B SERIES

COMMODORE

USER GUIDE

COMMODORE 'B' Series ADVANCED BUSINESS COMPUTERS User's Guide

User's Guide Statement

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

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

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

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

First Edition—1983
First Printing—1983

Printed in the United States of America

TABLE OF CONTENTS

4. SOFTWARE

5. USING YOUR DISK DRIVE

CHAPTER 1

INTRODUCTION

- Features overview
- Organization of the Manual
- How to Use This Guide

You can design the business computer system that best meets your needs by choosing one of the Advanced Business Computers in Commodore's 'B' Series:

- The B-128-80
- The B-256-80
- The BX-128-80
- The BX-256-80
- The CBM-128-80
- The CBM-256-80
- The CBMX-128-80
- The CBMX-256-80

The computer is only one part of your business computer **system.** Your system should also include a high capacity dual floppy or hard disk drive . and a dot matrix or letter-quality printer. Networking and telecommunications accessories help extend your system to include multiple computers. even in different sites. Your Commodore business dealer can tell you more about these peripherals.

Software is also important to your business system—word processing. electronic spread sheets. accounting. record keeping—these are just a few of the many practical functions good business software can provide. expecially if it's easy-to-use and "friendly" like the business programs licensed and developed by Commodore for your 'B' Series computer system.

FEATURES OVERVIEW

The computers in the 'B' Series have many common features. and you can add enhancements to the lower end systems to give them the extra capabilities that are standard on our more sophisticated systems. The following features are common to all 'B' Series computers:

- 80 column by 25 line screen display
- Separate calculator keypad for quick computations
- 10 predefined function keys
- Total of 20 easy-to-define function keys
- Extended BASIC version 4.0+
- Expandable memory
- IEEE-488 bus
- RS-232C interface
- 6509 microprocessor
- Direct audio output

These features distinguish the models:

- Amount of memory (128K or 256K)
- Monochrome tilt and swivel monitor built in (CBM models)
- Dual microprocessors, with the 16-bit 8088 microprocessor built in (indicated by the X in the name)

The following table shows which features are offered by various 'B' Series computers:

Model	Memory (RAM)	Standard Microprocessors	Optional Microprocessors	Built-in Monitor
B-128-80	128	6509	Z80, 8088	NO
B-256-80	256	6509	Z80, 8088	NO
BX-128-80	128	6509, 8088	Z80*	NO
BX-256-80	256	6509, 8088	Z80*	NO
CBM-128-80	128	6509	Z80, 8088	YES
CBM-256-80	256	6509	Z80, 8088	YES
CBMX-128-80	128	6509, 8088	Z80*	YES
CBMX-256-80	256	6509, 8088	Z80*	YES

All of these models will not necessarily be offered for sale in your area.

NOTE: Your Commodore dealer can install a Z80 microprocessor in a machine that has the 8088 microprocessor built in. Only one at a time of these two microprocessors can be present in your 'B' system.

You can customize your system by adding the variety of easy-to-install peripherals and additional microprocessors that are available for the 'B' Series computers. These peripherals include Commodore's Floppy Disk Drives and Hard Disk Drives, a variety of printers for letter-quality printing or fast printing, modems for telecomputing, monochrome monitors for the B and BX machines, and other devices that make your computer the ideal business assistant.

The microprocessors you can add to your computer include Commodore's Z80 microprocessor, which gives you access to CP/M* software. If your 'B' Series computer doesn't have the 8088 microprocessor built-in, you can add it yourself to gain access to MS-DOS**, CP/M-86, and Concurrent CP/M-86*** software.

---

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

** MS-DOS is a trademark of Microsoft, Inc.

*** CP/M-86 and Concurrent CP/M-86 are trademarks of Digital Research, Inc.

Chapter 4. Software, explains the capabilities of these useful microprocessors.

The new 'B' Series computers give you state-of-the-art computer capabilities at an affordable price. Commodore is committed to providing you with hardware and software that meet your needs. See your Commodore dealer for more information about Commodore's peripherals and software packages.

ORGANIZATION OF THE MANUAL

This User's Guide introduces you to the 'B' Series of Advanced Business Computers. The manual begins by showing you how to set up your computer and by describing optional equipment that expands your computer's uses. The next chapters explain how to use the keyboard, and how to load and save programs. You'll also find descriptions of BASIC commands, statements, and functions, and some information about software available for the 'B' Series.

Chapter 1

INTRODUCTION describes Commodore's 'B' Series of Advanced Business Computers and presents the different features of each machine. The introduction also shows how to use this manual.

Chapter 2

SETTING UP THE COMPUTER contains the instructions you need to unpack, connect, and install your 'B' Series computer. The CBM / CBMX systems, which include built-in monitors, and the B /BX systems, which do not include built-in monitors, are described in separate sections. Chapter 2 also describes the variety of configurations and optional equipment (peripherals) available for your computer. This chapter also contains a few trouble shooting and diagnostic procedures that can help you make adjustments to solve minor problems that may appear after you've installed your computer system.

Chapter 3

USING THE KEYBOARD describes how to use the keys on your computer's keyboard. Special keys, including the programmable function keys, are explained in detail.

Chapter 4

SOFTWARE describes how you can enhance your computer system with software systems that give you access to a variety of business, scientific, and educational software.

Chapter 5

USING YOUR DISK DRIVE tells you how to load and save both prepackaged software and your own custom designed programs. This chapter also explains how to prepare new disks and how to copy old ones. For additional details on the Disk Operating Systems, consult the manuals that come with your Floppy Disk Drive or the fast and powerful Commodore Hard Disk Drive.

Chapter 6

EXTENDED BASIC 4.0+ COMMