set number of times. This is useful when something needs to be counted or something must be done a certain number of times (such as printing).

This statement executes all the commands enclosed between the FOR and NEXT statements repetitively, according to the start and end values. The start value and the end value are the beginning and ending counts for the loop variable. The loop variable is added to or subtracted from during the FOR/NEXT loop.

The logic of the FOR/NEXT statement is as follows. First, the loop variable is set to the start value. When the program reaches a program line containing the NEXT statement, 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 value of the loop. If the loop variable is less than the end value, the loop is executed again, starting with the

statement immediately following the FOR statement. If the loop variable is greater than the end value, the loop terminates and the program resumes immediately following the NEXT statement. The opposite is true if the step size is negative. See also the NEXT statement.

EXAMPLE:

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

This program prints the numbers from one to 10 followed by the message I'M DONE! L = 11.

The end value of the loop may be followed by the word STEP and another number or variable. In this case, the value following the STEP is added each time instead of one. This allows counting backwards, by fractions, or in increments other than one.

The user can set up loops inside one another. These are known as nested loops. Care must be taken when nesting loops so the last loop to start is the first one to end.

EXAMPLE:

```
10 FOR L = 1 TO 100
20 FOR A = 5 TO 11 STEP .5
30 NEXT A
40 NEXT L
```

The FOR . . . NEXT loop in lines 20 and 30 are nested inside the one in line 10 and 40. Using a STEP increment of .5 is used to illustrate the fact that floating point indices are valid.

GET

—Receive input data from the keyboard, one character at a time, without waiting for a key to be pressed

GET variable list

The GET statement is a way to receive data from the keyboard, one character at a time. When GET is encountered in a program, the character that is typed is stored in the C128 memory. If no character is typed, a null (empty) character is returned, and the program continues without waiting for a key. There is no need to hit the RETURN key. The word GET is followed by a variable name, either numeric or string.

If the C128 intends to GET a numeric key and a key besides a number is pressed, the program stops and an error message is displayed. The GET statement may also be put into a loop, checking for an empty result. The GETKEY statement could also be used in this case. See GETKEY for more information. The GET and GETKEY statements can be executed only within a program.

EXAMPLE:

10 DO:GETA$:LOOP UNTIL A$ = "A"	This line waits for the A key to be pressed to continue.
20 GET B, C, D	GET numeric variables B,C and D from the keyboard without waiting for a key to be pressed.

GETKEY

—Receive input data from the keyboard, one character at a time and wait for a key to be pressed

GETKEY variable list

The GETKEY statement is very similar to the GET statement. Unlike the GET statement, GETKEY waits for the user to type a character on the keyboard. This lets the computer wait for a single character to be typed. This statement can be executed only within a program.

EXAMPLE:

10 GETKEY A$

This line waits for a key to be pressed. Typing any key continues the program.

10 GETKEY A$,B$,C$

This line waits for three alphanumeric characters to be entered from the keyboard.

GET#

—Receive input data from a tape, disk or RS232

GET# file number, variable list

This statement inputs one character at a time from a previously opened file. Otherwise, it works like the GET statement. This statement can be executed only within a program.

EXAMPLE:

```
10 GET#1,A$
```

This example receives one character, which is stored in the variable A$, from file number 1. This example assumes that file 1 was previously opened. See the OPEN statement.

GO64

—Switch to C64 mode

GO64

This statement switches from C128 mode to C64 mode. The question "Are You Sure?" is displayed in response to the GO64 statement. If Y is typed, then the currently loaded program is lost and control is given to C64 mode; otherwise, if any other key is pressed, the computer remains in C128 mode. This statement can be used in direct mode or within a program. The prompt is not displayed in program mode.

GOSUB

—Call a subroutine from the specified line number

GOSUB line number

This statement is similar to the GOTO statement, except the Commodore 128 returns from where it came when the subroutine is finished. When a line with a RETURN statement is encountered, the program jumps back to the statement immediately following the GOSUB statement.

The target of a GOSUB statement is called a subroutine. A subroutine is useful if a task is repeated several times within a program. Instead of duplicating the section of program over and over, set up a subroutine, and GOSUB to it at the appropriate time in the program. See also the RETURN statement.

EXAMPLE:

```
20 GOSUB 800
```

This example calls the subroutine beginning at line 800 and executes it. All subroutines must terminate with a RETURN statement.

```
:
:
800 PRINT "HI THERE": RETURN
```

GOTO/GO TO

—Transfer program execution to the specified line number

GOTO line number

After a GOTO statement is encountered in a program, the computer executes the statement specified by the line number in the GOTO statement. When used in direct mode, GOTO executes (RUNs) the program starting at the specified line number without clearing the variables. This is the same as the RUN command except it does not clear variable values.

EXAMPLES:

10 PRINT"COMMODORE" **20 GOTO 10**	The GOTO in line 20 makes line 10 repeat continuously until RUN/STOP is pressed.
GOTO 100	Starts (RUNs) the program starting at line 100, without clearing the variable storage area.

GRAPHIC

—Select a graphic mode

1) **GRAPHIC mode [,clear][,s] or**
2) **GRAPHIC CLR**

This statement puts the Commodore 128 in one of the six graphic modes:

mode	description
0	40-column text
1	standard bit-map graphics
2	standard bit-map graphics (split screen)
3	multicolor bit-map graphics
4	multicolor bit-map graphics (split screen)
5	80-column text

The clear parameter specifies whether the bit mapped screen is cleared (equal to 1) upon running the program, or left intact (equal to 0). The S parameter indicates the starting line number of the split screen when in graphic mode 2 or 4 (multicolor or standard bit map split screen modes). The default starting line number of the split screen is 19.

When executed, GRAPHIC 1-4 allocates a 9K-bit mapped area. The start of BASIC text area is moved above the bit-map area, and any BASIC program is automatically relocated. This area remains allo-

cated even if the user returns to TEXT mode (GRAPHIC 0). If the clear option is specified as 1, the screen is cleared. The GRAPHIC CLR command deallocates the 9k, bit-mapped area, places it in its original location below the bit-mapped area and makes it available once again for BASIC text .

EXAMPLES:

GRAPHIC 1,1	Select standard bit map mode and clear the bit map.
GRAPHIC 4,0,10	Select split screen multicolor bit map mode, do not clear the bit map and start the split screen at line 10.
GRAPHIC 0	Select 40-column text.
GRAPHIC 5	Select 80-column text.
GRAPHIC CLR	Clear and deallocate the bit map screen.

HEADER

—Format a diskette

HEADER "diskname" [,I i.d.] [,Ddrive number]
[⟨ON ,⟩Udevice number]

Before a new disk can be used for the first time, it must be formatted with the HEADER command. The HEADER command can also be used to erase a previously formatted disk, which can then be reused.

When you enter a HEADER command in direct mode, the prompt **ARE YOU SURE?** appears. In program mode, the prompt does not appear.

This command divides the disk into sections called blocks. It creates a table of contents of files, called a directory. The diskname can be any name up to 16 characters long. The i.d. number is any two alpha-numeric characters. Give each disk a unique i.d. number. Be careful when using the HEADER command because it erases all stored data.

You can HEADER a diskette quicker if it was already formatted, by omitting the new disk i.d. number. The old i.d. number is used. The quick header can be used only if the disk was previously formatted, since it clears out the directory rather than formatting the disk. The default device number is 8. The drive number must be specified (0 for a single disk drive).

As a precaution, the system asks "ARE YOU SURE?" before the Commodore 128 completes the operation. Press the "Y" key to perform the HEADER, or press any other key to cancel it.

The HEADER command reads the disk command error channel, and if an error is encountered, the error message "?BAD DISK ERROR" is displayed.

The HEADER command is analogous to the BASIC 2.0 command:

OPEN 1,8,15,"N0:diskname,i.d."

EXAMPLES:

HEADER "MYDISK", I23, D0	This HEADERS "MYDISK" using i.d I23 on drive 0, (default) device number 8.
HEADER "RECS", I45, D1, ON U9	This HEADERS "RECS" using i.d I45, on Drive 1, device number 9
HEADER "C128 PROGRAMS", D0	This is a quick header on drive 0, device number 8, assuming the disk in the drive was already formatted. The old i.d. is used.
HEADER (A$),I(B$),D0,U9	This example HEADERS the diskette with the name specified by the variable A$, and the i.d. specified by the variable B$, on drive 0, device number 9.

HELP —Highlight the line where the error occurred

HELP

The HELP command is used after an error has been reported in a program. When HELP is typed in 40-column format, the line where the error occurs is listed, with the portion containing the error displayed in reverse field. In 80-column format, the portion of the line where the error occurs is underlined.

IF/THEN/ELSE

—Evaluate a conditional expression and execute portions of a program depending on the outcome of the expression

IF expression THEN statements [:ELSE else-clause]

The IF. . . THEN statement evaluates a BASIC expression and takes one of two possible courses of action depending upon the outcome of the expression. If the expression is true, the statement(s) following THEN is executed. This can be any BASIC statement. If the expression is false, the program resumes with the program line immediately following the program line containing the IF statement, unless an ELSE clause is present. The entire IF. . . THEN statement must be contained within 160 characters. Also see BEGIN/BEND.

The ELSE clause, if present, must be on the same line as the IF. . . THEN portion of the statement, and separated from the THEN clause by a colon. When an ELSE clause is present, it is executed only when the expression is false. The expression being evaluated may be a variable or formula, in which case it is considered true if nonzero, and false if zero. In most cases, there is an expression involving relational operators (=, <, >, <=, >=, <>).

The IF. . . THEN statement can take two alternate forms:

IF expression THEN line number

or:

IF expression GOTO line number

These forms transfer program execution to the specified line number if the expression is true. Otherwise, the program resumes with the program line number immediately following the line containing the IF statement.

EXAMPLE:

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

This line checks the value of X. If X is greater than 0, the statement immediately following the keyword THEN (PRINT "OK") is executed and the ELSE clause is ignored. If X is less than or equal to 0, the ELSE clause is executed and the statement immediately following THEN is ignored.

```
10 IF X=10 THEN 100

20 PRINT "X does not equal 10"
:
99 STOP
100 PRINT "X equals 10"
```

This example evaluates the value of X. IF X equals 10, the program control is transferred to line 100 and the message "X EQUALS 10" is printed. IF X does not equal 10, the program resumes with line 20, the C128 prints the prompt "X does not equal 10" and the program stops.

INPUT

—Receive a data string or a number from the keyboard and wait for the user to press RETURN

INPUT ["prompt string";] variable list

The INPUT statement asks for data from the user while the program is RUNning and places the data into a variable or variables. The program stops, prints a question mark (?) on the screen, and waits for the user to type the answer and hit the RETURN key. The word INPUT is followed by a prompt string and a variable name or list of variable names separated by commas. The message in the prompt string inside quotes suggests (prompts) the information the user should enter. If this message is present, there must be a semicolon (;) after the closing quote of the prompt.

When more than one variable is INPUT, separate them by commas. The computer asks for the remaining values by printing two question marks (??). If the RETURN key is pressed without INPUTting a value, the INPUT variable retains the value previously input. The INPUT statement can be executed only within a program.

EXAMPLE:

```
10 INPUT "PLEASE TYPE A NUMBER";A
20 INPUT "AND YOUR NAME";A$
30 PRINT A$ "YOU TYPED THE NUMBER";A
```

INPUT#

—Inputs data from a file into the computer's memory

INPUT# file number, variable list

This statement works like INPUT, but takes the data from a previously OPENed file, usually on a disk or tape instead of the keyboard. No prompt string is used. This statement can be used only within a program.

EXAMPLE:

```
10 OPEN 2,8,2
20 INPUT#2, A$, C, D$
```

This statement INPUTs the data stored in variables A$, C and D$ from the disk file number 2, which was OPENed in line 10.

KEY

—Define or list function key assignments

KEY [key number, string]

There are eight function keys (F1-F8) available to the user on the Commodore 128: four unshifted and four shifted. The Commodore 128 allows you to perform a function or operation for each time the specified function key is pressed. The definition assigned to a key can consist of data, or a command or series of commands. KEY with no parameters specified returns a listing displaying all current KEY assignments. If data is assigned to a function key, that data is displayed on the screen when that function key is pressed. The maximum length for all the definitions together is 246 characters.

EXAMPLE:

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

This tells the computer to select the (VIC) text screen and list the program whenever the F7 key is pressed (in direct mode). CHR$(13) is the ASCII character for RETURN and performs the same action as pressing the RETURN key. Use CHR$(27) for ESCape. Use CHR$(34) to incorporate the double quote character into a KEY string. The keys may be redefined in a program. For example:

```
10 KEY 2,PRINT DS$ + CHR$(13)
```

This tells the computer to check and display the disk drive error channel variables (PRINT DS$) each time the F2 function key is pressed.

```
10 FOR I = 1 to 8:KEY I, CHR$(I + 132):NEXT
```

This defines the function keys as they are defined on the Commodore 64.

To restore all function keys to their BASIC default values, reset the Commodore 128 by pressing the RESET button.

LET

—Assigns a value to a variable

[LET] variable = expression

The word LET is rarely used in programs, since it is not necessary. Whenever a variable is defined or given a value, LET is always implied. The variable name that receives the result of a calculation is on the left side of the equal sign. The number, string or formula is on the right side. You can only assign one value with each (implied) LET statement. For example, LET A = B = 2 is illegal.

EXAMPLE:

10 LET A = 5	Assign the value 5 to numeric variable A.
20 B = 6	Assign the value 6 to numeric variable B.
30 C = A * B + 3	Assign the numeric variable C, the value resulting from 5 times 6 plus 3.
40 D$ = "HELLO"	Assign the string "HELLO" to string variable D$.

LIST

—List the BASIC program currently in memory

LIST [first line] [– last line]

The LIST command displays a BASIC program listing that has been typed or LOADed into the Commodore 128's memory so you can read and edit it. When LIST is used alone (without numbers following it), the Commodore 128 gives a complete LISTing of the program on the screen. The listing process may be slowed down by holding down the COMMODORE key, paused by CONTROL S or NO SCROLL KEY (and resumed by pressing any other key), or stopped by hitting the RUN/STOP key. If the word LIST is followed by a line number, the Commodore 128 shows only that line number. If LIST is typed with two numbers separated by a dash, all lines from the first to the second line number are displayed. If LIST is typed followed by a number and just a dash, the Commodore 128 shows all lines from that number to the end of the program. And if LIST is typed with a dash, then a number, all lines from the beginning of the program to that line number are LISTed. By using these variations, any portion of a program can be examined or brought to the screen for modification. In Commodore 128 mode, LIST can be used in a program and the program can resume with the CONT command.

EXAMPLES:

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

LOAD —Load a program from a peripheral device such as the disk drive or Datassette

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

This is the command used to recall a program stored on disk or cassette tape. Here, the filename is a program name up to 16 characters long, in quotes. The name must be followed by a comma (outside the quotes) and a number which acts as a device number to determine where the program is stored (disk or tape). If no number is supplied, the Commodore 128 assumes device number 1 (the Datassette tape recorder).

The relocate flag is a number (0 or 1) that determines where a program is loaded in memory. A relocate flag of 0 tells the Commodore 128 to load the program at the start of the BASIC program area. A flag of 1 tells the computer to LOAD from the point where it was SAVEd. The default value of the relocate flag is 0. The program parameter of 1 is generally used when loading machine language programs.

The device most commonly used with the LOAD command is the disk drive. This is device number 8, though the DLOAD command is more convient to use when working with disk.

If LOAD is typed with no arguments, followed by RETURN, the C128 assumes you are loading from tape and you are prompted to "PRESS PLAY ON TAPE". If you press PLAY, the Commodore 128 starts looking for a program on tape. When the program is found, the Commodore 128 prints FOUND"filename", where the filename is the name of the first file which the Datassette finds on the tape. Press the Commodore key to LOAD the found filename, or press the spacebar to keep searching on the tape. Once the program is LOADed, it can be RUN, LISTed or modified.

EXAMPLES:

LOAD	Reads in the next program from tape.
LOAD "HELLO"	Searches tape for a program called HELLO, and LOADs it if found.
LOAD (A$),8	LOADs the program from disk whose name is stored in the variable A$.
LOAD"HELLO",8	Looks for the program called HELLO on disk drive number 8, drive 0. (This is equivalent to DLOAD "HELLO").
LOAD"MACHLANG",8,1	LOADs the machine language program called "MACHLANG" into the location from which it was SAVEd.

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

LOCATE

—Position the bit map pixel cursor on the screen

LOCATE x, y

The LOCATE statement places the pixel cursor (PC) at any specified pixel coordinate on the screen.

The pixel cursor (PC) is the coordinate on the bit map screen where drawing of circles, boxes, lines and points and where PAINTing begins. The PC ranges from X and Y coordinates 0,0 through 320,200. The PC is not visible like the text cursor but it can be controlled through the graphics statements (BOX,CIRCLE,DRAW etc.) The default location of the pixel cursor is the coordinate specified as the X and Y portions in each particular graphics command. So the LOCATE command does not have to be specified.

The X and Y values can place the pixel cursor at absolute coordinates such as (100,100) or at coordinates relative to previous position (+ / − x and + / − y) of the pixel cursor such as (+ 20, − 10). The coordinate of one axis (x or y) can be relative and the other can be absolute. Here are the possible combinations of ways to specify the x and y coordinates:

x,y	absolute x, absolute y
+/−x,y	relative x, absolute y
x,+/−y	absolute x, relative y
+/−x,+/−y	relative x, relative y

EXAMPLE:

LOCATE 160,100	Positions the PC in the center of the bit map screen. Nothing will be seen until something is drawn.
LOCATE +20,100	Move the pixel cursor 20 pixels to the right of the last PC position and place it at Y coordinate 100.
LOCATE −30,+20	Move the PC 30 pixels to the left and 20 down from the previous PC position.

The PC can be found by using the RDOT(0) function to get the X-coordinate and RDOT(1) to get the Y-coordinate. The color source of the dot at the PC can be found by PRINTing RDOT(2).

MONITOR

—Enter the Commodore 128 machine language monitor

MONITOR

See Appendix J for details on the Commodore 128 Machine Language Monitor.

MOVSPR

—Position or move sprite on the screen

1) **MOVSPR number,x,y** Place the specified sprite at absolute coordinate x,y.

2) **MOVSPR number +/- x, +/- y**
Move sprite relative to the position of the pixel cursor.

3) **MOVSPR number,x;y** Move sprite distance x at angle y relative to the pixel cursor.

4) **MOVSPR number,x angle #y speed**
Move sprite at an angle (x) relative to its original coordinates, in the clockwise direction and at the specified speed (y).

where:

number is sprite's number (1 through 8)
〈,x1,y1〉 is coordinate of the sprite location.

ANGLE is the angle (0-360) of motion in the clockwise direction relative to the sprites original coordinate.

SPEED is a speed (0-15) in which the sprite moves.

This statement locates a sprite at a specific location on the screen according to the SPRITE coordinate plane (not the bit map plane) or initiates sprite motion at a specified rate. See MOVSPR in Section 6 for a diagram of the sprite coordinate system.

EXAMPLES:

MOVSPR 1,150,150	Position sprite 1 near the center of the screen, x,y coordinate 150,150.
MOVSPR 1, + 20, – 30	Move sprite 1 to the right 20 coordinates and up 30 coordinates.
MOVSPR 4, – 50, + 100	Move sprite 4 to the left 50 coordinates and down 100 coordinates.
MOVSPR 5, 45 #15	Move sprite 5 at a 45 degree angle in the clockwise direction, relative to its original x and y coordinates. The sprite moves at the fastest rate (15).

NOTE: Once you specify an angle and a speed in the third form of the MOVSPR statement, you must set the angle back to zero before moving other sprites, or their movement will be affected.

NEW —Clear (erase) program and variable storage

NEW

This command erases the entire program in memory and clears any variables that may have been used. Unless the program was stored on disk or tape, it is lost. Be careful with the use of this command. The NEW command also can be used as a statement in a BASIC program. However, when the Commodore 128 gets to this line, the program is erased and everything stops.

ON —Conditionally branch to a specified program line number according to the results of the specified expression

ON expression ⟨GOTO/GOSUB⟩ line #1 [, line #2, . . .]

This statement can make the GOTO and GOSUB statements operate like special versions of the (conditional) IF statement. The word ON is followed by a logical or mathematical expression, then either of the keywords GOTO or GOSUB and a list of line numbers separated by commas. If the result of the expression is 1, the first line in the list is executed. If the result is 2, the second line number is executed and so on. If the result is 0, or larger than the number of line numbers in

the list, the program resumes with the line immediately following the ON statement. If the number is negative, an ILLEGAL QUANTITY ERROR results.

EXAMPLE:

```
10 INPUT X:IF X<0 THEN 10
20 ON X GOTO 30, 40, 50, 60
25 STOP

30 PRINT "X = 1"
40 PRINT "X = 2"
50 PRINT "X = 3"
60 PRINT "X = 4"
```

When X = 1,ON sends control to the first line number in the list (30) When X = 2, ON sends control to the second line (40),etc

OPEN

—Open files for input or output

OPEN logical file number, device number [,secondary address] [,"filename, filetype, mode"]/[,cmd string]

The OPEN statement allows the Commodore 128 to access files within devices such as a disk drive, a Datassette cassette recorder, a printer or even the screen of the Commodore 128. The word OPEN is followed by a logical file number, which is the number to which all other BASIC input/output statements will refer, such as PRINT#(write), INPUT#(read), etc. This number is from 0 to 255.

The second number, called the device number, follows the logical file number. Device number 0 is the Commodore 128 keyboard; 1 is the cassette recorder; 3 is the Commodore 128 screen, 4-7 are the printer(s); and 8-11 are reserved for disk drives. It is often a good idea to use the same file number as the device number because it makes it easy to remember which is which.

Following the device number may be a third parameter called the secondary address. In the case of the cassette, this can be 0 for read, 1 for write and 2 for write with END-OF-TAPE marker at the end. In the case of the disk, the number refers to the channel number. See your disk drive manual for more information on channels and channel numbers. For the printer, the secondary addresses are used to select certain programming functions.

There may also be a filename specified for disk or tape OR a string following the secondary address, which could be a command to the disk/tape drive or the name of the file on tape or disk. If the filename

is specified, the type and mode refer to disk files only. File types are PROGRAM, SEQUENTIAL, RELATIVE and USER; modes are READ and WRITE.

EXAMPLES:

10 OPEN 3,3	OPENs the screen as file number 3.
20 OPEN 1,0	OPENs the keyboard as file number 1.
30 OPEN 1,1,0,"DOT"	OPENs the cassette for reading, as file number 1, using "DOT" as the filename.
OPEN 4,4	OPENs the printer as file number 4.
OPEN 15,8,15	OPENs the command channel on the disk as file 15, with secondary address 15. Secondary address 15 is reserved for the disk drive error channel.
5 OPEN 8,8,12,"TESTFILE,SEQ,WRITE"	OPENs a sequential disk file for writing called TESTFILE as file number 8, with secondary address 12.

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

PAINT

—Fill area with color

PAINT [color source],x,y[,mode]

where:

color source . . . . . .0 Bit map foreground
1 Bit map background (default)
2 Multicolor 1
3 Multicolor 2

x,y . . . . . .starting coordinate, scaled (default at pixel cursor (PC))

mode . . . . . .0 = paint an area defined by the color source selected
1 = paint an area defined by any non-background source

The PAINT command fills an area with color. It fills in the area around the specified point until a boundary of the same source is encountered. For example, if you draw a circle in the foreground color

source, start PAINTing the circle where the coordinate assumes the background source. The Commodore 128 will only PAINT where the specified source in the PAINT statement is different than the source of the x and y pixel coordinate. It cannot PAINT points where the sources are the same in the PAINT statement and the pixel coordinate. The x and y coordinate must lie completely within the boundary of the shape you intend to PAINT, and the source of the starting pixel coordinate and the specified color source must be different.

The x and y values can place the pixel cursor at absolute coordinates such as (100,100) or at coordinates relative to previous position (+ / – x and + / – y) of the pixel cursor such as (+ 20, – 10). The coordinate of one axis is (x or y) can be relative and the other can be absolute. Here are the posible combinations of ways to specify the x and y coordinates:

x,y	absolute x, absolute y
+/–x,y	relative x, absolute y
x,+/–y	absolute x, relative y
+/–x,+/–y	relative x, relative y

Also see the LOCATE command for information on the pixel cursor.

EXAMPLE:

10 CIRCLE 1, 160,100,65,50	Draws an outline of a circle.
20 PAINT 1, 160,100	Fills in the circle with color from source 1 (VIC foreground), assuming point 160,100 is colored in the background color (source 0).
10 BOX 1, 10, 10, 20, 20	Draws an outline of a box.
20 PAINT 1, 15, 15	Fills the box with color from source 1, assuming point 15,15 is colored in the background source (0).
30 PAINT 1, +10, +10	PAINT the screen in the foreground color source at the coordinate relative to the pixel cursor's previous position plus 10 in both the vertical and horizontal positions.

PLAY

—Defines and plays musical notes and elements

PLAY "Vn,On,Tn,Un,Xn,elements"

where:

Vn = Voice (n = 1-3)
On = Octave (n = 0-6)
Tn = Tune Envelope Defaults (n = 0-9)
0 = piano
1 = accordion
2 = calliope
3 = drum
4 = flute
5 = guitar
6 = harpsichord
7 = organ
8 = trumpet
9 = xylophone
Un = Volume (n = 0-15)
Xn = Filter on (n = 1), off (n = 0)
Notes: A,B,C,D,E,F,G
Elements:

#	Sharp
$	Flat
W	Whole note
H	Half note
Q	Quarter note
I	Eighth note
S	Sixteenth note
.	Dotted
R	Rest
M	Wait for all voices currently playing to end current measure

The PLAY statement gives you the power to select voice, octave and tune envelope (including ten predefined musical instrument envelopes), the volume and the notes you want to PLAY. All these controls are enclosed in quotes.

All elements except R and M precede the musical notes in a PLAY string.

EXAMPLES:

PLAY "V1O4T0U5X0CDEFGAB"	Play the notes C,D,E,F,G,A and B in voice 1, octave 4, tune envelope 0 (piano), at volume 5, with the filter off.

PLAY "V3O5T6U7X1#B$AW.CHDQEIF" Play the notes B-sharp, A-flat, a whole dotted-C note, a half D-note, a quarter E-note and an eighth F-note.

POKE

—Change the contents of a RAM memory location

POKE address, value

The POKE statement allows changing of any value in the Commodore 128 RAM, and allows modification of many of the Commodore 128 Input/Output registers. The keyword POKE is always followed by two parameters. The first is a location inside the Commodore 128 memory. This can be a value from 0 to 65535. The second parameter is a value from 0 to 255, which is placed in the location, replacing any value that was there previously. The value of the memory location determines the bit pattern of the memory location. The POKE occurs into the currently selected RAM bank. The POKE address depends on the BANK number. See BANK in this Encyclopedia for the appropriate BANK configurations.

EXAMPLE:

10 POKE 53280,1 Changes VIC border color

NOTE: PEEK, a function related to POKE, which returns the contents of the specified memory location, is listed under FUNCTIONS.

PRINT

—Output to the text screen

PRINT [print list]

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

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

The characters inside quotes are often called literals because they are printed literally, exactly as they appear. Variable names have the value they contain (either a number or a string) printed. Functions also have their number values printed.

Punctuation marks are used to help format the data neatly on the screen. The comma separates printed output by 10 spaces, while the semicolon separates printed output by three spaces. Either punctuation mark can be used as the last symbol in the statement. This results in the next PRINT statement acting as if it is continuing the previous PRINT statement.

EXAMPLES:	RESULTS
10 PRINT "HELLO"	HELLO
20 A$ = "THERE":PRINT "HELLO ";A$	HELLO THERE
30 A = 4:B = 2:?A + B	6
40 J = 41:PRINT J;PRINT J-1	41 40
50 PRINT A;B;D = A + B:PRINT D;A-B	4 2 6 2

See also POS, SPC, TAB and CHAR functions.

PRINT #

—Output data to files

PRINT# file number, print list

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

EXAMPLE:

10 OPEN 4,4 **20 PRINT#4,"HELLO THERE!",A$,B$**	Outputs the data "HELLO THERE" and the variables A$ and B$ to the printer.
10 OPEN 2,8,2 **20 PRINT#2,A,B$,C,D**	Outputs the data variables A, B$, C and D to the disk file number 2.

NOTE: The PRINT# command is used by itself to close the channel to the printer before closing the file, as follows:

```
10 OPEN 4,4
30 PRINT#4,"PRINT WORDS"
40 PRINT#4
50 CLOSE 4
```

—Output using format

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

This statement defines the format of string and numeric items for printing to the text screen, printer or other device. The format is put in quotes. This is the format list. Then add a semicolon and a list of what is to be printed in the format for the print list. The list can be variables or the actual values to be printed.

EXAMPLE:

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

When this is RUN, line 10 prints:

$13.25 $32.00 $***** Five asterisks (*****) are printed instead of a Y value because Y has five digits, and this condition does not conform to format list (as explained below)

Line 20 prints this:

CBM CAT Leaves three spaces before printing "CBM" as defined in format list.

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

The pound sign (#) reserves room for a single character in the output field. If the data item contains more characters than there are # signs in the format field, the entire field is filled with asterisks (*); no characters are printed.

EXAMPLE:

```
10 PRINT USING "####";X
```

For these values of X, this format displays:

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

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

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

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

If neither a plus nor a minus sign is used in the format field for a numeric data item, a minus sign is printed before the first digit or dollar symbol if the number is negative. No sign is printed if the number is positive. This means that one additional character, the minus sign, is printed if the number is negative. If there are too many characters to fit into the field specified by the pound sign and plus/minus signs, then an overflow occurs and the field is filled with asterisks (*).

A decimal point (.) symbol designates the position of the decimal point in the number. There can be only one decimal point in any format field. If a decimal point is not specified in the format field, the value is rounded to the nearest integer and printed without decimal places.

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

A comma (,) allows placing of commas in numeric fields. The position of the comma in the format list indicates where the commas appear in a printed number. Only commas within a number are

printed. Unused commas to the left of the first digit appear as filler character. At least one pound sign must precede the first comma in a field.

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

EXAMPLES:

FIELD	EXPRESSION	RESULT	COMMENT
##.#	−.1	−0.1	Leading zero added.
##.#	1	1.0	Trailing zero added.
####	−100.5	−101	Rounded to no decimal places.
####	−1000	****	Overflow because four digits and a minus sign cannot fit in field.
###.	10	10.	Decimal point added.
#$##	1	$1	Leading dollar sign.

A dollar sign ($) symbol shows that a dollar sign will be printed in the number. If the dollar sign is to float (always be placed before the number), at least one pound sign must be specified before the dollar sign. If a dollar sign is specified without a leading pound sign, the dollar sign is printed in the position shown in the format field. If commas and/or a plus or minus sign are specified in a format field with a dollar sign, the program prints a comma or sign before the dollar sign. The up arrows or caret symbols (^^) are used to specify that the number is to be printed in E + format (scientific notation). A pound sign must be used in addition to the four carets to specify the field width. The carets can appear either before or after the pound sign in the format field. Four carets must be specified when a number is to be printed in E format. If more than one but fewer than four carets are specified, a syntax error results. If more than four carets are specified, only the first four are used. The fifth caret is interpreted as a no-text symbol. An equal sign (=) is used to center a string in a field. The field width is specified by the number of characters (pound sign and equal sign) in the format field. If the string contains fewer characters than the field width, the string is centered in the field. If the string contains more characters that can be fit into the field, then the right-most characters are truncated and the string fills the entire field. A greater than sign (>) is used to right justify a string in a field.

PUDEF

—Redefine symbols in PRINT USING statement

PUDEF "nnnn"

Where "nnnn" is any combination of characters, up to four in all, PUDEF allows you to redefine any of the following four symbols in the PRINT USING statement: blanks, commas, decimal points and dollar signs. These four symbols can be changed into some other character by placing the new character in the correct position in the PUDEF control string.

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

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

Position 3 is the decimal point. Default is a decimal point.

Position 4 is the dollar sign. Default is a dollar sign.

EXAMPLES:

10 PUDEF "*"	PRINTs * in the place of blanks.
20 PUDEF " ⟨"	PRINTs ⟨ in the place of commas.

READ

—Read data from DATA statements and input it into the computer's memory (while the program is RUNning)

READ variable list

This statement inputs information from DATA statements and stores it in variables, where the data can be used by the RUNning program. The READ statement variable list may contain both strings and numbers. Be careful to avoid reading strings where the READ statement expects a number and vice versa. This produces a TYPE MISMATCH ERROR message.

The data in the DATA statements are READ in sequential order. Each READ statement can read one or more data items. Every variable in the READ statement requires a data item. If one is not supplied, an OUT OF DATA ERROR occurs.

In a program, you can READ the data and then re-read it by issuing the RESTORE statement. The RESTORE sets the sequential data pointer back to the beginning, where the data can be read again. See the RESTORE statement.

EXAMPLES:

10 READ A, B, C **20 DATA 3, 4, 5**	READ the first three numeric variables from the closest data statement.
10 READ A$, B$, C$ **20 DATA JOHN, PAUL, GEORGE**	READ the first three string string variables from the nearest data statement.
10 READ A, B$, C **20 DATA 1200, NANCY, 345**	READ (and input into the C128 memory) a numeric variable, a string variable and another numeric variable.

RECORD

—Position relative file pointers

RECORD# logical file number, record number [,byte number]

This statement positions a relative file pointer to select any byte (character) of any record in the relative file. The logical file number can be in the range between 0 and 255. The record number can be in the range 0 through 65535. Byte number is in the range 1 through 254. See your disk drive manual for details about relative files.

When the record number value is set higher than the last record number in the file, the following occurs:

For a write (PRINT#) operation, additional records are created to expand the file to the desired record number.

For a read (INPUT#) operation, a null record is returned and a "RECORD NOT PRESENT ERROR occurs".

EXAMPLES:

```
10 OPEN 2,8,2"CUSTOMER,R,W"
20 RECORD#2,10,1
30 PRINT#2,A$
40 CLOSE 2
```

This example opens an existing relative file called "CUSTOMER" as file number 2 in line 10. Line 20 positions the relative file pointer at the first byte in record number 10. Line 30 actually writes the data, A$, to file number 2.

The RECORD command accepts variables for its parameters. It is often convienent to place a RECORD command within a FOR . . . NEXT or DO loop. Also see DOPEN and OPEN.

REM —Comments or remarks about the operation of a program line

REM message

The REMark statement is a note to whoever is reading a listing of the program. REM may explain a section of the program, give information about the author, etc. REM statements do not affect the operation of the program, except to add length to it (and therefore use more memory). Nothing to the right of the keyword REM is interpreted by the computer as an executable instruction. Therefore, no other executable statement can follow a REM on the same line.

EXAMPLE:

10 NEXT X:REM This line increments X.

RENAME —Change the name of a file on disk

RENAME "old filename" TO "new filename" [,Ddrive number] [,Udevice number]

This command is used to rename a file on a disk, from the old filename to the new filename. The disk drive does not RENAME a file if it is OPEN.

EXAMPLES:

RENAME "TEST" TO "FINALTEST",D0	Change the name of the file "TEST" to "FINAL TEST".
RENAME (A$) to (B$),D0,U9	Change the filename specified in A$ to the filename specified in B$ on drive 0, device number 9. Remember, whenever a variable name is used as a filename, it must be enclosed in parentheses.

RENUMBER

—Renumber lines of a BASIC program

RENUMBER [new starting line number][,increment] [,old starting line number]

The new starting line is the number of the first line in the program after renumbering; the default value is 10. The increment is the interval between line numbers, (i.e., 10, 20, 30, etc.); the increment default value is also 10. The old starting line number is the first line number before you renumber the program. This allows renumbering of a select portion of the program. The default in this case is the first line of the program. This command can only be executed from direct mode.

A "LINE NUMBER NOT FOUND ERROR" occurs if any reference to line number that doesn't exist is encountered. An "OUT OF MEMORY" occurs if RENUMBERing expands the program beyond its limits. Either error leaves the program unharmed.

EXAMPLES:

RENUMBER	Renumbers the program starting at 10, and increments each additional line by 10.
RENUMBER 20, 20, 1	Starting at line 1, renumbers the program. Line 1 becomes line 20, and other lines are numbered in increments of 20.
RENUMBER, , 65	Starting at line 65, renumbers in increments of 10. Line 65 becomes line 10. If you omit a parameter, you must still enter a comma as a placeholder.

RESTORE

—Reset READ pointer so the DATA can be reREAD

RESTORE [line #]

When executed in a program, the pointer to the item in a DATA statement that is to be read next is reset to the first item in the DATA statement. This provides the capability to reREAD the data. If a line number follows the RESTORE statement, the READ pointer is set to the first data item in the specified program line. Otherwise the pointer is reset to the beginning of the BASIC program.

EXAMPLES:

```
10 FOR I = 1 TO 3
20 READ X
30 TOTAL = X + TOTAL
40 NEXT
50 RESTORE
60 GOTO 10
70 DATA 10,20,30
```

This example READs the data in line 70 and stores it in numeric variable X. It adds the total of all the numeric data items. Once all the data has been READ, three cycles through the loop, the READ pointer is RESTOREd to the beginning of the program and it returns to line 10 and performs repetitively.

```
10 READ A,B,C
20 DATA 100,500,750
30 READ X,Y,Z
40 DATA 36,24,38
50 RESTORE 40
60 READ S,P,Q
```

This example RESTORES the DATA pointer to the beginning data item in line 40. When line 60 is executed, it will READ the DATA 36,24,38 from line 40, since you don't need to READ line 20's DATA again.

RESUME

—Define where the program will continue (RESUME) after an error has been trapped

RESUME [line # / NEXT]

This statement is used to restart program execution after TRAPping an error. With no parameters, RESUME attempts to re-execute the line in which the error occurred. RESUME NEXT resumes execution at the statement immediately following the one containing the error; RESUME followed by a line number will GOTO the specific line and resume execution from that line number. RESUME can only be used in program mode.

EXAMPLE:

```
10 INPUT "■ENTER A NUMBER";A
15 TRAP 100: B = 100/A
40 PRINT"THE RESULT = ";B: PRINT"THE END"
50 INPUT "DO YOU WANT TO RUN IT AGAIN (Y/N)";Z$:IF
Z$ = "Y" THEN 10
60 STOP
100 INPUT"ENTER ANOTHER NUMBER (NOT ZERO)";A
110 RESUME 15
```

This example traps a "division by zero error" in line 15 if 0 is entered in line 10. If zero is entered, the program goes to line 100, where you are asked to input another number besides 0. Line 110 returns to line

15 to complete the calculation. Line 50 asks if you want to repeat the program again. If you do, press the Y key.

RETURN —Return from subroutine

RETURN

This statement is always paired with the GOSUB statement. When the program encounters a RETURN statement, it goes to the statement immediately following the last GOSUB command executed. If no GOSUB was previously issued, then a RETURN WITHOUT GOSUB ERROR message is displayed and the program stops. All subroutines end with a RETURN statement.

EXAMPLE:

```
10 PRINT "ENTER SUBROUTINE"
20 GOSUB 100
30 PRINT "END OF SUBROUTINE"
.
.
.
90 STOP
100 PRINT "SUBROUTINE 1"
110 RETURN
```

This example calls the subroutine at line 100 which prints the message "SUBROUTINE 1" and RETURNs to line 30, the rest of the program.

RUN —Execute BASIC program

1) **RUN [line #]**
2) **RUN "filename" [,Ddrive number][,Udevice number]**

Once a program has been typed into memory or LOADed, the RUN command executes it. RUN clears all variables in the program before starting program execution. If there is a number following the RUN command, execution starts at that line number. If there is a filename following the RUN command, the named file is loaded from the disk drive and RUN, with no further action required of the user. RUN may be used within a program. The default drive number is 0 and default device number is 8.

EXAMPLES:

RUN	Starts execution from the beginning of the program.

RUN 100	Starts program execution at line 100.
RUN"PRG1"	DLOADS "PRG1" from disk drive 8, and runs it from the starting line number.
RUN(A$)	DLOADs the program named in the variable A$.

SAVE

—Store the program in memory to disk or tape

SAVE ["filename"][,device number][,EOT flag]

This command stores a program currently in memory onto a cassette tape or disk. If the word SAVE is typed alone followed by RETURN, the Commodore 128 assumes that the program is to be stored on cassette tape. It has no way of checking if there is already a program on the tape in that location, so make sure you do not record over valuable information on your tape. If SAVE is followed by a filename in quotes or a string variable name, the Commodore 128 gives the program that name, so it may be located easily and retrieved in the future. If a device number is specified for the SAVE, follow the name with a comma (after the quotes) and a number or numeric variable. Device number 1 is the tape drive, and number 8 is the disk drive. After the device number on a tape command, there can be a comma and a second number, which is either 0 or 1. If the second number is 1, the Commodore 128 puts an END-OF-TAPE marker (EOT flag) after the program (tape output only). If, in trying to LOAD a program, the Commodore 128 finds one of these markers, rather than the program to be LOADed, a FILE NOT FOUND ERROR is reported.

EXAMPLES:

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

SCALE

—Alter scaling in graphics mode

SCALE n [,xmax,ymax]

where:

n = 1 (on) or 0 (off)

In Standard bit map mode	320 < = X max < 32767 (default = 1023) 200 < = Y max < 32767 (default = 1023)
In Multicolor mode	160 < = X max < 32767 (default = 511) 160 < = Y max < 32767 (default = 511)

This statement changes the scaling of the bit maps in multicolor and high-resolution modes. Entering:

SCALE 1

turns scaling on. Coordinates may then be scaled from 0 to 32767 in both X and Y, rather than the normal scale values, which are:

multicolor mode .....X = 0 to 159 Y = 0 to 199
bit map mode .....X = 0 to 319 Y = 0 to 199

EXAMPLES:

```
10 GRAPHIC 1,1
20 SCALE 1:CIRCLE 1,180,100,100,100
```

Enter standard bit map, turn scaling on to default size (1023,1023) and draw a circle.

```
10 GRAPHIC 1,3
20 SCALE 1,1000,5000
30 CIRCLE 1,180,100,100,100
```

Enter multicolor mode, turn scaling on to size (1000,5000) and draw a circle.

SCNCLR

—Clear screen

SCNCLR mode number

The modes are as follows:

Mode Number	Mode
0	40 column (VIC) text
1	bit map
2	split screen bit map
3	multicolor bit map

4 split screen multicolor bit map
5 80 column (8563) text

This statement with no argument clears the graphic screen, if it is present otherwise the current text screen is cleared.

EXAMPLES:

SCNCLR 5 Clears 80 column text screen.
SCNCLR 1 Clears the (VIC) bit map screen.
SCNCLR 4 Clears the (VIC) split screen multicolor bit map.

SCRATCH

—Delete a file from the disk directory

SCRATCH "filename" [,Ddrive number][,Udevice number]

This command deletes a file from the disk directory. As a precaution, the system asks "ARE YOU SURE?" (in direct mode only) before the Commodore 128 completes the operation. Type a Y to perform the SCRATCH or press any other key to cancel the operation. Use this command to erase unwanted files, and to create more space on the disk. The filename may contain template, or wildcards (?,* etc.). The default drive number is 0 and default device number is 8.

EXAMPLE:

SCRATCH "MY BACK", D0

This erases the file MY BACK from the disk in drive 0.

SLEEP

—Delay program for a specific period of time

SLEEP N

where N is seconds 0< N < 65535

SLOW

—Return the Commodore 128 to 1Mhz operation

SLOW

The Commodore 128 is capable of running the 8502 microprocessor at a speed of 1 or 2 megahertz (Mhz).

The SLOW command slows down the microprocessor to 1 Megahertz from 2 Megahertz. The FAST command sets the Commodore at 2 Mhz. The Commodore 128 can process and access disk substantially faster at 2 Mhz than operating at 1 Mhz.

SOUND

—Output sound effects and musical notes

SOUND v,f,d[,dir][,m][,s][,w][,p]

where : **v** = voice (1..3)
f = frequency value (0..65535)
d = duration (0..32767)
dir = step direction (0(up) ,1(down) or 2(oscillate)) default = 0
m = minimum frequency (if sweep is used) (0..65535) default = 0
s = step value for sweep (0..32767) default = 0
w = waveform (0 = triangle,1 = sawtooth,2 = variable, 3 = noise) default = 2
p = pulse width (0..4095) default = 2048

The SOUND command is a fast and easy way to create sound effects and musical tones. The three required parameters v,f and d select the voice, frequency and duration of the sound. The duration is in units called jiffies. Sixty jiffies equals 1 second.

The SOUND command can sweep through a series of frequencies which allows sound effects to pass through a range of notes. Specify the direction of the sweep with the DIR parameter. Set the minimum frequency of the sweep with M and the step value of the sweep with S. Select the appropriate waveform with W and specify P as the width of the variable pulse waveform if selected in W.

EXAMPLES:

SOUND 1,40960,60	Play a SOUND at frequency 40960 in voice 1 for 1 second.
SOUND 2,20000,50,0,2000,100	Output a sound by sweeping through frequencies starting at 2000 and incrementing upward in units of 100. Each frequency is played for 50 jiffies.
SOUND 3,5000,90,2,3000,500,1	This example outputs a range of sounds starting at a minimum frequency of 3000, through 5000, in increments of 500. The direction of the sweep is back and forth (oscillating). The selected waveform is sawtooth and the voice selected is 3.

SPRCOLOR

—Set multicolor 1 and/or multicolor 2 colors for all sprites

SPRCOLOR [smcr-1][,smcr-2]

where:

smcr-1 Sets multicolor 1 for all sprites.
smcr-2 Sets multicolor 2 for all sprites.

Either of these parameters may be any color from 1 through 16.

EXAMPLES:

SPRCOLOR 3,7 Sets sprite multicolor 1 to red and multicolor 2 to blue.
SPRCOLOR 1,2 Sets sprite multicolor 1 to black and multicolor 2 to white.

SPRDEF

—Enter the SPRite DEFinition mode to create and edit sprite images.

SPRDEF

The SPRDEF command defines sprites interactively.

Entering the SPRDEF command, displays a sprite work area on the screen which is 24 characters wide by 21 characters tall. Each character position in the grid corresponds to a sprite pixel in the sprite displayed to the right of the work area. Here is a summary of the SPRite DEFinition mode operations and the keys that perform them:

User Input	Description
1-8	Selects a sprite number.
A	Turns on and off automatic cursor movement.
CRSR keys	Moves cursor.
RETURN key	Moves cursor to start of next line.
RETURN key	Exits sprite designer mode at the SPRITE NUMBER? prompt only.
HOME key	Moves cursor to top left corner of sprite work area.
CLR key	Erases entire grid.
1-4	Selects color source.
CTRL key, 1-8	Selects sprite foreground color (1-8).
Commodore key, 1-8	Selects sprite foreground color (9-16).
STOP key	Cancels changes and returns to prompt.
SHIFT RETURN	Saves sprite and returns to SPRITE NUMBER? prompt.

X	Expands sprite in X (horizontal) direction.
Y	Expands sprite in Y (vertical) direction.
M	Multicolor sprite.
C	Copies sprite data from one sprite to another.

SPRITE

—Turn on and off, color, expand and set screen priorities for a sprite

SPRITE ⟨number⟩ [,on/off][,fgnd][,priority][,x-exp] [,y-exp][,mode]

The SPRITE statement controls most of the characteristics of a sprite.

Parameter	Description
number	Sprite number (1-8)
on/off	Turn sprite on (1) or off (0)
foreground	Sprite foreground color (1-16)
priority	Priority is 0 if sprites appear in front of objects on the screen. Priority is 1 if sprites appear in back of objects on the screen.
x-exp	Horizontal EXPansion on (1) or off (0)
y-exp	Vertical EXPansion on (1) or off (0)
mode	Select standard sprite (0) or multicolor sprite (1)

Unspecified parameters in subsequent sprite statements take on the characters of the previous SPRITE statement. You may check the characteristics of a SPRITE with the RSPRITE function.

EXAMPLES:

SPRITE 1,1,3	Turn on sprite number 1 and color it red.
SPRITE 2,1,7,1,1,1	Turn on sprite number 2, color it blue make it pass behind objects on the screen and expand it in the vertical and horizontal directions.
SPRITE 6,1,1,0,0,1,1	Turn on SPRITE number 6, color it black. The first 0 tells the computer to display the sprites in front of objects on the screen. The second 0 and the 1 following tell the C128 to expand the sprite verticaly only. The last 1 specifies multicolor mode. Use the SPRCOLOR command to select the sprite's multicolor.

SPRSAV —Store sprite data from a text string variable into a sprite storage area or vice versa

SPRSAV ⟨origin⟩,⟨destination⟩

This command transfers a sprite image from a string variable to a sprite storage area. It can also transfer the data from the sprite storage area into a string variable. Either the origin or the destination can be a sprite number or a string variable but they both cannot be string variables. If you are moving a string into a sprite, only the first 63 bytes of data are used. The rest are ignored since a sprite can only hold 63 data bytes.

EXAMPLES:

SPRSAV 1,A$	Transfers the image pattern from sprite 1 to the string named A$.
SPRSAV B$,2	Transfers the data from string variable B$ into sprite 2.
SPRSAV 2,3	Transfers the data from sprite 2 to sprite 3.

SSHAPE/GSHAPE —Save/retrieve shapes to/from string variables

SSHAPE and GSHAPE are used to save and load rectangular areas of multicolor or bit mapped screens to/from BASIC string variables. The command to save an area of the screen into a string variable is:

SSHAPE string variable, X1, Y1 [,X2,Y2]

where:

string variable . . . . String name to save data in
X1,Y1 . . . . . Corner coordinate (0,0 through 319,199) (scaled)
X2,Y2 . . . . . Corner coordinate opposite (X1,Y1) (default is the PC)

Because BASIC limits strings to 255 characters, the size of the area that can be saved is limited. The string size required can be calculated using one of the following (unscaled) formulas:

L(mcm) = INT ((ABS(a1 − a2) + 1) / 4 + .99) * (ABS(b1 − b2) + 1) + 4

L(h − r) = INT ((ABS(a1 − a2) + 1) / 8 + .99) * (ABS(b1 − b2) + 1) + 4

The command to retrieve (load) the data from a string variable and display it on specified screen coordinates is:

GSHAPE string variable [X,Y] [,mode]

where:

string.....Contains shape to be drawn
X,Y.....Top left coordinate (0,0 through 319,199) telling where to draw the shape (scaled—the default is the pixel cursor)
mode.....Replacement mode:
0: place shape as is (default)
1: invert shape
2: OR shape with area
3: AND shape with area
4: XOR shape with area

The replacement mode allows you to change the data in the string variable so you can invert it, perform a logical OR, exlusive OR or AND operation on the image. The X and Y values can place the pixel cursor at absolute coordinates such as (100,100) or at coordinates relative to the previous position (+ / – X and + / – Y) of the pixel cursor such as (+ 20, – 10). The coordinate of one axis (X or Y) can be relative and the other can be absolute. Here are the posible combinations of ways to specify the X and Y coordinates.

x,y	absolute x, absolute y
+/–x,y	relative x, absolute y
x,+/–y	absolute x, relative y
+/–x,+/–y	relative x, relative y

Also see the LOCATE command for information on the pixel cursor.

EXAMPLES:

SSHAPE A$,10,10	Saves a rectangular area from the coordinates 10,10 to the location of the pixel cursor, into string variable A$.
SSHAPE B$,20,30,47,51	Saves a rectangular area from top left coordinate (20,30) through bottom right coordinate (47,51) into string variable B$.
SSHAPE D$,+10,+10	Saves a rectangular area 10 pixels to the right and 10 pixels down from the current position of the pixel cursor.
GSHAPE A$,120,20	Retrieves shape contained in string variable A$ and displays it at top left coordinate (120,20).

GSHAPE B$,30,30,1	Retrieves shape contained in string variable B$ and displays it at top left coordinate 30,30. The shape is inverted due to the replacement mode being selected by the 1.
GSHAPE C$, +20, +30	Retrieve shape from string variable C$ and displays it 20 pixels to the right and 30 pixels down from the current position of the pixel cursor.

NOTE: Beware using modes 1-4 with multicolor shapes. You may obtain unpredictable results.

STASH —Move contents of host memory to expansion RAM

STASH #bytes, intsa, expb, expsa

Refer to FETCH command for description of parameters.

STOP —Halt program execution

STOP

This statement halts the program. A message, BREAK IN LINE XXX, occurs (only in program mode), where XXX is the line number containing the STOP command. The program can be restarted at the statement following STOP if the CONT command is used immediately, without any editing occurring in the listing. The STOP statement is often used while debugging a program.

SWAP —Swap contents of host RAM with contents of expansion RAM

SWAP #bytes, intsa, expb, expsa

Refer to FETCH command for description of parameters.

SYS —Call and execute a machine language subroutine at the specified address

SYS address [,a] [,x] [,y] [,s]

This statement performs a call to a subroutine at a given address in a memory configuration set up according to the BANK command. Optionally, arguments a,x,y and s are loaded into the accumulator, x, y and status registers, respectively before the subroutine is called.

The address range is 0 to 65535. The program begins executing the machine-language program starting at that memory location. Also see the BANK command.

EXAMPLES:

SYS 40960 Calls and executes the machine-language routine at location 40960.

SYS 8192,0 Calls and executes the machine-language routine at location 8192 and load zero into the accumulator.

TEMPO

—Define the speed of the song being played

TEMPO n

where n is a relative duration between (0 and 255)

The actual duration for a whole note is determined by using the formula given below:

whole note duration = 19.22/n seconds

The default value is 8, and note duration increases with n.

EXAMPLES:

TEMPO 16 Defines the Tempo at 16.
TEMPO 1 Defines the TEMPO at the slowest speed.
TEMPO 250 Defines the TEMPO at 250.

TRAP

—Detect and correct program errors while a BASIC program is RUNning

TRAP [line #]

When turned on, TRAP intercepts most error conditions (excluding DOS error messages but including the STOP KEY) except an "UNDEF'D STATEMENT ERROR." In the event of any execution error, the error flag is set and execution is transferred to the line number specified in the TRAP statement. The line number in which the error occurred can be found by using the system variable EL. The specific error condition is contained in system variable ER. The string function ERR$ (ER) gives the error message corresponding to any error condition.

The RESUME statement can be used to resume program execution. TRAP with no line number turns off error trapping. An error in a TRAP routine cannot be trapped. Also see system variables ST, DS and DS$.

EXAMPLES:

100 TRAP 1000	If an error occurs, go to line 1000.
1000 ?ERR$ (ER);EL	Print the error message, and the error number.
1010 RESUME	Resume with program execution.

TROFF

—Turn off error TRACing mode

TROFF

This statement turns off trace mode.

TRON

—Turn on error TRACing mode

TRON

TRON is used in program debugging. This statement begins trace mode. When you RUN the program, the line numbers of the program appear in brackets before any action for that line occurs.

VERIFY

—Verify program in memory against one saved to disk or tape

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

This command causes the Commodore 128 to check the program on tape or disk against the one in memory, to determine if the program is really SAVEd. This command is also very useful for positioning a tape so that the Commodore 128 writes after the last program on the tape. It will do so, and inform the user that the programs don't match. The tape is then positioned properly, and the next program can be stored without fear of erasing the old one.

VERIFY, with no arguments after the command, causes the Commodore 128 to check the next program on tape, regardless of its name, against the program now in memory. VERIFY, followed by a program name in quotes or a string variable in parentheses, searches the tape for that program and then checks it against the program in memory when found. VERIFY, followed by a name, a comma and a number, checks the program on the device with that number (1 for tape, 8 for disk). The relocate flag is the same as in the LOAD command. It verifies the program from the memory location from which it was SAVEd.

EXAMPLES:

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

NOTE: If a graphic area is reallocated for use after a SAVE, VERIFY and DVERIFY will report an error. Technically this is correct. BASIC text in this case has been moved from its original (saved) location to another address range. Hence, VERIFY, which performs byte-to-byte comparisons, will fail, even though the program is valid.

VOL —Define output level of sound

VOL volume level

This statement sets the volume for SOUND and PLAY statements. VOLUME level can be set from 0 to 15, where 15 is the maximum volume, and 0 is off. VOL affects all voices.

EXAMPLES:

VOL 0 Sets volume to its lowest level.

VOL 15 Sets volume for SOUND and PLAY statements to its highest output.

WAIT —Pause program execution until a data condition is satisfied

WAIT ⟨Location⟩, ⟨mask-1⟩ [,mask-2⟩]

The WAIT statement causes program execution to be suspended until a given memory address recognizes a specified bit pattern or value. In other words, WAIT can be used to halt the program until some external event has occurred. This is done by monitoring the status of bits in the Input/Output registers. The data items used with the WAIT can be any values. For most programmers, this statement should never be used. It causes the program to halt until a specific memory location's bits change in a specific way. This is used for certain I/O operations and almost nothing else. The WAIT statement takes the value in the memory location and performs a logical AND operation with the value in mask-1. If mask-2 is specified, the result of the first operation is exclusively ORed with mask-2. In other words, mask-1 "filters out" any bits not to be tested. Where the bit is 0 in mask-1, the corresponding bit in the result will always be 0. The mask-2 value flips any bits, so that an off condition can be tested for

as well as an on condition. Any bits being tested for a 0 should have a 1 in the corresponding position in mask-2. If corresponding bits of the ⟨mask-1⟩ and ⟨mask-2⟩ operands differ, the exclusive-OR operation gives a bit result of 1. If the corresponding bits get the same result the bit is 0. It is possible to enter an infinite pause with the WAIT statement, in which case the RUN/STOP and RESTORE keys can be used to recover. WAIT may require a BANK command if the memory you wish to access is not in the currently selected BANK.

The first example below WAITs until a key is pressed on the tape unit to continue with the program. The second example will WAIT until a sprite collides with the screen background.

EXAMPLES:

```
WAIT 1, 32, 32
WAIT 53273, 6, 6
WAIT 36868, 144, 16
```

(144 and 16 are binary masks. 144 = 10010000 in binary and 16 = 10000 in binary.)

WIDTH

—Set the width of drawn lines

WIDTH n

This command sets the width of lines drawn using BASIC's graphic commands to either single or double width. Giving n a value of 1 defines a single width line; a value of 2 defines a double width line.

EXAMPLES:

WIDTH 1 Set single width for graphic commands
WIDTH 2 Set double width for drawn lines

WINDOW

—Defines a screen window

WINDOW top left col,top left row,bot right col, bot right row[,clear]

This command defines a logical window within the 40 or 80 column text screen. The coordinates must be in the range 0-39/79 for column values and 0-24 for row values screen. The clear flag, if provided (1), causes a screen-clear to be performed (but only within the limits of the newly described window).

EXAMPLES:

WINDOW 5,5,35,20 Defines a window with top left corner coordinate as 5,5 and bottom right corner coordinate as 35,20.

WINDOW 10,2,33,24,1 Defines a window with upper left corner coordinate (10,2) and lower right corner coordinate (33,24). Also clears the portion of the screen with the window as specified by the 1.

SECTION 18
Basic Functions

The format of the function description is:

FUNCTION (argument)

where the argument can be a numeric value, variable or string.

Each function description is followed by an EXAMPLE. The lines appearing in bold face in the examples are the functions you type in. The line without bold is the computer's response.

ABS

—Return absolute value

ABS (X)

The absolute value function returns the positive value of the argument X.

EXAMPLE:

PRINT ABS (7*(– 5))

35

ASC

—Return CBM ASCII code for character

ASC(X$)

This function returns the ASCII code of the first character of X$. You no longer have to append CHR$(Ø) to a null string. ILLEGAL QUANTITY ERROR is no longer issued.

EXAMPLE:

X$ = "C128":PRINT X$

65

ATN

—Return angle whose tangent is X radians

ATN (X)

This function returns the angle whose tangent is X, measured in radians.

EXAMPLE:

PRINT ATN (3)

1.24904577

BUMP

—Return sprite collision information

BUMP (N)

To determine which sprites have collided since the last check, use the BUMP function. BUMP(1) records which sprites have collided with each other and BUMP(2) records which sprites have collided with other objects on the screen. COLLISION need not be active to use BUMP. The bit positions (0-7) in the BUMP value correspond to sprites 1 through 8 respectively. BUMP(n) is reset to zero after each call.

The value returned by BUMP is the result of two raised to the power of the bit position. For example, if BUMP returned a value of 16, sprite 4 was involved in a collision since 2 raised to the fourth power equals 16.

EXAMPLES:

PRINT BUMP (1) Indicates that sprite 2 and 3 have collided.
12
PRINT BUMP (2) Indicates that sprite 5 has collided with an object on the screen.
32

CHR$

—Return ASCII character for specified CBM ASCII code

CHR$(X)

This is the opposite of ASC and returns the string character whose CBM ASCII code is X. Refer to Appendix E for a table of CHR$ codes.

EXAMPLES:

PRINT CHR$ (65) Prints the A character.
A
PRINT CHR$ (147) Clears the text screen.

COS

—Return cosine for angle of X radians

COS(X)

This function returns the value of the cosine of X, where X is an angle measured in radians.

EXAMPLE:

PRINT COS (π/3)
.5

DEC —Return decimal value of hexadecimal number string

DEC (hexidecimal-string)

This function returns the decimal value of hexadecimal-string.

EXAMPLE:

```
PRINT DEC ("D020")
53280
```

ERR$ —Return the string describing an error condition

ERR$(N)

This function returns a string describing an error condition. Also see system variables EL and ER and Appendix A for a list of BASIC error messages.

EXAMPLE:

```
PRINT ERR$(ER)
ILLEGAL QUANTITY ERROR
```

EXP —Return value of an approximation of e (2.7182813) raised to the X power

EXP(X)

This function returns a value of e (2.7182813) raised to the power of X.

EXAMPLE:

```
PRINT EXP(1)
2.7182813
```

FNxx —Return value from user defined function

FNxx(x)

This function returns the value from the user-defined function xx created in a DEF FNxx statement.

EXAMPLE:

```
10 DEF FNAA(X)=(X-32)*5/9
20 INPUT X
30 PRINT FNAA(X)
RUN
? 40 (? is input prompt)
4.44444445
```

FRE —Return number of available bytes in memory

FRE (X)

where X is the bank number. X = 0 for BASIC program storage and X = 1 to check for available BASIC variable storage.

EXAMPLES:

PRINT FRE (0) Returns the number of free bytes for BASIC programs.
48893

PRINT FRE (1) Returns the number of free bytes for BASIC variable storage.
64256

HEX$ —Return hexadecimal number string from decimal number

HEX$(X)

This function returns a four-character string containing the hexadecimal representation of value X (0 < = X < 65535). The decimal counterpart of this function is DEC.

EXAMPLE:

```
PRINT HEX$(53280)
D020
```

INSTR —Return position of string 1 in string 2

INSTR (string 1, string 2 [,starting position])

The INSTR function searches for the first occurrence of string 2 within string 1, and returns the position within the string where the match is found. The optional parameter for STARTING POSITION establishes the position in string 1 where the search begins. The STARTING POSITION must be in the range 1 through 255. If no match is found or, if the STARTING POSITION is greater than the length of string 1 or if string 1 is null, INSTR returns the value 0. If

string 2 is null, INSTR returns the value of the STARTING POSITION or the value 1.

EXAMPLE:

PRINT INSTR ("COMMODORE 128","128")
11

INT —Return integer form (truncated) of a floating point value

INT(X)

This function returns the integer value of the expression. If the expression is positive, the fractional part is left out. If the expression is negative, any fraction causes the next lower integer to be returned.

EXAMPLES:

PRINT INT(3.14)
3

PRINT INT(– 3.14)
– 4

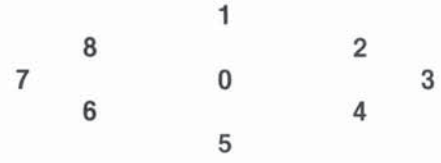
JOY —Return position of joystick and the status of the fire button

JOY(N)

when N equals:

1 JOY returns position of joystick 1.
2 JOY returns position of joystick 2.

Any value of 128 or more means that the fire button is also pressed. To find the JOY value, add the direction value of the joystick plus 128, if the fire button is pressed. The direction is indicated as follows:



EXAMPLES:

JOY (2) = 135
Joystick 2 fires to the left.

IF (JOY (1) AND 128) = 128 THEN PRINT "FIRE".
Determines whether the fire button is pressed.

LEFT$ —Return the leftmost characters of string

LEFT$ (string,integer)

This function returns a string comprised of the number of leftmost characters of the string determined by the specified integer. The integer argument must be in the range 0 to 255. If the integer is greater than the length of the string, the entire string is returned. If an integer value of zero is used, then a null string (of zero length) is returned.

EXAMPLE:

PRINT LEFT$ ("COMMODORE",5)
COMMO

LEN —Return the length of a string

LEN (string)

This function returns the number of characters in the string expression. Non-printed characters and blanks are included.

EXAMPLE:

PRINT LEN ("COMMODORE128")
12

LOG —Return natural log of X

LOG(X)

This function 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).

EXAMPLE:

PRINT LOG (37/5)
2.00148

MID$ —Return a substring from a larger string

MID$ (string,starting position[,length])

This function returns a substring specified by the LENGTH, starting at the character specified by the starting position. The starting position of the substring defines the first character where the substring begins. The length of the substring is specified by the length arguement. Both of the numeric arguments can have values ranging from

0 to 255. If the starting position value is greater than the length of the string, or if the length value is zero, then MID$ returns a null string value. If the length argument is left out, all characters to the right of the starting position are returned.

EXAMPLE:

```
PRINT MID$("COMMODORE 128",3,5)
MMODO
```

PEEK —Return contents of a specified memory location

PEEK(X)

This function returns the contents of memory location X, where X is located in the range 0 to 65535, returning a result between 0 and 255. This is the counterpart of the POKE statement. The data will be returned from the bank selected by the most recent BANK command. See the BANK command.

EXAMPLE:

```
10 BANK 15:VIC = DEC("D000")
20 FOR I = 1 TO 47
30 PRINT PEEK(VIC + I)
40 NEXT
```

This example displays the contents of the registers of the VIC chip.

PEN —Return X and Y coordinates of the light pen

PEN(n)

where n = 0 PEN returns the X coordinate of light pen position.
n = 1 PEN returns the Y coordinate of light pen position.
n = 2 PEN returns the X coordinate of the 80 column display.
n = 3 PEN returns the Y coordinate of the 80 column display.
n = 4 PEN returns the light pen trigger value.

Note that, like sprite coordinates, the PEN value is not scaled and uses real coordinates, not graphic bit map coordinates. The X position is given as an even number, ranging from approximately 60 to 320, while the Y position can be any number from 50 to 250. These are the visible screen coordinate ranges, where all other values are not visible on the screen. A value of zero for either position means the light pen is off screen and has not triggered an interrupt since

the last read. Note that COLLISION need not be active to use PEN. A white background is usually required to stimulate the light pen. PEN values vary from CRT to CRT.

Unlike the 40 column (VIC) screen, the 80 column (8563) coordinates are character row and column positions and not pixel coordinates like the VIC screen. Both the 40 and 80 column screen coordinate values are approximate and vary, due to the nature of light pens. The read values are not valid until PEN(4) is true.

EXAMPLES:

```
10 PRINT PEN(0);PEN(1)      Displays the X and Y coordinates of
                            the light pen.
10 DO UNTIL PEN(4):LOOP     Ensures the read values are valid.
20 X = PEN(2)
30 Y = PEN(3)
40 REM:REST OF PROGRAM
```

π

—Return the value of pi (3.14159265)

π

EXAMPLE:

PRINT π This returns the result 3.14159265.

POINTER

—Return the address of a variable name

POINTER (variable name)

EXAMPLE:

A = POINTER (Z) This example returns the address of variable Z.

POS

—Return the current cursor column position within the current screen window

POS(X)

The POS function indicates where the cursor is within the defined screen window. X is a dummy argument, which must be specified, but the value is ignored.

EXAMPLE:

```
PRINT POS(0)
10
```

This displays the current cursor position within the defined text window, in this case 10.

POT

—Returns the value of the game-paddle potentiometer

POT (n)

when:

n = 1, POT returns the position of paddle #1
n = 2, POT returns the position of paddle #2
n = 3, POT returns the position of paddle #3
n = 4, POT returns the position of paddle #4

The values for POT range from 0 to 255. Any value of 256 or more means that the fire button is also depressed.

EXAMPLE:

```
10 PRINT POT(1)
20 IF POT(1) > 256 THEN PRINT "FIRE"
```

This example displays the value of the game paddle 1.

RCLR

—Return color of color source

RCLR(N)

This function returns the color (1 through 16) assigned to the color source N (0< N < 6), where the following N values apply:

0 = 40-column background
1 = bit map foreground
2 = multicolor 1
3 = multicolor 2
4 = 40-column border
5 = 40- or 80-column character color
6 = 80-column background color

The counterpart to the RCLR function is the COLOR command.

EXAMPLE:

```
10 FOR I = 0 TO 6
20 PRINT "SOURCE";I;"IS COLOR CODE";RCLR(I)
30 NEXT
```

This example prints the color codes for all six color sources.

RDOT

—Return current position or color source of pixel cursor

RDOT (N)

where:

N = 0 returns the X coordinate of the pixel cursor
N = 1 returns the Y coordinate of the pixel cursor
N = 2 returns the color source of the pixel cursor

This function returns the location of the current position of the pixel cursor (PC) or the current color source of the pixel cursor.

EXAMPLES:

PRINT RDOT(0) Returns X position of PC
PRINT RDOT(1) Returns Y position of PC
PRINT RDOT(2) Returns color source of PC

RGR

—Return current graphic mode

RGR(X)

This function returns the current graphic mode. X is a dummy argument, which must be specified. The counterpart of the RGR function is the GRAPHIC command. The value returned by RGR(X) pertains to the following modes:

VALUE	GRAPHIC MODE
0	40 column (VIC) text
1	Standard bit map
2	Split screen bit map
3	Multicolor bit map
4	Split screen Multicolor bit map
5	80 column (8563) text

EXAMPLE:

PRINT RGR(0) Displays the current graphic mode;
1 in this case, standard bit map mode.

RIGHT$

—Return sub-string from rightmost end of string

RIGHT$ (⟨string⟩, ⟨numeric⟩)

This function returns a sub-string taken from the rightmost characters of the string argument. The length of the sub-string is defined by the length argument which can be any integer in the range of 0 to 255. If the value of the numeric expression is zero, then a null string is returned. If the value given in the length argument is greater than

the length of the string, the entire string is returned. Also see the LEFT$ and MID$ functions.

EXAMPLE:

```
PRINT RIGHT$("BASEBALL",5)
EBALL
```

RND

—Return a random number

RND (X)

This function returns a random number between 0 and 1. This is useful in games, to simulate dice roll and other elements of chance. It is also used in some statistical applications.

If X = 0 RND returns a random number based on the hardware clock.
If X > 1 RND generates a reproducible psuedo-random number based on the seed value below.
If X < 0 produces a random number which is used as a base called a 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 to 0-6 (actually, greater than zero and less than six). Then 1 is added, making the range greater than 1 and less than 7. The INT function truncates all the decimal places, leaving the result as a digit from 1 to 6.

EXAMPLES:

```
PRINT RND(0)                    Displays a random number
.507824123                      between 0 and 1.
PRINT INT(RND(1)*100 + 1)       Displays a random number
89                              between 1 and 100.
```

RSPCOLOR

—Return sprite multicolor values

RSPCOLOR (register)

When:

X = 1 RSPCOLOR returns the sprite multicolor 1.
X = 2 RSPCOLOR returns the sprite multicolor 2.

The returned color value is a value between 1 and 16. The counterpart of the RSPCOLOR function is the SPRCOLOR statement. Also see the SPRCOLOR statement.

EXAMPLE:

```
10 SPRITE 1,1,2,0,1,1,1
20 SPRCOLOR 5,7
30 PRINT"SPRITE MULTICOLOR 1 IS";RSPCOLOR(1)
40 PRINT"SPRITE MULTICOLOR 2 IS";RSPCOLOR(2)
RUN

SPRITE MULTICOLOR 1 IS 5
SPRITE MULTICOLOR 2 IS 7
```

In this example line 10 turns on sprite 1, colors it white, expands it in both the X and Y directions and displays it in multicolor mode. Line 20 selects sprite multicolors 1 and 2. Lines 30 and 40 print the RSPCOLOR values for multicolor 1 and 2.

RSPPOS

—Return the speed and position values of a sprite

RSPPOS (sprite number,position/speed)

where sprite number identifies which sprite is being checked, and position and speed specifies X and Y coordinates or the sprite's speed.

When position equals:

0 RSPPOS returns the current X position of the specified sprite.
1 RSPPOS returns the current Y position of the specified sprite.

When speed equals:

2 RSPPOS returns the speed (0-15) of the specified sprite.

EXAMPLE:

```
10 SPRITE 1,1,2
20 MOVSPR 1,45#13
30 PRINT RSPPOS (1,0);RSPPOS (1,1);RSPPOS (1,2)
```

This example returns the current X and Y sprite coordinates and the speed (13).

RSPRITE

—Return sprite characteristics

RSPRITE (sprite number,characteristic)

RSPRITE returns sprite characteristics that were specified in the SPRITE command. Sprite number specifies the sprite you are check-

ing and the characteristic specifies the sprite's display qualities as follows:

Characteristic	RSPRITE returns these values:	
0	Enabled(1) / disabled(0)	
1	Sprite color (1-16)	
2	Sprites are displayed in front of (0) or behind (1) objects on the screen	
3	Expand in X direction	yes = 1, no = 0
4	Expand in Y direction	yes = 1, no = 0
5	Multicolor	yes = 1, no = 0

EXAMPLE:

```
10 FOR I = 0 TO 5
20 PRINT RSPRITE (1,I)
30 NEXT
```

This example prints all 6 characteristics of sprite 1.

RWINDOW

—Returns the size of the current window

RWINDOW (n)

When **n** equals:

- **0** RWINDOW returns the number of lines in the current window.
- **1** RWINDOW returns the number of rows in the current window.
- **2** RWINDOW returns either of the values 40 or 80, depending on the current screen output format you are using.

The counterpart of the RWINDOW function is the WINDOW command.

EXAMPLE:

```
10 WINDOW 1,1,10,10
20 PRINT RWINDOW(0);RWINDOW(1);RWINDOW(2)
RUN
10 10 40
```

This example returns the number of lines (10) and columns (10) in the current window. This example assumes you are displaying the window in 40 column format.

SGN —Return sign of argument X

SGN(X)

This function returns the sign,(positive, negative or zero) of X. The result is + 1 if X > 0, 0 if X = 0, and − 1 if X < 0.

EXAMPLE:

```
PRINT SGN(4.5);SGN(0);SGN( - 2.3)
1 0 -1
```

SIN —Return sine of argument

SIN(X)

This is the trigonometric sine function. The result is the sine of X. X is measured in radians.

EXAMPLE:

```
PRINT SIN (π/3)
.866025404
```

SPC —Skip spaces on the screen

SPC (X)

This function is used in PRINT or PRINT# commands to control the formatting of data, as either output to the screen or output to a logical file. The number of SPaCes specified by X determines the number of characters to fill with spaces across the screen or in a file. For screen or tape files, the value of the argument is in the range 0 to 255 and for disk files the maximum is 254. For printer files, an automatic carriage-return and line-feed will be performed by the printer if a SPaCe is printed in the last character position of a line. No SPaCes are printed on the following line.

EXAMPLE

```
PRINT "COMMODORE";SPC(3);"128"
COMMODORE   128
```

SQR —Return square root of argument

SQR (X)

This function returns the value of the SQuare Root of X, where X is a positive number or 0. The value of the argument must not be negative, or the BASIC error message ?ILLEGAL QUANTITY is displayed.

EXAMPLE:
PRINT SQR(25)
5

STR$ —Return string representation of number

STR$ (X)

This function returns the STRing representation of the numeric value of the argument X. When the STR$ value is converted to each variable represented in the argument, any number displayed is preceded and followed by a space except for negative numbers which are preceded by a minus sign. The counterpart of the STR$ function is the VAL function.

EXAMPLE

PRINT STR$(123.45)
123.45

PRINT STR$(– 89.03)
– 89.03

PRINT STR$(1E20)
1E + 20

TAB —Moves cursor to tab position in present statement

TAB (X)

This function moves the cursor forward if possible to a relative position on the text screen given by the argument X, starting with the leftmost position of the current line. The value of the argument can range from 0 to 255. If the current print position is already beyond position X, TAB places the cursor in the X position in the next line. The TAB function can only be used with the PRINT statement, since it has no effect if used with the PRINT# to a logical file.

EXAMPLE:

10 PRINT"COMMODORE"TAB(25)"128"
COMMODORE 128

TAN

—Return tangent of arguement

TAN(X)

This function returns the tangent of X, where X is an angle in radians.

EXAMPLE:

```
PRINT TAN(.785398163)
1
```

USR

—Call user-defined subprogram

USR(X)

When this function is used, the program jumps to a machine language program whose starting point is contained in memory locations 4633($1219) and 4634($121A), (and 785($0311) and 786($0312) for C64 mode). The parameter X is passed to the machine-language program in the floating point accumulator. A value is returned to the BASIC program through the calling variable. You must redirect the value into a variable in your program in order to receive the value back from the floating point accumulator. An ILLEGAL QUANTITY ERROR results if you don't specify this variable. This allows the user to exchange a variable between machine code and BASIC.

EXAMPLE:

```
10 POKE 4633,0
20 POKE 4634,192
30 A = USR(X)
40 PRINT A
```

Place starting location ($C000 = 49152:$00 = 0:$C0 = 192) of machine language routine in location 4633 and 4634. Line 30 stores the returning value from the floating point accumulator.

VAL

—Return the numeric value of a number string

VAL(X$)

This function converts the string X$ into a number which is the inverse operation of STR$. The string is examined from the left-most character to the right, for as many characters as are in recognizable number format. If the Commodore 128 finds illegal characters, only the portion of the string up to that point is converted. If no numeric characters are present, VAL returns a 0.

EXAMPLE:

```
10 A$ = "120"
20 B$ = "365"
30 PRINT VAL A$ + B$
RUN
485
```

XOR —Return exclusive OR

XOR (n1,n2)

This function provides the exclusive OR of the argument values n1 and n2.

```
x = XOR (n1,n2)
```

where n1, n2, are 2 unsigned values (0-65535).

EXAMPLE:

```
PRINT XOR(128,64)
192
```

SECTION 19
Variables and Operators

VARIABLES 325

OPERATORS 327

Variables

The Commodore 128 uses three types of variables in BASIC. These are: normal numeric, integer numeric and string (alphanumeric).

Normal NUMERIC VARIABLES, also called floating point variables, can have any exponent value from −10 to +10, with up to nine digits of accuracy. When a number becomes larger than nine digits can show, as in +10 or −10, the computer displays it in scientific notation form, with the number normalized to one digit and eight decimal places, followed by the letter E and the power of 10 by which the number is multiplied. For example, the number 12345678901 is displayed as 1.23456789E+10.

INTEGER VARIABLES can be used when the number is from +32767 to −32768, and with no fractional portion. An integer variable is a number like 5, 10 or −100. Integers take up less space than floating point variables, particularly when used in an array.

STRING VARIABLES are those used for character data, which may contain numbers, letters and any other characters the Commodore 128 can display. An example of a string variable is "Commodore 128."

VARIABLE NAMES may consist of a single letter, a letter followed by a number or two letters. Variable names may be longer than two characters, but only the first two are significant. An integer is specified by using the percent sign (%) after the variable name. String variables have a dollar sign ($) after their names.

EXAMPLES:

Numeric Variable Names: A, A5, BZ
Integer Variable Names: A%, A5%, BZ%
String Variable Names: A$, A5$, BZ$

ARRAYS are lists of variables with the same name, using an extra number (or numbers) to specify an element of the array. Arrays are defined using the DIM statement and may be floating point, integer or string variable arrays. The array variable name is followed by a set of parentheses () enclosing the number of the variable in the list.

EXAMPLE:

A(7), BZ%(11), A$(87)

Arrays can 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 identifying the column (as if specifying a certain grid on a map).

EXAMPLE:

A(7,2), BZ%(2,3,4), Z$(3,2)

RESERVED VARIABLE NAMES are names reserved for use by the Commodore 128, and may not be used for another purpose. These are the variables DS, DS$, ER, ERR$, EL, ST, TI and TI$. KEYWORDS such as TO and IF or any other names that contain KEYWORDS, such as RUN, NEW or LOAD cannot be used.

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 I/O operation. In general, if the value of ST is 0, then the operation was successful.

TI and TI$ are variables that relate to the real time clock built into the Commodore 128. The system clock is updated every 1/60th of a second. It starts at 0 when the Commodore 128 is turned on, and is reset only by changing the value of TI$. The variable TI gives the current value of the clock in 1/60th of a second. TI$ is a string that reads the value of the real time clock as a 24-hour clock. The first two characters of TI$ contain the hour, the third and fourth characters are minutes and the fifth and sixth characters are seconds. This variable can be set to any value (so long as all characters are numbers) and will be updated automatically as a 24-hour clock.

EXAMPLE:

TI$ = "101530" Sets the clock to 10:15 and 30 seconds (AM).

The value of the clock is lost when the Commodore 128 is turned off. It starts at zero when the Commodore 128 is turned on, and is reset to zero when the value of the clock exceeds 235959 (23 hours, 59 minutes and 59 seconds).

The variable DS reads the disk drive command channel and returns the current status of the drive. To get this information in words, PRINT DS$. These status variables are used after a disk operation, like DLOAD or DSAVE, to find out why the error light on the disk drive is blinking.

ER, EL and ERR$ are variables used in error trapping routines. They are usually only useful within a program. ER returns the last error encountered since the program was RUN. EL is the line where the error occurred. ERR$ is a function that allows the program to print one of the BASIC error messages. PRINT ERR$(ER) prints out the proper error message.

Operators

The BASIC OPERATORS include ARITHMETIC, RELATIONAL and LOGICAL OPERATORS. The ARITHMETIC operators include the following signs:

+ addition
– subtraction
*** multiplication**
/ division
↑ raising to a power (exponentiation)

On a line containing more than one operator, there is a set order in which operations always occur. If several operators are used together, the computer assigns priorities as follows: First, exponentiation, then multiplication and division, and last, addition and subtraction. If two operators have the same priority, then calculations are performed in order from left to right. If these operations are to occur in a different order, Commodore 128 BASIC allows giving a calculation a higher priority by placing parentheses around it. Operations enclosed in parentheses will be calculated before any other operation. Make sure the equations have the same number of left and right parentheses, or a SYNTAX ERROR message is posted when the program is run.

There are also operators for equalities and inequalities, called RELATIONAL operators. Arithmetic operators always take priority over relational operators.

=	**is equal to**
〈	**is less than**
〉	**is greater than**
〈= or =〈	**is less than or equal to**
〉= or =〉	**is greater than or equal to**
〈〉 or 〉〈	**is not equal to**

Finally, there are three LOGICAL operators, with lower priority than both arithmetic and relational operators:

AND
OR
NOT

These are most often used to join multiple formulas in IF . . . THEN statements. When they are used with arithmetic operators, they are evaluated last (i.e., after + and –). If the relationship stated in the

expression is true, the result is assigned an integer value of −1. If false, a value of 0 is assigned.

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>3	Displays a value of −1.
PRINT 123 AND 15:PRINT 5 OR 7	Displays 11 and 7.

SECTION 20
Reserved Words and Symbols

RESERVED SYSTEM WORDS (KEYWORDS) 331

RESERVED SYSTEM SYMBOLS 332

Reserved System Words (Keywords)

This section lists the words and symbols used to make up the BASIC 7.0 language. These words and symbols cannot be used within a program as other than a component of the BASIC language. The only exception is that they may be used within quotes in a PRINT statement.

ABS	DIM	HEADER	PRINT USING	SPRITE
AND	DIRECTORY	HELP	PRINT#	SPRSAV
APPEND	DLOAD	HEX$	PRINT# USING	SQR
ASC	DO	IF	PUDEF	ST
ATN	DOPEN	INPUT	RCLR	STASH
AUTO	DRAW	INPUT#	RDOT	STEP
BACKUP	D$	INSTR	READ	STOP
BANK	DSAVE	INT	RECORD	STR$
BEGIN	DS$	JOY	RENAME	SWAP
BEND	DVERIFY	KEY	RENUMBER	SYS
BLOAD	EL	LEFT$	RESTORE	TAB
BOOT	ELSE	LEN	RESUME	TAN
BOX	END	LET	RETURN	TEMPO
BSAVE	ENVELOPE	LIST	RGR	THEN
BUMP	ER	LOAD	RIGHT$	TI
CHAR	ERR$	LOCATE	RND	TI$
CHR$	EXIT	LOG	RSPCOLOR	TO
CIRCLE	EXP	LOOP	RSPPOS	TRAP
CLOSE	FAST	MID$	RSPRITE	TRON
CLR	FETCH	MONITOR	RUN	TROFF
CMD	FILTER	MOVSPR	RWINDOW	UNTIL
COLLECT	FN	NEW	SAVE	USR
COLOR	FOR	NEXT	SCALE	VAL
CONCAT	FRE	NOT	SCNCLR	VERIFY
CONT	GET	ON	SCRATCH	VOL
COPY	GETKEY	OPEN	SGN	WAIT
COS	GET#	OR	SIN	WHILE
DATA	G064	PAINT	SLEEP	WIDTH
DCLEAR	GOSUB	PEN	SLOW	WINDOW
DCLOSE	GOTO	PLAY	SOUND	XOR
DEC	GO TO	POS	SPC	
DEF FN	GRAPHIC	POT	SPRCOLOR	
DELETE	GSHAPE	PRINT	SPRDEF	

Reserved System Symbols

The following characters are reserved system symbols.

Symbol		Use(s)
+	Plus sign movement	Arithmetic addition; string concatenation; relative Pixel Cursor/sprite movement; declare decimal number in machine language monitor
–	Minus sign movement	Arithmetic subtraction; negative number; unary minus; relative pixel cursor/ sprite movement
*	Asterisk	Arithmetic multiplication
/	Slash	Arithmetic division
↑	Up arrow	Arithmetic exponentiation
	Blank space	Separate keywords and variable names
=	Equal sign	Value assignment; relationship testing
<	Less than	Relationship testing
>	Greater than	Relationship testing
,	Comma	Format output in variable lists; command/ statement function parameters
.	Period	Decimal point in floating point constants
;	Semicolon	Format output in variable lists
:	Colon	Separate multiple BASIC statements on a program line
""	Quotation mark	Enclose string constants
?	Question mark	Abbreviation for the keyword PRINT
(	Left parenthesis	Expression evaluation and functions
)	Right parenthesis	Expression evaluation and functions
%	Percent	Declare a variable name as integer; declare binary number in machine language monitor
#	Number	Precede the logical file number in input/ output statements
$	Dollar sign	Declare a variable name as a string and declares hexadecimal number in machine language monitor
&	And sign	Declare octal number in machine language monitor
π	Pi	Declare the numeric constant 3.141592654

APPENDICES

APPENDIX A — BASIC LANGUAGE ERROR MESSAGES
APPENDIX B — DOS ERROR MESSAGES
APPENDIX C — CONNECTORS/PORTS FOR PERIPHERAL EQUIPMENT
APPENDIX D — SCREEN DISPLAY CODES
APPENDIX E — ASCII AND CHR$ CODES
APPENDIX F — SCREEN AND COLOR MEMORY MAPS
APPENDIX G — DERIVED MATHEMATICAL FUNCTIONS
APPENDIX H — MEMORY MAP
APPENDIX I — CONTROL AND ESCAPE CODES
APPENDIX J — MACHINE LANGUAGE MONITOR
APPENDIX K — BASIC 7.0 ABBREVIATIONS
APPENDIX L — DISK COMMAND SUMMARY

APPENDIX A BASIC LANGUAGE ERROR MESSAGES

The following error messages are displayed by BASIC. Error messages can also be displayed with the use of the ERR$() function. The error numbers below refer only to the number assigned to the error for use with the ERR$() function.

ERROR #	ERROR NAME	DESCRIPTION
1	TOO MANY FILES	There is a limit of 10 files OPEN at one time.
2	FILE OPEN	An attempt was made to open a file using the number of an already open file.
3	FILE NOT OPEN	The file number specified in an I/O statement must be opened before use.
4	FILE NOT FOUND	Either no file with that name exists (disk) or an end-of-tape marker was read (tape).
5	DEVICE NOT PRESENT	The required I/O device is not available or buffers deallocated (cassette). Check to make sure the device is connected and turned on.
6	NOT INPUT FILE	An attempt was made to GET or INPUT data from a file that was specified as output only.
7	NOT OUTPUT FILE	An attempt was made to send data to a file that was specified as input only.
8	MISSING FILE NAME	File name missing in command.
9	ILLEGAL DEVICE NUMBER	An attempt was made to use a device improperly (SAVE to the screen, etc.).

10	NEXT WITHOUT FOR	Either loops are nested incorrectly, or there is a variable name in a NEXT statement that doesn't correspond with one in FOR.
11	SYNTAX	A statement not recognized by BASIC. This could be because of a missing or extra parenthesis, misspelled key word, etc.
12	RETURN WITHOUT GOSUB	A RETURN statement was encountered when no GOSUB statement was active.
13	OUT OF DATA	A READ statement encountered without data left unREAD.
14	ILLEGAL QUANTITY	A number used as the argument of a function or statement is outside the allowable range.
15	OVERFLOW	The result of a computation is larger than the largest number allowed (1.701411834E + 38).
16	OUT OF MEMORY	Either there is no more room for program code and/or program variables, or there are too many nested DO, FOR or GOSUB statements in effect.
17	UNDEF'D STATEMENT	A line number referenced doesn't exist in the program.
18	BAD SUBSCRIPT	The program tried to reference an element of an array out of the range specified by the DIM statement.
19	REDIM'D ARRAY	An array can only be DIMensioned once.

20	DIVISION BY ZERO	Division by zero is not allowed.
21	ILLEGAL DIRECT	INPUT or GET, or INPUT # or GET # statements are only allowed within a program.
22	TYPE MISMATCH	This occurs when a numeric value is used in place of a string or vice versa.
23	STRING TOO LONG	A string can contain up to 255 characters.
24	FILE DATA	Bad data read from a tape or disk file.
25	FORMULA TOO COMPLEX	The computer was unable to understand this expression. Simplify the expression (break into two parts or use fewer parentheses).
26	CAN'T CONTINUE	The CONT command does not work if the program was not RUN, there was an error, or a line has been edited.
27	UNDEF'D FUNCTION	A user-defined function was referenced that was never defined.
28	VERIFY	The program on tape or disk does not match the program in memory.
29	LOAD	There was a problem loading. Try again.
30	BREAK	The stop key was hit to halt program execution.
31	CAN'T RESUME	A RESUME statement was encountered without a TRAP statement in effect.

32	LOOP NOT FOUND	The program has encountered a DO statement and cannot find the corresponding LOOP.
33	LOOP WITHOUT DO	LOOP was encountered without a DO statement active.
34	DIRECT MODE ONLY	This command is allowed only in direct mode, not from a program.
35	NO GRAPHICS AREA	A command (DRAW, BOX, etc.) to create graphics was encountered before the GRAPHIC command was executed.
36	BAD DISK	An attempt failed to HEADER a diskette, because the quick header method (no ID) was attempted on an unformatted diskette or the diskette is bad.
37	BEND NOT FOUND	The program encountered an "IF . . . THEN BEGIN" or "IF . . . THEN . . . ELSE BEGIN" construct, and could not find a BEND keyword to match the BEGIN.
38	LINE # TOO LARGE	An error has occurred in renumbering a BASIC program. The given parameters result in a line number > 63999 being generated; therefore, the renumbering was not performed.
39	UNRESOLVED REFERENCES	An error has occurred in renumbering a BASIC program. A line number referred to by a command (e.g., GOTO 999) does not exist. Therefore, the renumbering was not performed.

40	UNIMPLEMENTED COMMAND	A command not supported by BASIC 7.0 was encountered.
41	FILE READ	An error condition was encountered while loading or reading a program or file from the disk drive (e.g., opening the disk drive door while a program was loading).

APPENDIX B DOS ERROR MESSAGES

The following DOS error messages are returned through the DS and DS$ variables. The DS variable contains just the error number and the DS$ variable contains the error number, the error message, and any corresponding track and sector number. NOTE: Error message numbers less than 20 should be ignored with the exception of 01, which gives information about the number of files scratched with the SCRATCH command.

ERROR NUMBER	ERROR MESSAGE AND DESCRIPTION
20	READ ERROR (block header not found) The disk controller is unable to locate the header of the requested data block. Caused by an illegal sector number, or the header has been destroyed.
21	READ ERROR (no sync character) The disk controller is unable to detect a sync mark on the desired track. Caused by misalignment of the read/write head, no diskette is present, or unformatted or improperly seated diskette. Can also indicate a hardware failure.
22	READ ERROR (data block not present) The disk controller has been requested to read or verify a data block that was not properly written. This error occurs in conjunction with the BLOCK commands and can indicate an illegal track and/or sector request.
23	READ ERROR (checksum error in data block) This error message indicates there is an error in one or more of the data bytes. The data has been read into the DOS memory, but the checksum over the data is in error. This message may also indicate hardware grounding problems.
24	READ ERROR (byte decoding error) The data or header has been read into the DOS memory but a hardware error has been created due to an invalid bit pattern in the data byte. This message may also indicate hardware grounding problems.

25 WRITE ERROR (write-verify error)
This message is generated if the controller detects a mismatch between the written data and the data in the DOS memory.

26 WRITE PROTECT ON
This message is generated when the controller has been requested to write a data block while the write protect switch is depressed. This is caused by using a diskette with a write protect tab over the notch.

27 READ ERROR
This message is generated when a checksum error has been detected in the header of the requested data block. The block has not been read into DOS memory.

28 WRITE ERROR
This error message is generated when a data block is too long and overwrites the sync mark of the next header.

29 DISK ID MISMATCH
This message is generated when the controller has been requested to access a diskette which has not been initialized. The message can also occur if a diskette has a bad header.

30 SYNTAX ERROR (general syntax)
The DOS cannot interpret the command sent to the command channel. Typically, this is caused by an illegal number of file names, or patterns are illegally used. For example, two file names appear on the left side of the COPY command.

31 SYNTAX ERROR (invalid command)
The DOS does not recognize the command. The command must start in the first position.

32 SYNTAX ERROR (invalid command)
The command sent is longer than 58 characters. Use abbreviated disk commands.

33 SYNTAX ERROR (invalid file name)
Pattern matching is invalidly used in the OPEN or SAVE command. Spell out the file name.

34 SYNTAX ERROR (no file given)
The file name was left out of the command or the DOS does not recognize it as such. Typically, a colon (:) has been left out of the command.

39 SYNTAX ERROR (invalid command)
This error may result if the command sent to the command channel (secondary address 15) is unrecognized by the DOS.

50 RECORD NOT PRESENT
Result of disk reading past the last record through INPUT# or GET# commands. This message will also occur after positioning to a record beyond end-of-file in a relative file. If the intent is to expand the file by adding the new record (with a PRINT# command), the error message may be ignored. INPUT # and GET # should not be attempted after this error is detected without first repositioning.

51 OVERFLOW IN RECORD
PRINT# statement exceeds record boundary. Information is truncated. Since the carriage return which is sent as a record terminator is counted in the record size, this message will occur if the total characters in the record (including the final carriage return) exceeds the defined size of the record.

52 FILE TOO LARGE
Record position within a relative file indicates that disk overflow will result.

60 WRITE FILE OPEN
This message is generated when a write file that has not been closed is being opened for reading.

61 FILE NOT OPEN
This message is generated when a file is being accessed that has not been opened in the DOS. Sometimes, in this case, a message is not generated; the request is simply ignored.

62 FILE NOT FOUND
The requested file does not exist on the indicated drive.

63 FILE EXISTS
The file name of the file being created already exists on the diskette.

64 FILE TYPE MISMATCH
The requested file access is not possible using files of the type named. Reread the chapter covering that file type.

65 NO BLOCK
Occurs in conjunction with Block Allocation. The sector you tried to allocate is already allocated. The track and sector numbers returned are the next higher track and sector available. If the track number returned is zero (0), all remaining sectors are full. If the diskette is not full yet, try a lower track and sector.

66 ILLEGAL TRACK AND SECTOR
The DOS has attempted to access a track or block which does not exist in the format being used. This may indicate a problem reading the pointer to the next block.

67 ILLEGAL SYSTEM T OR S
This special error message indicates an illegal system track or sector.

70 NO CHANNEL (available)
The requested channel is not available, or all channels are in use. A maximum of five buffers are available for use. A sequential file requires two buffers; a relative file requires three buffers; and the error/command channel requires one buffer. You may use any combination of those as long as the combination does not exceed five buffers.

71 DIRECTORY ERROR
The BAM (Block Availability Map) on the diskette does not match the copy on disk memory. To correct, initialize the diskette.

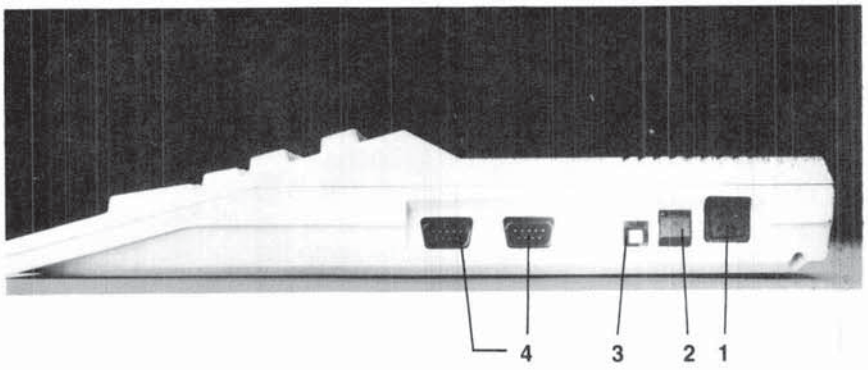
72 DISK FULL
Either the blocks on the diskette are used or the directory is at its entry limit. DISK FULL is sent when two blocks are still available on the diskette, in order to allow the current file to be closed.

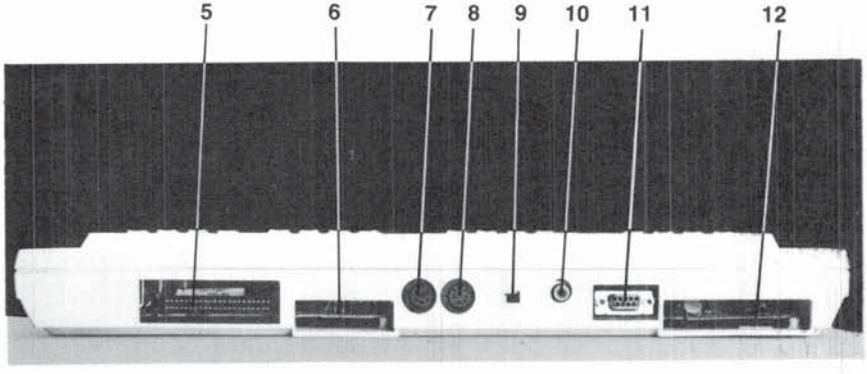
73 DOS VERSION NUMBER
DOS 1 and 2 are read compatible but not write compatible. Disks may be interchangeably read with either DOS, but a disk formatted on one version cannot be written upon with the other version because the format is different. This error is displayed whenever an attempt is made to write upon a disk which has been formatted in a non-compatible format. This message will also appear after power-up and is not an error in this case.

74 DRIVE NOT READY
An attempt has been made to access the disk drive without a diskette inserted; or the drive lever or door is open.

APPENDIX C CONNECTORS/ PORTS FOR PERIPHERAL EQUIPMENT

COMMODORE CONNECTIONS FOR PERIPHERALS





Side Panel Connections

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

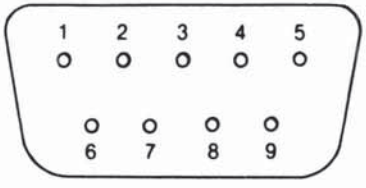
2. Power Switch—Turns on power from the transformer.

3. Reset Button—Resets computer (warm start).

4. Controller Ports—There are two Controller ports, numbered 1 and 2. Each Controller port can accept a joystick or game controller paddle. A light pen can be plugged only into port 1, 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

5. Expansion Port—This rectangular slot is a parallel port that accepts program or game cartridges as well as special interfaces.

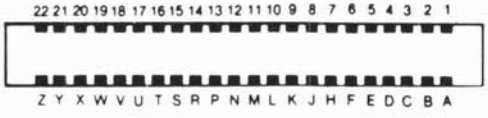
Cartridge Expansion Slot

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

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

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

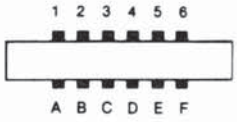
Pin	Type
A	GND
B	$\overline{\text{ROMH}}$
C	$\overline{\text{RESET}}$
D	$\overline{\text{NMI}}$
E	S 02
F	A15
H	A14
J	A13
K	A12
L	A11
M	A10



6. Cassette Port—A 1530 Datassette recorder can be attached here to store programs and information.

Cassette

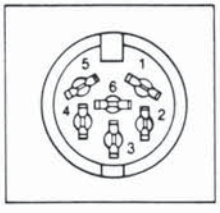
Pin	Type
A-1	GND
B-2	+5V
C-3	CASSETTE MOTOR
D-4	CASSETTE READ
E-5	CASSETTE WRITE
F-6	CASSETTE SENSE



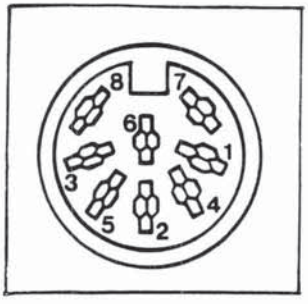
7. Serial Port—A Commodore serial printer or disk drive can be attached directly to the Commodore 128 through this port.

Serial I/O

Pin	Type
1	SERIAL $\overline{\text{SRQIN}}$
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	$\overline{\text{RESET}}$



8. Video Connector—This DIN connector supplies direct audio and composite video signals. These can be connected to the Commodore monitor or used with separate components. This is the 40 column output connector.

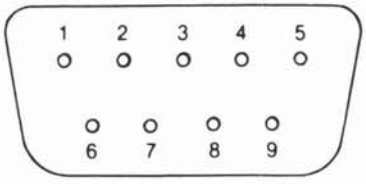


Pin	Type	Note
1	LUM/SYNC	Luminance/SYNC output
2	GND	
3	AUDIO OUT	
4	VIDEO OUT	Composite signal output
5	AUDIO IN	
6	COLOR OUT	Chroma signal output
7	NC	No connection
8	NC	No connection

9. Channel Selector—Use this switch to select which TV channel (L = channel 3, H = channel 4) the computer's picture will be displayed on when using a television instead of a monitor.

10. RF Connector—This connector supplies both picture and sound to your television set. (A television can display only a 40 column picture.)

11. RGBI Connector—This 9-pin connector supplies direct audio and an RGBI (Red/Green/Blue/Intensity) signal. This is the 80-column output.



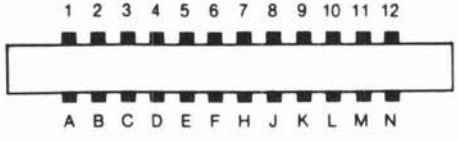
Pin	Signal
1	Ground
2	Ground
3	Red
4	Green
5	Blue
6	Intensity
7	Monochrome
8	Horizontal Sync
9	Vertical Sync

12. User Port—Various interface devices can be attached here, including a Commodore modem.

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	



APPENDIX D SCREEN DISPLAY CODES

Screen Display Codes 40 Columns

The following chart lists all of the characters built into the Commodore screen character sets. It shows which numbers should be POKEd into screen memory (location 1024 to 2023) to get a desired character on the 40-column screen. (Remember, to set color memory, use locations 55296 to 56295.) Also shown is which character corresponds to a number PEEKed from the screen.

Two character sets are available. Both are available simultaneously in 80-column mode, but only one is available at a time in 40-column mode. The sets are switched by holding down the SHIFT and C= (Commodore) keys simultaneously.

From BASIC, PRINT CHR$(142) will switch to upper-case/graphics mode and PRINT CHR$(14) will switch to upper/lower-case mode.

Any number on the chart may also be displayed in REVERSE. The reverse character code may be obtained by adding 128 to the values shown.

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	N	n	14	£		28
A	a	1	O	o	15	]		29
B	b	2	P	p	16	↑		30
C	c	3	Q	q	17	←		31
D	d	4	R	r	18	SPACE		32
E	e	5	S	s	19	!		33
F	f	6	T	t	20	"		34
G	g	7	U	u	21	#		35
H	h	8	V	v	22	$		36
I	i	9	W	w	23	%		37
J	j	10	X	x	24	&		38
K	k	11	Y	y	25	'		39
L	l	12	Z	z	26	(		40
M	m	13	[		27	)		41

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
*		42		G	71			100
+		43		H	72			101
,		44		I	73			102
–		45		J	74			103
.		46		K	75			104
/		47		L	76			105
0		48		M	77			106
1		49		N	78			107
2		50		O	79			108
3		51		P	80			109
4		52		Q	81			110
5		53		R	82			111
6		54		S	83			112
7		55		T	84			113
8		56		U	85			114
9		57		V	86			115
:		58		W	87			116
;		59		X	88			117
<		60		Y	89			118
=		61		Z	90			119
>		62			91			120
?		63			92			121
		64			93			122
	A	65			94			123
	B	66			95			124
	C	67	SPACE		96			125
	D	68			97			126
	E	69			98			127
	F	70			99			

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

APPENDIX E ASCII AND CHR$ CODES

ASCII and CHR$ Codes

This appendix shows you what characters will appear if you PRINT CHR$(X), for all possible values of X. It also shows the values obtained by typing PRINT ASC ("x"), where x is any character that can be displayed. This is useful in evaluating the character received in a GET statement, converting upper to lower case and printing character-based commands (like switch to upper/lower case) that could not be enclosed in quotes.

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	0		23	.	46	E	69
	1		24	/	47	F	70
	2		25	0	48	G	71
	3		26	1	49	H	72
	4		27	2	50	I	73
WHT	5	RED	28	3	51	J	74
	6	CRSR →	29	4	52	K	75
	7	GRN	30	5	53	L	76
DISABLES SHIFT C=	8	BLU	31	6	54	M	77
ENABLES SHIFT C=	9	SPACE	32	7	55	N	78
	10	!	33	8	56	O	79
	11	"	34	9	57	P	80
	12	#	35	:	58	Q	81
RETURN	13	$	36	;	59	R	82
SWITCH TO LOWER CASE	14	%	37	<	60	S	83
	15	&	38	=	61	T	84
	16	'	39	>	62	U	85
CRSR ↓	17	(	40	?	63	V	86
RVS ON	18	)	41	@	64	W	87
CLR HOME	19	*	42	A	65	X	88
INST DEL	20	+	43	B	66	Y	89
	21	,	44	C	67	Z	90
	22	-	45	D	68	[	91

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
£	92	♥	115	f4	138	▌	161
]	93	▯	116	f6	139	▄	162
↑	94	▯	117	f8	140	▯	163
←	95	▯	118	SHIFT RETURN	141	▯	164
▯	96	▯	119	SWITCH TO UPPER CASE	142	▯	165
♠	97	♣	120		143	▒	166
▯	98	▯	121	BLK	144	▯	167
▯	99	♦	122	CRSR	145	▒	168
▯	100	▯	123	RVS OFF	146	◢	169
▯	101	▒	124	CLR HOME	147	▐	170
▯	102	▯	125	INST DEL	148	▯	171
▯	103	▯	126	Brown	149	▗	172
▯	104	◥	127	Lt. Red	150	▖	173
▯	105		128	Dk. Gray	151	▘	174
▯	106	Orange	129	Gray	152	▀	175
▯	107		130	Lt. Green	153	▙	176
▯	108		131	Lt. Blue	154	▚	177
▯	109		132	Lt. Gray	155	▛	178
▯	110	f1	133	PUR	156	▜	179
▯	111	f3	134	←	157	▟	180
▯	112	f5	135	YEL	158	▞	181
●	113	f7	136	CYN	159	▝	182
▯	114	f2	137	SPACE	160	▘	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

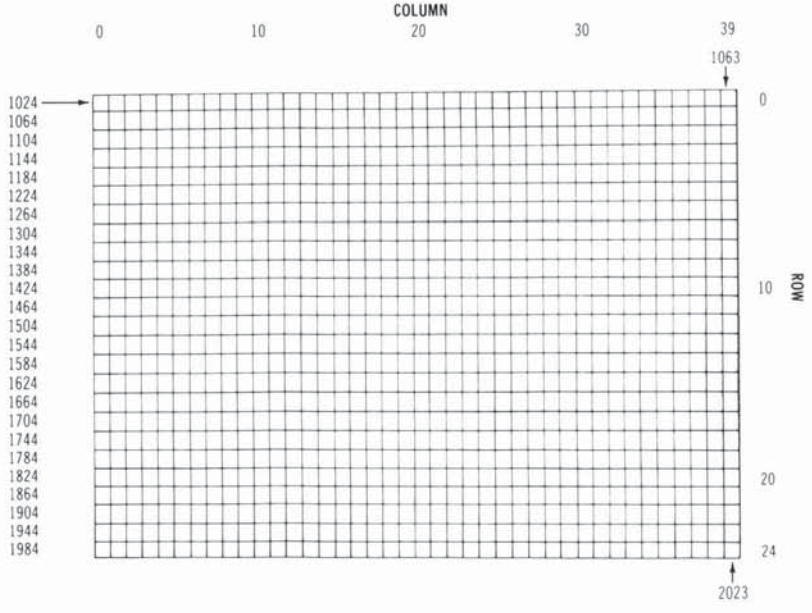
APPENDIX F SCREEN AND COLOR MEMORY MAPS

Screen And Color Memory Maps—C128 Mode, 40 Column And C64 Mode

The following maps display the memory locations used in 40-column mode (C128 and C64) for identifying the characters on the screen as well as their color. Each map is separately controlled and consists of 1,000 positions.

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

SCREEN MEMORY MAP

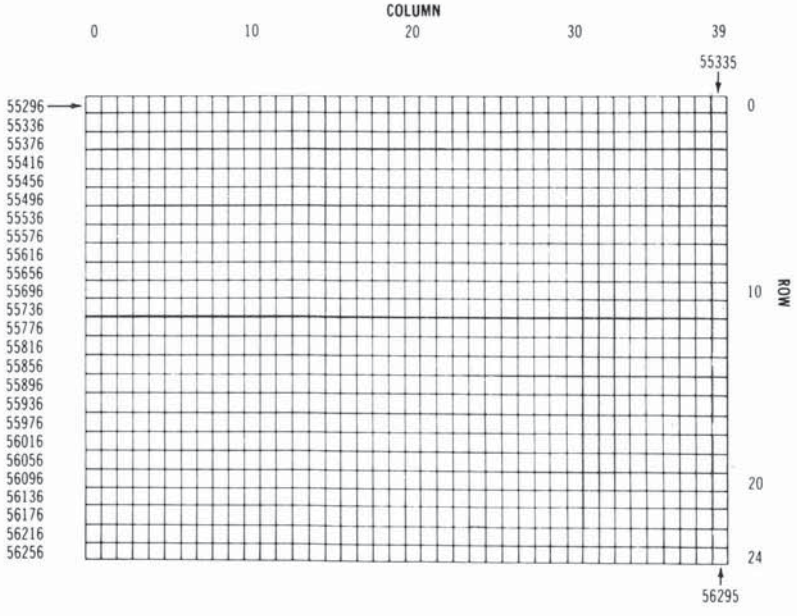


The Screen Map is POKEd with a Screen Display Code value (see Appendix D). For example:

POKE 1024, 13

will display the letter M in the upper-left corner of the screen.

COLOR MEMORY MAP



If the color map is POKEd with a color value; this changes the character color. For example:

POKE 55296, 1

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

Color Codes—40 Columns

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 G DERIVED TRIGONOMETRIC FUNCTIONS

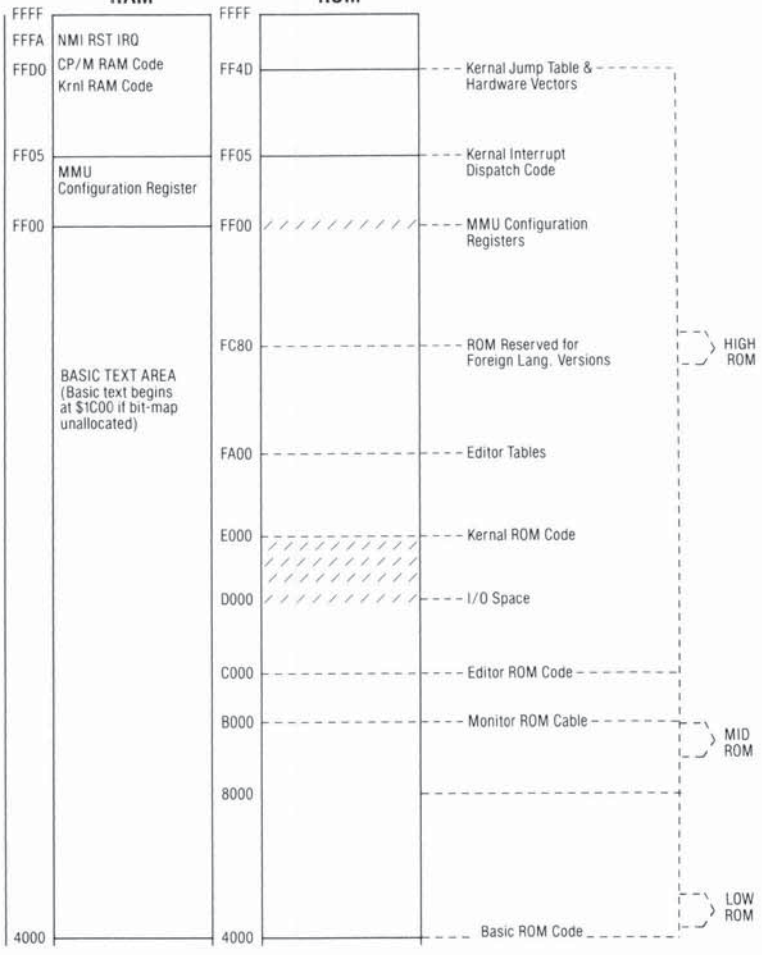
FUNCTION	BASIC EQUIVALENT
SECANT	SEC(X)=1/COS(X)
COSECANT	CSC(X)=1/SIN(X)
COTANGENT	COT(X)=1/TAN(X)
INVERSE SINE	ARCSIN(X)=ATN(X/SQR(−X*X+1))
INVERSE COSINE	ARCCOS(X)=−ATN(X/SQR (−X*X +1)) +π/2
INVERSE SECANT	ARCSEC(X)=ATN(X/SQR(X*X−1))
INVERSE COSECANT	ARCCSC(X)=ATN(X/SQR(X*X−1)) +(SGN(X)−1*π/2
INVERSE COTANGENT	ARCOT(X)=ATN(X)+π/2
HYPERBOLIC SINE	SINH(X)=(EXP(X)−EXP(−X))/2
HYPERBOLIC COSINE	COSH(X)=(EXP(X)+EXP(−X))/2
HYPERBOLIC TANGENT	TANH(X)=EXP(−X)/(EXP(x)+EXP (−X))*2+1
HYPERBOLIC SECANT	SECH(X)=2/(EXP(X)+EXP(−X))
HYPERBOLIC COSECANT	CSCH(X)=2/(EXP(X)−EXP(−X))
HYPERBOLIC COTANGENT	COTH(X)=EXP(−X)/(EXP(X) −EXP(−X))*2+1
INVERSE HYPERBOLIC SINE	ARCSINH(X)=LOG(X+SQR(X*X+1))
INVERSE HYPERBOLIC COSINE	ARCCOSH(X)=LOG(X+SQR(X*X−1))
INVERSE HYPERBOLIC TANGENT	ARCTANH(X)=LOG((1+X)/(1−X))/2
INVERSE HYPERBOLIC SECANT	ARCSECH(X)=LOG((SQR (−X*X+1)+1/X)
INVERSE HYPERBOLIC COSECANT	ARCCSCH(X)=LOG((SGN(X)*SQR (X*X+1/x)
INVERSE HYPERBOLIC COTAN-GENT	ARCCOTH(X)=LOG((X+1)/(x−1))/2

APPENDIX H
MEMORY MAP

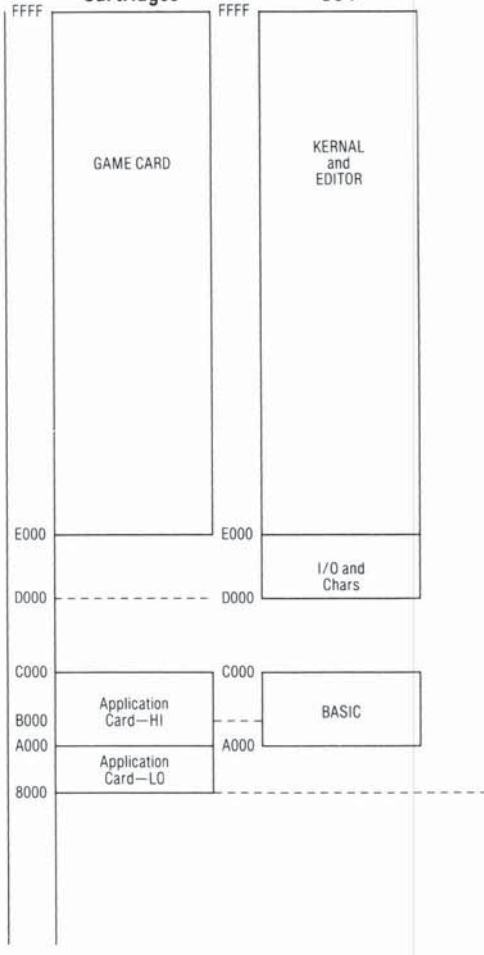
System Memory Map

The Commodore 128 memory map is shown below.

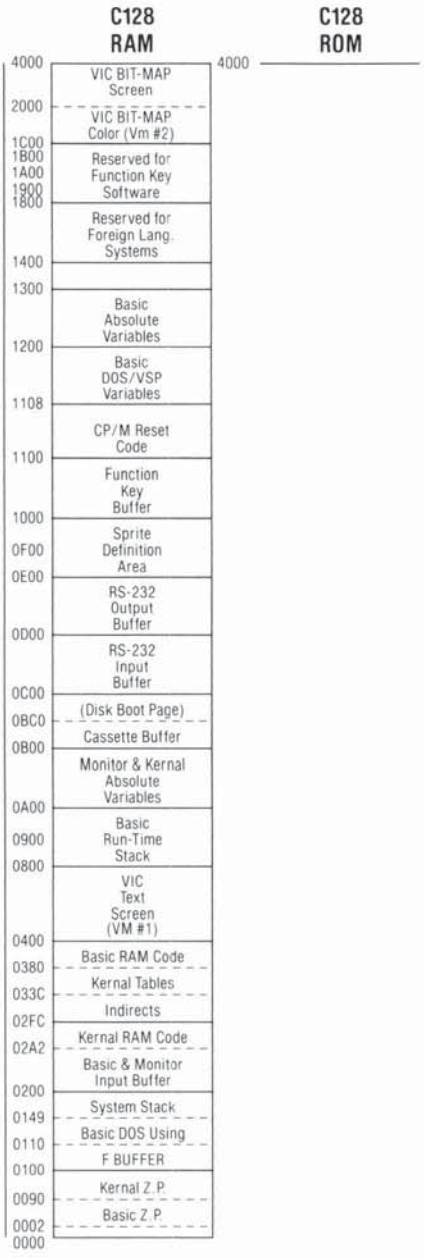
COMMODORE 128 MODE MEMORY MAP



COMMODORE 64 MODE MEMORY MAP



COMMODORE 128 MODE MEMORY MAP



COMMODORE 64 MODE MEMORY MAP



APPENDIX I CONTROL AND ESCAPE CODES

Control Codes

CHR$	Key Sequence	Function	Effective in Mode: C64	C128
CHR$(2)	CTRL B	Underline (80)		*
CHR$(5)	CTRL 2/CTRL E	Set character color to white (40) and (80)	*	*
CHR$(7)	CTRL G	Produce bell tone		*
CHR$(8)	CTRL H	Disable character set change	*	
CHR$(9)	CTRL I	Enable character set change	*	
		Move cursor to next set tab position		*
CHR$(10)	CTRL J	Send a carriage return with line feed.	*	
		Send a line feed		*
CHR$(11)	CTRL K	Enable character set change		*
CHR$(12)	CTRL L	Disable character mode change		*
CHR$(13)	CTRL M	Send a carriage return and line feed to the computer and enter a line of BASIC	*	*
CHR$(14)	CTRL N	Set character set to upper/lowercase	*	*
CHR$(15)	CTRL O	Turn flash on (80)		*
CHR$(17)	CRSR DOWN/CTRL Q	Move the cursor down one row		
CHR$(18)	CTRL 9	Cause characters to be printed in reverse field	*	*
CHR$(19)	HOME/CTRL S	Move the cursor to the home position (top left) of the display (the current window)	*	*
CHR$(20)	DEL/CTRL T	Delete last character typed and move all characters to the right of the deleted character one space to the left	*	*

NOTE: (40) . . . 40 column screen only
(80) . . . 80 column screen only

CHR$	Key Sequence	Function	Effective in Mode: C64	C128
CHR$(24)	CTRL X	Tab set/clear		
CHR$(27)	ESC/CTRL [	Send an ESC character		*
CHR$(28)	CTRL 3/CTRL /	Set character color to red (40) and (80)	*	*
CHR$(29)	CRSR /CTRL]	Move cursor one column to the right	*	*
CHR$(30)	CTRL 6/CTRL	Set character color to green (40) and (80)	*	*
CHR$(34)	"	Print a double quote on screen and place editor in quote mode	*	*
CHR$(129)	C= 1	Set character color to orange (40); dark purple (80)	*	*
CHR$(130)		Underline off (80)		*
CHR$(131)		Run a program. This CHR$ code does not work in PRINT CHR$ (131), but works from keyboard buffer	*	*
CHR$(133)	F1	Reserved CHR$ code for F1 key	*	*
CHR$(134)	F3	Reserved CHR$ code for F3 key	*	*
CHR$(135)	F5	Reserved CHR$ code for F5 key	*	*
CHR$(136)	F7	Reserved CHR$ code for F7 key	*	*
CHR$(137)	F2	Reserved CHR$ code for F2 key	*	*
CHR$(138)	F4	Reserved CHR$ code for F4 key	*	*
CHR$(139)	F6	Reserved CHR$ code for F6 key	*	*
CHR$(140)	F8	Reserved CHR$ code for F8 key	*	*
CHR$(141)	SHIFT RETURN	Send a carraige return and line feed without entering a BASIC line	*	*
CHR$(142)		Set the character set to uppercase/graphic	*	*
CHR$(143)		Turn flash off (80)		*

NOTE: (40) . . . 40 column screen only
(80) . . . 80 column screen only

CHR$	Key Sequence	Function	Effective in Mode: C64	C128
CHR$(144)	CTRL 1	Set character color to black (40) and (80)	*	*
CHR$(145)	CRSR UP	Move cursor or printing position up one row	*	*
CHR$(146)	CTRL 0	Terminate reverse field display	*	*
CHR$(147)	HOME	Clear the window screen and move the cursor to the top left position	*	*
CHR$(148)	INST	Move character from cursor position end of line right one column	*	*
CHR$(149)	C= 2	Set character color to brown (40); dark yellow (80)	*	*
CHR$(150)	C= 3	Set character color to light red (40) and (80)	*	*
CHR$(151)	C= 4	Set character color to dark grey (40) dark cyan (80)	*	*
CHR$(152)	C= 5	Set character color to medium grey (40) and (80)	*	*
CHR$(153)	C= 6	Set character color to light green (40) and (80)	*	*
CHR$(154)	C= 7	Set character color to light blue (40) and (80)	*	*
CHR$(155)	C= 8	Set character color to light grey (40) and (80)	*	*
CHR$(156)	CTRL 5	Set character color to purple (40) and (80)	*	*
CHR$(157)	CRSR LEFT	Move cursor left by one column	*	*
CHR$(158)	CTRL 4	Set character color to cyan (40); light cyan (80)	*	*

NOTE: (40) . . . 40 column screen only
(80) . . . 80 column screen only

Escape Codes

Following are key sequences for the ESCape functions available on the Commodore 128. ESCape sequences are entered by pressing and releasing the "ESC" key, followed by pressing the key listed below.

ESCAPE FUNCTION	ESCAPE KEY
Cancel quote and insert mode	ESC C
Erase to end of current line	ESC Q
Erase to start of current line	ESC P
Clear to end of screen	ESC @
Move to start of current line	ESC J
Move to end of current line	ESC K
Enable auto-insert mode	ESC A
Disable auto-insert mode	ESC O
Delete current line	ESC D
Insert line	ESC I
Set default tab stop (8 spaces)	ESC Y
Clear all tab stops	ESC Z
Enable scrolling	ESC L
Disable scrolling	ESC M
Scroll up	ESC V
Scroll down	ESC W
Enable bell (by control-G)	ESC G
Disable bell	ESC H
Set cursor to non-flashing mode	ESC E
Set cursor to flashing mode	ESC F
Set bottom of screen window at cursor position	ESC B
Set top of screen window at cursor position	ESC T
Swap 40/80 column display output device	ESC X

The following ESCape sequences are valid on an 80-column screen only. (See Section 8 for information on using an 80-column screen.)

Change to underlined cursor	ESC U
Change to block cursor	ESC S
Set screen to reverse video	ESC R
Return screen to normal (non reverse video) state	ESC N

NOTE: (40) . . . 40 column screen only
(80) . . . 80 column screen only

APPENDIX J MACHINE LANGUAGE MONITOR

Introduction

Commodore 128 has a built-in machine language monitor program which lets the user write and examine machine language programs easily. Commodore 128 MONITOR includes a machine language monitor, a mini-assembler and a disassembler. The built-in monitor works only in C128 mode; either 40 column or 80 column.

Machine language programs written using Commodore 128 MONITOR can run by themselves or be used as very fast subroutines for BASIC programs since the Commodore 128 MONITOR has the ability to coexist peacefully with BASIC.

Care must be taken to position the assembly language programs in memory so the BASIC program does not overwrite them.

To enter the monitor from BASIC, type:

MONITOR RETURN

Summary of Commodore 128 Monitor Commands

KEYWORD	FUNCTION		FORMAT
ASSEMBLE	Assembles a line of 8502 code	A	⟨start_address⟩ ⟨opcode⟩ [operand]
		C	⟨start_address⟩ ⟨end_address⟩ ⟨new_start_address⟩
COMPARE	Compares two sections of memory and reports differences		
DISASSEMBLE	Disassembles a line or lines of 8502 code	D	[⟨start_address⟩ ⟨end_address⟩]
FILL	Fills a range of memory with the specified byte	F	⟨start_address⟩ ⟨end_address⟩ ⟨byte⟩
GO	Starts execution at the specified address	G	[address]
		H	⟨start_address⟩ ⟨end_address⟩ ⟨byte1⟩ [⟨byte_n⟩...]
HUNT	Hunts through memory within a specified range for all occurrences of a set of bytes	H	⟨start_address⟩ ⟨end_address⟩ '⟨ascii_string⟩
JUMP	Jumps to the subroutine	J	[address]
LOAD	Loads a file from tape or disk	L	"⟨filename⟩" [,⟨device_#⟩ [,⟨load_address⟩]]
MEMORY	Displays the hexadecimal values of memory locations	M	[⟨start_address⟩ [end_address]]
REGISTERS	Displays the 8502 registers	R	
		S	"⟨filename⟩",⟨device_#⟩,⟨start_address⟩ ⟨last_address + 1⟩
SAVE	Saves to tape or disk		
		T	⟨start_address⟩ ⟨end_address⟩ ⟨new_start_address⟩
TRANSFER	Transfers code from one section of memory to another		
VERIFY	Compares memory with tape or disk	V	"⟨filename⟩" [,⟨device_#⟩ [,⟨load_address⟩]]
EXIT	Exits Commodore 128 MONITOR	X	
(period)	Assembles a line of 8502 code	.	
(greater than)	Modifies memory	>	
(semicolon)	Modifies 8502 register displays	;	
(at sign)	Displays disk status, sends disk command, displays directory	@	
	disk status		@[device_#]
	disk command		@[device_#],⟨command_string⟩]
	disk catalog		@[device_#],$[[⟨drive⟩][:⟨file_spec⟩]]

NOTES: ⟨ ⟩ enclose required parameters.
[ ] enclose optional parameters.

The Commodore 128 displays 5-digit hexadecimal addresses within the machine language monitor. Normally, a hexadecimal number is only four digits, representing the allowable address range. The extra left-most (high order) digit specifies the BANK configuration (at the time the given command is executed) according to the following memory configuration table:

0—RAM 0 only
1—RAM 1 only
2—RAM 2 only
3—RAM 3 only
4—INT ROM, RAM 0, I/O
5—INT ROM, RAM 1, I/O
6—INT ROM, RAM 2, I/O
7—INT ROM, RAM 3, I/O
8—EXT ROM, RAM 0, I/O
9—EXT ROM, RAM 1, I/O
A—EXT ROM, RAM 2, I/O
B—EXT ROM, RAM 3, I/O
C—KERNAL + INT (lo), RAM 0, I/O
D—KERNAL + EXT (lo), RAM 1, I/O
E—KERNAL + BASIC, RAM 0, CHARROM
F—KERNAL + BASIC, RAM 0, I/O

Summary of Monitor Field Descriptors

The following designators precede monitor data fields (e.g., memory dumps). When encountered as a command, these designators instruct the monitor to alter memory or register contents using the given data.

. ⟨period⟩ precedes lines of disassembled code.
) ⟨right_angle⟩ precedes lines of a memory dump.
; ⟨semicolon⟩ precedes line of a register dump.

The following designators precede number fields (e.g., address) and specify the radix (number base) of the value. Entered as commands, these designators instruct the monitor simply to display the given value in each of the four radices.

⟨null⟩ (default) precedes hexidecimal values.
$ ⟨dollar⟩ precedes hexidecimal (base-16) values.
+ ⟨plus⟩ precedes decimal (base-10) values.
& ⟨ampersand⟩ precedes octal (base-8) values.
% ⟨percent⟩ precedes binary (base-2) values.

The following characters are used by the monitor as field delimiters or line terminators (unless encountered within an ASCII string).

⟨space⟩ delimiter—separates two fields.
, ⟨comma⟩ delimiter—separates two fields.
: ⟨colon⟩ terminator—logical end of line.
? ⟨question⟩ terminator—logical end of line.

Commodore 128 Monitor Command Descriptions

Except as noted earlier, there are no changes at this time to the functionality of the MONITOR commands. Please note however that any number field (e.g. addresses, device numbers, and data bytes) may be specified as a based number. This affects the operand field of the ASSEMBLE command as well. Also note the addition of the directory syntax to the disk command.

As a further aid to programmers, the Kernal error message facility has been automatically enabled while in the Monitor. This means the Kernal will display 'I/O ERROR #' and the error code, should there be any failed I/O attempt from the MONITOR. The message facility is turned off when exiting the MONITOR.

COMMAND: **A**
PURPOSE: Enter a line of assembly code.
SYNTAX: **A** ⟨address⟩ ⟨opcode mnemonic⟩ ⟨operand⟩

⟨address⟩	A number indicating the location in memory to place the opcode.
⟨opcode mnemonic⟩	A standard MOS technology assembly language mnemonic, e.g., LDA, STX, ROR.
⟨operand⟩	The operand, when required, can be any of the legal addressing modes.

A RETURN is used to indicate the end of the assembly line. If there are any errors on the line, a question mark is displayed to indicate an error, and the cursor moves to the next line. The screen editor can be used to correct the error(s) on that line.

EXAMPLE
.A01200 LDX #$00
.A01202

NOTE: A period (.) is equal to the ASSEMBLE command.

EXAMPLE:
.02000 LDA #$23

COMMAND: **C**
PURPOSE: Compare two areas of memory.
SYNTAX: **C** ⟨address 1⟩ ⟨address 2⟩ ⟨address 3⟩

⟨address 1⟩	A number indicating the start address of the area of memory to compare against.
⟨address 2⟩	A number indicating the end address of the area of memory to compare against.
⟨address 3⟩	A number indicating the start address of the other area of memory to compare with. Addresses that do not agree are printed on the screen.

COMMAND: **D**
PURPOSE: Disassemble machine code into assembly language mnemonics and operands.
SYNTAX: **D** [⟨address⟩][⟨address 2⟩]

⟨address⟩	A number setting the address to start the disassembly.
⟨address 2⟩	An optional ending address of code to be disassembled.

The format of the disassembly differs slightly from the input format of an assembly. The difference is that the first character of a disassembly is a period rather than an A (for readability), and the hexidecimal of the code is listed as well.

A disassembly listing can be modified using the screen editor. Make any changes to the mnemonic or operand on the screen, then hit the carriage return. This enters the line and calls the assembler for further modifications.

A disassembly can be paged. Typing a D ⟨RETURN⟩ causes the next page of disassembly to be displayed.

EXAMPLE:

```
D 3000 3003
  .03000 A900      LDA #$00
  .03002 FF        ???
  .03003 D0 2B     BNE $3030
```

COMMAND: **F**
PURPOSE: Fill a range of locations with a specified byte.
SYNTAX: **F** ⟨address 1⟩ ⟨address 2⟩ ⟨byte⟩

⟨address 1⟩	The first location to fill with the ⟨byte⟩.
⟨address 2⟩	The last location to fill with the ⟨byte⟩.
⟨byte value⟩	A 1- or 2-digit hexadecimal number to be written.

This command is useful for initializing data structures or any other RAM area.

EXAMPLE:

F 0400 0518 EA

Fill memory locations from $0400 to $0518 with $EA (a NOP instruction).

COMMAND: **G**
PURPOSE: Begin execution of a program at a specified address.
SYNTAX: **G** [⟨address⟩]

⟨address⟩	An address where execution is to start. When address is left out, execution begins at the current PC. (The current PC can be viewed using the R command.)

The GO command restores all registers (displayable by using the R command) and begins execution at the specified starting address. Caution is recommended in using the GO command. To return to Commodore 128 MONITOR mode after executing a machine language program, use the BRK instruction at the end of the program.

EXAMPLE:

G 140C

Execution begins at location $140C.

COMMAND: **H**
PURPOSE: Hunt through memory within a specified range for all occurrences of a set of bytes.
SYNTAX: **H** ⟨address 1⟩ ⟨address 2⟩ ⟨data⟩

⟨address 1⟩	Beginning address of hunt procedure.
⟨address 2⟩	Ending address of hunt procedure.
⟨data⟩	Data set to search for data may be hexadecimal or an ASCII string.

EXAMPLE:

H A000 A101 A9 FF 4C

Search for data $A9, $FF, $4C, from A000 to A101.

H 2000 9800 'CASH'

Search for the alpha string "CASH".

COMMAND: **L**
PURPOSE: Load a file from cassette or disk.
SYNTAX: **L** ⟨"file name"⟩[,⟨device⟩ [,alt load address]]

⟨"file name"⟩	Any legal Commodore 128 file name.
⟨device⟩	A number indicating the device to load from. 1 is cassette. 8 is disk (or 9, A, etc.).
[alt load address]	Option to load a file to a specified address.

The LOAD command causes a file to be loaded into memory. The starting address is contained in the first two bytes of the disk file (a program file). In other words, the LOAD command always loads a file into the same place it was saved from. This is very important in machine language work, since few programs are completely relocatable. The file is loaded into memory until the end of file (EOF) is found.

EXAMPLE:

L "PROGRAM",8 Loads the file named PROGRAM from the disk.

COMMAND: **M**

PURPOSE: To display memory as a hexadecimal and ASCII dump within the specified address range.

SYNTAX: **M** [⟨address 1⟩][⟨address 2⟩]

⟨address 1⟩	First address of memory dump. Optional. If omitted, one page is displayed. The first byte is the bank number to be displayed, the next four bytes are the first address to be displayed.
⟨address 2⟩	Last address of memory dump. Optional. If omitted, one page is displayed. The first byte is the bank number to be displayed, the next four bytes are the ending address to be displayed.

Memory is displayed in the following format:

```
)1A048 41 E7 00 AA AA 00 98 56 45 :A!.*..VE
```

Memory content may be edited using the screen editor. Move the cursor to the data to be modified, type the desired correction and hit ⟨RETURN⟩. If there is a bad RAM location or an attempt to modify ROM has occurred, an error flag (?) is displayed. An ASCII dump of the data is displayed in REVERSE (to contrast with other data displayed on the screen) to the right of the hex data. When a character is not printable, it is displayed as a reverse period (▫). As with the disassembly command, paging down is accomplished by typing M and ⟨RETURN⟩.

EXAMPLE:

```
M 21C00 21C10

)21C00 41 E7 00 AA AA 00 98 56 45 :A!.*..VE
)21C08 42 43 02 AZ AD 11 94 57 44 :BC.*..WD
)21C10 45 E7 00 DF FE 07 06 46 47 :E!.*..EF
```

Note: The above display is produced by the 40-column editor.

COMMAND: **R**
PURPOSE: Show important 6502 registers. The program status register, the program counter, the accumulator, the X and Y index registers and the stack pointer are displayed.
SYNTAX: **R**

EXAMPLE:

```
   R
    PC   SR  AC  XR  YR  SP
;  01002 01  02  03  04  F6
```

NOTE: ; (semicolon) can be used to modify register displays in the same fashion as) can be used to modify memory registers.

COMMAND: **S**
PURPOSE: Save the contents of memory onto tape or disk.
SYNTAX: **S** ⟨"file name"⟩,⟨device⟩,⟨address 1⟩, ⟨address 2⟩

⟨"file name"⟩	Any legal Commodore 128 file name. To save the data the file name must be enclosed in double quotes. Single quotes cannot be used.
⟨device⟩	A number indicating on which device the file is to be placed. Cassette is 01; disk is 08, 09, etc.
⟨address 1⟩	Starting address of memory to be saved.
⟨address 2⟩	Ending address of memory to be saved + 1. All data up to, but not including the byte of data at this address, is saved.

The file created by this command is a program file. The first two bytes contain the starting address ⟨address 1⟩ of the data. The file may be recalled, using the L command.

EXAMPLE:

```
S "GAME",8,0400,0BFF
```

Saves memory from $0400 to $0BFF onto disk.

COMMAND: **T**

PURPOSE: Transfer segments of memory from one memory area to another.

SYNTAX: **T** ⟨address 1⟩ ⟨address 2⟩ ⟨address 3⟩

⟨address 1⟩	Starting address of data to be moved.
⟨address 2⟩	Ending address of data to be moved.
⟨address 3⟩	Starting address of new location where data will be moved.

Data can be moved from low memory to high memory and vice versa. Additional memory segments of any length can be moved forward or backward. An automatic "compare" is performed as each byte is transferred, and any differences are listed by address.

EXAMPLE:

T 1400 1600 1401

Shifts data from $1400 up to and including $1600 one byte higher in memory.

COMMAND: **V**

PURPOSE: Verify a file on cassette or disk with the memory contents.

SYNTAX: **V** ⟨"file name"⟩[,⟨device⟩][,alt start address]

⟨"file name"⟩	Any legal Commodore 128 file name.
⟨device⟩	A number indicating which device the file is on; cassette is 01, disk is 08, 09, etc.
[alt start address]	Option to start verification at this address.

The verify command compares a file to memory contents. The Commodore 128 responds with VERIFYING. If an error is found the word ERROR is added; if the file is successfully verified the cursor reappears.

EXAMPLE:

V "WORKLOAD", 08

COMMAND: **X**
PURPOSE: Exit to BASIC.
SYNTAX: **X**

COMMAND:) (greater than)
PURPOSE: Can be used to set one to eight memory locations at a time.
SYNTAX:) ⟨address⟩ ⟨data byte⟩ 1 ⟨data byte 2 . . . 8⟩

⟨address⟩	First memory address to set.
⟨data byte 1⟩	Data to be put at address.
⟨data byte 2 . . . 8⟩	Data to be placed in the successive memory locations following the first address (optional) with a space preceding each data byte.

COMMAND: @ (at sign)
PURPOSE: Can be used to display the disk status.
SYNTAX: @ [⟨unit#⟩], ⟨disk cmd string⟩

⟨unit #⟩	Device unit number (optional).
⟨disk cmd string⟩	String command to disk.

NOTE: @ alone gives the status of the disk drive.

EXAMPLES:

@	checks disk status
	00, 0K, 00, 00
@,I	initializes drive 8

APPENDIX K
BASIC 7.0
ABBREVIATIONS

Note: The abbreviations below operate in uppercase/graphics mode. Press the letter key(s) indicated, then hold down the SHIFT key and press the letter key following the word SHIFT.

KEYWORD	ABBREVIATION
ABS	A SHIFT B
APPEND	A SHIFT P
ASC	A SHIFT S
ATN	A SHIFT T
AUTO	A SHIFT U
BACKUP	BA SHIFT C
BANK	B SHIFT A
BEGIN	B SHIFT E
BEND	BE SHIFT N
BLOAD	B SHIFT L
BOOT	B SHIFT O
BOX	none
BSAVE	B SHIFT S
BUMP	B SHIFT U
CATALOG	C SHIFT A
CHAR	CH SHIFT A
CHR$	C SHIFT H
CIRCLE	C SHIFT I
CLOSE	CL SHIFT O
CLR	C SHIFT L
CMD	C SHIFT M
COLLECT	COLL SHIFT E
COLINT	none
COLLISION	COL SHIFT L
COLOR	COL SHIFT O
CONCAT	C SHIFT O
CONT	none
COPY	CO SHIFT P
COS	none
DATA	D SHIFT A
DEC	none
DCLEAR	DCL SHIFT E
DCLOSE	D SHIFT C
DEF FN	none
DELETE	DE SHIFT L
DIM	D SHIFT I
DIRECTORY	DI SHIFT R
DLOAD	D SHIFT L
DO	none
DOPEN	D SHIFT O

KEYWORD	ABBREVIATION
DRAW	D SHIFT R
DSAVE	D SHIFT S
DVERIFY	D SHIFT V
EL	none
END	none
ENVELOPE	E SHIFT N
ER	none
ERR$	E SHIFT R
EXIT	EX SHIFT I
EXP	E SHIFT X
FAST	none
FETCH	F SHIFT E
FILTER	F SHIFT I
FOR	F SHIFT O
FRE	F SHIFT R
FNXX	none
GET	G SHIFT E
GETKEY	GETK SHIFT E
GET #	none
GOSUB	GO SHIFT S
GO64	none
GOTO	G SHIFT O
GRAPHIC	G SHIFT R
GSHAPE	G SHIFT S
HEADER	HE SHIFT A
HELP	
HEX$	H SHIFT E
IF . . . GOTO	none
IF . . . THEN . . . ELSE	none
INPUT	none
INPUT#	I SHIFT N
INSTR	IN SHIFT S
INT	none
JOY	J SHIFT O
KEY	K SHIFT E
LEFT$	LE SHIFT F
LEN	none
LET	L SHIFT E
LIST	L SHIFT I
LOAD	L SHIFT O
LOCATE	LO SHIFT C
LOG	none
LOOP	LO SHIFT O

KEYWORD	ABBREVIATION
MID$	M SHIFT I
MONITOR	MO SHIFT N
MOVESHAPE	none
MOVSPR	M SHIFT O
NEW	none
NEXT	N SHIFT E
ON . . . GOSUB	ON . . . GO SHIFT S
ON . . . GOTO	ON . . . G SHIFT O
OPEN	O SHIFT P
PAINT	P SHIFT A
PEEK	PE SHIFT E
PEN	P SHIFT E
PI	none
PLAY	P SHIFT L
POKE	PO SHIFT K
POS	none
POT	P SHIFT O
PRINT	?
PRINT#	P SHIFT R
PRINT USING	?US SHIFT I
PUDEF	P SHIFT U
RBUMP	RB SHIFT U
RCLR	R SHIFT C
RDOT	R SHIFT D
READ	RE SHIFT A
RECORD	R SHIFT E
REM	none
RENAME	RE SHIFT N
RENUMBER	REN SHIFT U
RESTORE	RE SHIFT S
RESUME	RES SHIFT U
RETURN	RE SHIFT T
RGR	R SHIFT G
RIGHT$	R SHIFT I
RLUM	none
RND	R SHIFT N
RREG	R SHIFT R
RSPCOLOR	RSP SHIFT C
RSPPOS	R SHIFT S
RSPR	none
RSPRITE	RSP SHIFT R
RUN	R SHIFT U
RWINDOW	R SHIFT W

KEYWORD	ABBREVIATION
SAVE	S SHIFT A
SCALE	SC SHIFT A
SCNCLR	S SHIFT C
SCRATCH	SC SHIFT R
SGN	S SHIFT G
SIN	S SHIFT I
SLEEP	S SHIFT L
SLOW	none
SOUND	S SHIFT O
SPC(	none
SPRCOLOR	SPR SHIFT C
SPRDEF	SPR SHIFT D
SPRITE	S SHIFT P
SPRSAV	SPR SHIFT S
SQR	S SHIFT Q
SSHAPE	S SHIFT S
STASH	S SHIFT T
STatus	none
STEP	ST SHIFT E
STOP	ST SHIFT O
STR$	ST SHIFT R
SWAP	S SHIFT W
SYS	none
TAB(	T SHIFT A
TAN	none
TEMPO	T SHIFT E
TI	none
TI$	none
TO	none
TRAP	T SHIFT R
TROFF	TRO SHIFT F
TRON	TR SHIFT O
UNTIL	U SHIFT N
USR	U SHIFT S
VAL	none
VERIFY	V SHIFT E
VOL	V SHIFT O
WAIT	W SHIFT A
WHILE	W SHIFT H
WIDTH	WI SHIFT D
WINDOW	W SHIFT I
XOR	X SHIFT O

APPENDIX L DISK COMMAND SUMMARY

This appendix lists the commands used for disk operation in C128 and C64 modes on the Commodore 128. For detailed information on any of these commands, see Chapter V, BASIC 7.0 Encyclopedia. Your disk drive manual also has information on disk commands.

The **new BASIC 7.0** commands can be used **only** in C128 mode. **All** BASIC 2.0 commands can be used in **both** C128 and C64 modes.

Command	Use	Basic 2.0	Basic 7.0
APPEND	Append data to file		✓
BLOAD	Load a binary file starting at the specified memory location		✓
BOOT	Load and execute program		✓
BSAVE	Save a binary file from the specified memory location		✓
CATALOG	Display directory contents of disk on screen*		✓
CLOSE	Close logical disk file	✓	
CMD	Redirect screen output to disk file	✓	
COLLECT	Free inaccessible disk space*		✓
CONCAT	Concatenate two data files*		✓
COPY	Copy files between devices*		✓
DCLEAR	Clear all open channels on disk drives		✓
DCLOSE	Close logical disk file		✓
DIRECTORY	Display directory of contents of disk on screen*		✓
DLOAD	Load a BASIC program from disk		✓
DOPEN	Open a disk file for a read and/or write operation		✓
DSAVE	Save a BASIC program to disk		✓
DVERIFY	Verify program in memory against programs on disk		✓
GET #	Receive input from open disk file	✓	
HEADER	Format a disk*		✓
LOAD	Load a file from disk	✓	
OPEN	Open a file for input or output	✓	
PRINT#	Output a data to file	✓	

*Although there is no single equivalent command for this function in BASIC 2.0, there is an equivalent multi-command instruction. See your disk drive manual for these BASIC 2.0 conventions.

Command	Use	Basic 2.0	Basic 7.0
RECORD	Position relative file pointers*		✓
RENAME	Change name of a file on disk*		✓
RUN filename	Execute BASIC program from disk		✓
SAVE	Store program in memory to disk	✓	
VERIFY	Verify program in memory against program on disk	✓	

*Although there is no single equivalent command in BASIC 2.0, there is an equivalent multi-command instruction. See your disk drive manual for these BASIC 2.0 conventions.

GLOSSARY

This glossary provides brief definitions of frequently used computing terms.

Acoustic Coupler or Acoustic Modem: A device that converts digital signals to audible tones for transmission over telephone lines. Speed is limited to about 1,200 baud, or bits per second (bps). Compare direct-connect modem.

Address: The label or number identifying the register or memory location where a unit of information is stored.

Alphanumeric: Letters, numbers and special symbols found on the keyboard, excluding graphic characters.

ALU: Arithmetic Logic Unit. The part of a Central Processing Unit (CPU) where binary data is acted upon.

Animation: The use of computer instructions to simulate motion of an object on the screen through gradual, progressive movements.

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: Acronym for American Standard Code for Information Interchange. A seven-bit code used to represent alphanumeric characters. It is useful for such things as sending information from a keyboard to the computer, and from one computer to another. See Character String Code.

Assembler: A program that translates assembly-language instructions into machine-language instructions.

Assembly Language: A machine-oriented language in which mnemonics are used to represent each machine-language instruction. Each CPU has its own specific assembly language. See CPU and machine language.

Assignment Statement: A BASIC statement that sets a variable, constant or array element to a specific numeric or string value.

Asynchronous Transmission: A scheme in which data characters are sent at random time intervals. Limits phone-line transmission to about 2,400 baud (bps). See Synchronous Transmission.

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 the characters are placed upon.

BASIC: Acronym for Beginner's All-purpose Symbolic Instruction Code.

Baud: Serial-data transmission speed. Originally a telegraph term, 300 baud is approximately equal to a transmission speed of 30 bytes or characters per second.

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 Control: A means of transmitting serial data in which each bit has a significant meaning and a single character is surrounded with start and stop bits.

Bit Map Mode: An advanced graphic mode in the Commodore 128 in which you can control every dot on the screen.

Border Color: The color of the edges around the screen.

Branch: To jump to a section of a program and execute it. GOTO and GOSUB are examples of BASIC branch instructions.

Bubble Memory: A relatively new type of computer memory, it uses tiny magnetic "pockets" or "bubbles" to store data.

Burst Mode: A special high speed mode of communication between a disk drive and a computer, in which information is transmitted at many times normal speed.

Bus: Parallel or serial lines used to transfer signals between devices. Computers are often described by their bus structure (i.e., S-100-bus computers, etc.).

Bus Network: A system in which all stations or computer devices communicate by using a common distribution channel or bus.

Byte: A group of eight bits that make up the smallest unit of addressable storage in a computer. Each memory location in the Commodore 128 contains one byte of information. One byte is the unit of storage needed to represent one character in memory. See Bit.

Carrier Frequency: A constant signal transmitted between communicating devices that is modulated to encode binary information.

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 128's memory which stores the encoded character patterns that are displayed on the screen.

Character Set: A group of related characters. The Commodore 128 character sets consist of: upper-case letters, lower-case letters and graphic characters.

Character String Code: The numeric value assigned to represent a Commodore 128 character in the computer's memory.

Chip: A miniature electronic circuit that performs a computer operation such as graphics, sound and input/output.

Clock: The timing circuit for a microprocessor.

Clocking: A technique used to synchronize a sending and a receiving data-communications device that is modulated to encode binary information.

Coaxial Cable: A transmission medium, usually employed in local networks.

Collision Detection: A task performed in a multiple-access network to prevent two computers transmitting at the same time.

Color Memory: The area in the Commodore 128'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.

Compiler: A program that translates a high-level language, such as BASIC, into machine language.

Composite Monitor: A device used to provide a 40-column video display.

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 condition 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.

CPU: Acronym for Central Processing Unit. The part of the computer containing the circuits that control and perform the execution of computer instructions.

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.

Data Base: A large amount of data stored in a well-organized manner. A data-base management system is a program that allows access to the information.

Data Link Layer: A logical portion of data communications control that mainly ensures that communication between adjacent devices is error free.

Data Packet: A means of transmitting serial data in an efficient package that includes an error-checking sequence.

Data Rate or Data Transfer Rate: The speed at which data is sent to a receiving computer—given in baud, or bits per second (bps).

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.

Dedicated Line or Leased Line: A special telephone line arrangement supplied by the telephone company, and required by certain computers or terminals, whereby the connection is always established.

Delay Loop: An empty FOR . . . NEXT loop that slows the execution of a program.

Dial-Up Line: The normal switched telephone line that can be used as a transmission medium for data communications.

Digital: Of or relating to the technology of computers and data communications where all information is encoded as bits of 1s or 0s that represent on or off states.

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.

Direct Connect Modem: A device that converts digital signals from a computer into electronic impulses for transmission over telephone lines. Contrast with Acoustic Coupler.

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.

Disk Operating System: Program used to transfer information to and from a disk. Often referred to as a DOS.

Duration: The length of time a musical note is played.

Electronic Mail or E-Mail: A communications service for computer users where textual messages are sent to a central computer, or electronic "mail box," and later retrieved by the addressee.

Enable: To turn on a bit, byte or specific operation of the computer.

Envelope Generator: Portion of the Commodore 128 that produces specific waveforms (sawtooth, triangle, pulse width and noise) for musical notes. See Waveform.

EPROM: A PROM that can be erased by the user, usually by exposing it to ultraviolet light. See PROM.

Error Checking or Error Detection: Software routines that identify, and often correct, erroneous data.

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 treated as a unit and stored on disk or tape.

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.

Full-Duplex Mode: Allows two computers to transmit and receive data at the same time.

Function: A predefined operation that returns a single value.

Function Keys: The four keys on the far right of the Commodore 128 keyboard. Each key can be programmed to execute a series of instructions. Since the keys can be SHIFTed, you can create eight different sets of instructions.

GCR Format: Method of storing information on a disk; when used by 1541 and 1571, disk can read and write on GCR-formatted disks.

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.

Half-Duplex Mode: Allows transmission in only one direction at a time; if one device is sending, the other must simply receive data until it's time for it to transmit.

Hardware: Physical components in a computer system such as keyboard, disk drive and printer.

Hexadecimal: Refers to the base-16 number system. Machine language programs are often written in hexadecimal notation.

Home: The upper-left corner of the screen.

IC: Integrated Circuit. A silicon chip containing an electric circuit

made up of components such as transistors, diodes, resistors and capacitors. Integrated circuits are smaller, faster and more efficient than the individual circuits used in older computers.

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. Input sources include the keyboard, disk drive, Datassette or modem.

Integer: A whole number (i.e., a number containing no fractional part), such as 0, 1, 2, etc.

Interface: The point of meeting between a computer and an external entity, whether an operator, a peripheral device or a communications medium. An interface may be physical, involving a connector, or logical, involving software.

I/O: Input/output. Refers to the process of entering data into the computer, or transferring data from the computer to a disk drive, printer or storage medium.

Keyboard: Input component of a computer system.

Kilobyte (K): 1,024 bytes.

Local Network: One of several short-distance data communications schemes typified by common use of a transmission medium by many devices and high-data speeds. Also called a Local Area Network, or LAN.

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. Machine language is written in binary form that a computer can execute directly. Also called machine code or object code.

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 131,072 memory locations (0-131,071) in the Commodore 128.

MFM: A method of storing information on disks. Can be read by 1541 and 1571 disk drives, but these drives cannot write disks in this format.

Microprocessor: A CPU that is contained on a single integrated circuit (IC). Microprocessors used in Commodore personal computers include the 6510, the 8502 and the Z80.

Mode: A state of operation.

Modem: Acronym for MOdulator/DEModulator. A device that transforms digital signals from the computer into electrical impulses for transmission over telephone lines, and does the reverse for reception.

Monitor: A display device resembling a television set but with a higher-resolution (sharper) image on the video screen.

Motherboard: In a bus-oriented system, the board that contains the bus lines and edge connectors to accommodate the other boards in the system.

Multi-Color Character Mode: A graphic mode that allows you to display four different colors within an 8 × 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 × 8 character grid. See Pixel.

Multiple-Access Network: A flexible system by which every station can have access to the network at all times; provisions are made for times when two computers decide to transmit at the same time.

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 NOT, and XOR.

Order of Operations: Sequence in which computations are performed in a mathematical expression. Also called Heirarchy of Operations.

Parallel Port: A port used for transmission of data one byte at a time over multiple wires.

Parity Bit: A 1 or 0 added to a group of bits that identifies the sum of the bits as odd or even.

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 displaced within an 8 × 8 grid of pixels. The entire screen is composed of a 320 × 200 pixel grid. In bit-map mode, each pixel corresponds to one bit in the computer's memory.

Polling: A communications control method used by some computer/terminal systems whereby a "master" station asks many devices attached to a common transmission medium, in turn, whether they have information to send.

Pointer: A register used to indicate the address of a location in memory.

Port: A channel through which data is transferred to and from the CPU. An 8-bit CPU can address 256 ports.

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 128 is 160 characters.

PROM: Acronym for Programmable Read Only Memory. A semiconductor memory whose contents cannot be changed.

Protocol: The rules under which computers exchange information, including the organization of the units of data to be transferred.

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 RaNDom (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 128 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.

RGBI Monitor: Red/Green/Blue/Intensity. A high-resolution display device necessary to produce an 80-column screen format.

Ribbon Cable: A group of attached parallel wires.

Ring Network: A system in which all stations are linked to form a continuous loop or circle.

RS-232: A recommended standard for electronic and mechanical specifications of serial transmission ports. The Commodore 128 parallel user port can be treated as a serial port if accessed through software, sometimes with the addition of an interface device.

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 128's memory that contains the information displayed on the video screen.

Serial Port: A port used for serial transmission of data; bits are transmitted one bit after the other over a single wire.

Serial Transmission: The sending of sequentially ordered data bits.

Software: Computer programs (sets of instructions) stored on disk, tape or cartridge that can be loaded into random access memory. Software, in essence, tells the computer what to do.

Sound Interface Device (SID): The MOS 6581 sound synthesizer chip responsible for all the audio features of the Commodore 128. See the Commodore 128 Programmer's Reference Guide for chip specifications.

Source Code: A non-executable program written in a high-level language. A compiler or assembler must translate the source code into an object code (machine language) that the computer can understand.

Sprite: A programmable, movable, high-resolution graphic image. Also called a Movable Object Block (MOB).

Standard Character Mode: The mode the Commodore 128 operates in when you turn it on and when you write programs.

Start Bit: A bit or group of bits that identifies the beginning of a data word.

Statement: A BASIC instruction contained in a program line.

Stop Bit: A bit or group of bits that identifies the end of a data word and defines the space between data words.

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.

Synchronous Transmission: Data communications using a synchronizing, or clocking signal between sending and receiving devices.

Syntax: The grammatical rules of a programming language.

Tone: An audible sound of specific pitch and waveform.

Transparent: Describes a computer operation that does not require user intervention.

Variable: A unit of storage representing a changing string or numeric value. Variable names can be any length, but only the first two characters are stored by the Commodore 128. The first character must be a letter.

Video Interface Controller (VIC): The MOS 6566 chip responsible for the 40-column graphics features of the Commodore 128. See the Commodore 128 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 128 can produce three different sounds simultaneously. Each voice consists of a tone oscillar/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.

Word: Number of bits treated as a single unit by the CPU. In an eight-bit machine, the word length is eight bits; in a 16-bit machine, the word length is 16 bits.

INDEX

A

Abbreviations—BASIC, 29, 33, 379
ABS function, 70, 305
Addition, 36
ADM 3, 223
ADSR, 131, 142
Alt key, 91
Alt mode, 223
Animation, 109, 122
APPEND, 235
Arrays, 61, 62, 325
ASC function, 69, 305
ASCII character codes, 69, 355
ASM, 199
Asterisk key (*), 36, 198
Attack, 142
ATN function, 305
AUTO command, 81, 235
AUXIN, 215
AUXOUT, 215

B

Bach, 156
BACKUP, 236
Bandpass, 151
BANK, 236
Bank table, 237
BAS, 99
BASIC
- abbreviations, 29
- commands, 233
- functions, 67, 303
- mathematics, 36
- operators, 36
- statements, 233
- variables, 325

BASIC 2.0, 11, 229
BASIC 7.0, 5, 229
BEGIN:/:BEND, 77, 237
Binary files, 124
Bit Map mode, 98
BLOAD, 119, 124, 238
BOOT, 239
Booting, 188
BOX, 96, 102, 239
BSAVE, 119, 124, 127, 240
BUMP, 306

C

C128 Mode, 10
C64 Mode, 10
Caps Lock key, 91
Cartridge Port, 349
Cartridges, 12
Cassette Port, 350
CATALOG, 241
Channel selector, 351
CHAR, 96, 104, 241
Character sets, 21
Character string code, 69
CHR$ codes, 174, 355, 365
CHR$ function, 69, 306
CIRCLE, 96, 101, 242
Clock, 326
CLOSE statement, 179, 244
CLR, 41, 100, 244
CLR/HOME key, 26
CMD, 245
COLLECT, 245
COLLISION, 245
Colon (:), 52
COLOR, 96, 97, 246
Color
- code display chart, 30, 97, 98, 247
- control, 25, 34, 223
- CHR$ codes, 355, 356
- keys, 30
- memory map, 358
- screen and border, 99
- source codes, 97

COM, 190, 199
Comma (,), 28
Command, 19
Command keys, 21
Command keyword, 190
Command line, 190
Command tail, 190
Commodore key, 25, 26
Composite monitor, 164
CONCAT, 247
CONIN:, 215
Connections, 347
CONOUT:, 215
Constants, 38
CONTinue command, 71, 248
Control characters table, 147
Control key, 25, 187

Coordinate grid, 101
COPY, 248
Copying music, 156
Copying programs, 27, 200
COPYSYS, 200, 213
COSine function, 306
CP/M characters, 199
CP/M mode, 187
CP/M Plus User's Guide, 224
CP/M Plus 3.0, 187
CTRL-, 190, 206
CuRSoR keys, 22, 173
Cursor, 21
Customer Support,
Cutoff frequency, 150

D

Datassette, 41
DATA, 59, 249
Data file, 195
DATE, 213
DCLEAR, 249
DCLOSE, 250
Debug, 70, 86
DEC, 307
Decay, 142
DEF FN, 250
Delay loop, 54
DELETE, 82, 250
DELete key, 24
DEVICE, 213, 215
Dice, 68
DIMension statement, 61, 251
DIR command, 191, 212, 213
Direct mode, 19
DIRECTORY, 46, 252
DIRSYS, 212
Disk commands, 43, 179, 383
Disk directory, 42, 46, 181
Disk Parameters, 42, 189
Division, 36
DLOAD", 19, 45, 253
Dollar sign ($), 40, 149, 181, 332
DO/LOOP, 75, 253
DOPEN, 254
DRAW, 96, 102, 255
Drive specifier, 196
DS/DS$ variables, 326
DSAVE", 19, 44, 256
Dual screens, 167
DUMP, 213
Duration, 133, 146
DVERIFY", 46, 256

E

Echo, 205
ED, 195, 213
Editing, 35, 205
EL variable, 326
ELSE clause, 77, 266
END statement, 51, 257
Envelope generator, 142
ENVELOPE, 142, 257
Equals (=), 39, 52
ERASE, 212, 213
ER/ERR$ variables, 85, 307, 326
Error functions, 85
Error messages, 335, 341
EU variable, 85
Escape codes, 368
ESCape key, 87, 100, 165, 223
EXIT, 76, 253
Exponentiation, 37
EXPonent function, 307

F

40/80 Display key, 91, 163
FAST command, 89, 258
Features, 9
FETCH, 258
File, 195
Filename, 196
File specifications, 195
File type, 196
FILE NOT FOUND, 45
FILTER, 152, 258
Filter—SID, 149
Flashing cursor, 88
FN function, 307
FOR . . . NEXT statement, 53, 259
FORMAT, 213
Formatting disks, 42, 179
FRE function, 308
Frequency, 133, 140
Function, 19
Function keys, 27, 89, 174

G

Game controls and ports, 348
GET, 57, 213, 214, 260
GETKEY, 80, 261
GET# statement, 261
GO64, 262
GOSUB, 64, 262
GOTO, 33, 263
GRAPHIC, 96, 99, 263
Graphic characters, 27
Graphic modes, 89, 99
GSHAPE, 295

H

Harmonics, 140
Hash mark (#), 78, 113, 149
HEADER, 42, 264
HELP, 83, 90, 213, 216, 265
HELP key, 90
HEX, 199
HEX$, 308
HLP, 199
HOME key, 26
Hyperbolic functions, 361

I

IF... THEN statement, 51, 266
INITDIR, 213
Initializing, 181
INPUT, 55, 267
INPUT#, 267
Input Prompt, 56
INSerT key, 24
INSTR, 308
INTeger function, 67, 309

J

JOY, 309
Joystick ports, 348

K

KEY command, 90, 268
Keyboard, 20
Key assignment—CP/M, 222

L

LEFT$ function, 310
LENgth function, 310
LET statement, 269
Light pen, 12
Line Feed key, 92
Line numbers, 31
LIST command, 32, 269
LOAD command, 45, 180, 270
LOADing cassette software, 180
LOADing CP/M software, 188
LOADing disk software, 180
LOCATE, 271
LOGarithm function, 310
Loops, 53
LST, 215

M

Machine language, 369
Mathematics, 36, 361
Memory maps, 357, 363
MID$ function, 310
Mode switching chart, 13
MONITOR, 272
Monitor—dual 1902, 14, 165
Monitor—machine language, 90, 369
Monitor switching, 101, 166
MOVSPR, 113, 272
Multicolor bit mode, 98
Multiplication, 36
Music programs, 153, 158
Music videos, 155
Musical notes, 146
Musical instruments, 144
Musical scale, 156

N

Nested loops, 54
NEW, 34, 273
NEXT statement, 53, 259
Noise, 141
No Scroll key, 91, 205
Notch Reject Filter, 155
Notes, 146
Numeric functions, 67

O

Object code file, 124
ON GOTO/GOSUB, 65
OPEN statement, 179, 274
Operating System, 187
Operators
  arithmetic, 36, 327
  logical, 327
  order of, 37
  relational, 52, 327

P

PAINT, 96, 103, 275
Parentheses, 38, 199
Password, 197
PEEK function, 66, 311
PEN, 311
PERFECT series software, 5
Period (.), 149
PI, 332
PIP, 195, 201, 213
Pixel, 98, 111
PLAY, 145, 276
POINTER, 312
POKE, 66, 278
POS function, 312
POT, 313
Pound symbol (#)—see hash mark
PRINT, 28, 278
PRINT USING, 78, 280
PRINT#, 179, 279
Printer control—CP/M, 205
PRN, 199
Program file, 195
Program mode, 19
Programmable keys, 89, 174
Programming aids, 81
Programmer's Reference Guide, 6
PUDEF, 79, 283
Pulse width, 136, 143
PUT, 213, 214

Q

Question mark (?), 29, 198
Quotation marks ("), 29
Quote mode, 31

R

RAM, 65, 188
Random sounds, 138
RCLR, 313
RDOT, 314
READ, 59, 283
RECORD, 284
Relational operators, 52
REL, 199
Release, 142
REMark statement, 28, 285
RENAME, 212, 213, 285
RENUMBER, 81, 286
Reset button, 163
Reserved variables, 326
Rest, 146
Restore key, 26
RESTORE statement, 60, 286
RESUME command, 84, 287
Return key, 21
RETURN statement, 64, 288
RGBI monitor, 165
RGBI port, 164, 352
RGR, 314
RIGHT$ function, 314
RND function, 68, 138, 315
RSPCOLOR, 315
RSPRITE, 316
RUN command, 32, 288
RUN/STOP key, 25, 100, 138, 146, 163
RWINDOW, 317

S

SAVE command, 44, 178, 289
Saving programs on tape, 180
Saving programs on disk, 178
Sawtooth waveform, 141
SCALE, 96, 104, 290
SCRATCH command, 291
SCNCLR command, 89, 290
Screen display codes, 353
Screen display, 98, 163
Screen memory map, 357
Scrolling, 88, 187
Sector, 42
Semicolon (;), 29
Serial port, 350

SET, 213
SETDEF, 205, 213
SGN function, 318
SSHAPE, 96, 111, 295
Sharps (#), 149
Sheet music, 156
Shift key, 22
SHOW, 213
SID chip, 131
SINe function, 318
Slash key (/), 36
SLEEP, 78, 291
SLOW command, 89, 291
Software—80 column, 165
SOUND, 133, 292
Sound Interface Device, 131
Sound Player Program, 137
Sound reset, 138, 146
SPC function, 318
Split screen display, 98
SPRCOLOR, 293
SPRDEF, 96, 116, 293
SPRITE, 96, 112, 294
Sprite Combinations, 120
Sprite control, 112
Sprite editor, 117
Sprite programming, 109, 115
Sprite memory map, 127
Sprite movement, 113
Sprite viewing area, 114
Sprites, 108
SPRSAV, 96, 112, 295
SQR function, 67, 70, 319
ST variable, 326
STASH, 297
Statement, 19, 31
STEP, 54, 259
STOP, 70, 297
STOP key, 25
Storing programs, 41, 177
String functions, 67
Strings, 29, 40
STR$ function, 70, 319
SUB, 199
SUBMIT, 213
Subroutine, 64
Subscripts, 61
Subtraction, 36
Sustain, 142
SWAP, 297
Sweep, 134
Syntax, 19
Syntax error, 22
Synthesizer, 131, 147
SYM, 199
SYS, 199, 297
System prompt, 189

T

Tab key, 92
TAB function, 319
TAB stops, 88
TANgent function, 320
TEMPO, 145, 298
Terminating CP/M, 216
THEN, 51, 266
Timbre, 140
Time delay, 54
TI/TI$ variables, 326
TO, 102, 259
Track, 42
Transient Utility commands, 190, 211
TRAP, 83, 298
Triangle waveform, 141
Trigonometric functions, 361
TRON/TROFF, 85, 299
TYPE command, 212, 213
Typing rules, 27

U

UNTIL statement, 75, 253
Up arrow (↑) key, 37
Upper case/graphic set, 21, 173
Upper/Lower case set, 21, 173
USER, 197, 212
User Number, 197
User port, 352
USR function, 320

V

VALue function, 70, 320
Variables, 39, 61, 325
VERIFY command, 46, 181, 299
VIC chip, 95

Video Ports, 164, 351
Voice, 131
VOLume, 134, 145, 300

W

WAIT command, 300
Waveform, 131, 141, 143
WHILE statement, 76, 253
Wildcard, 198
WINDOW command, 86, 301
Windowing, 86

X

XOR, 321

Z

Z80 Microprocessor, 187

NOTES

NOTES

TEXT by:
LARRY GREENLEY
NORM MCVEY
STEVE FINKEL
MAX SPOLOWICH
ADAM TAIT
JOYCE WETMORE

DESIGN by:
WILSON HARP
JO-ELLEN TEMPLE

SPECIAL ACKNOWLEDGEMENT

The manifold contributions of the Commodore ENGINEERING, QUALITY ASSURANCE, and MARKETING groups to the C128 System Guide are gratefully acknowledged. These contributions, which included but were not limited to technical guidance and the review of voluminous draft materials and galleys, were indispensable to the production of this book. Without the contributions of these groups there would be no C128 System Guide—and indeed, no Commodore 128 Personal Computer.



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 . 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 _____

METHOD OF PAYMENT

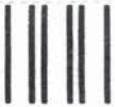
- ☐ Enclosed is my check or money order for $_____ *(Make check or money order payable to COMMODORE PUBLICATIONS)*
- ☐ Bill me
- ☐ Charge my VISA or MasterCard *(circle one)* Card number

☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

or call 800-345-8112 to order (in Penna. 800-662-2444)

Signature _____ Expiration Date _____

IBC128



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



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

COMMODORE

Commodore Business Machines, Inc.
1200 Wilson Drive • West Chester, PA 19380
Commodore Business Machines, Limited
3370 Pharmacy Avenue • Agincourt, Ontario, M1W 2K4

310638-01

Printed in USA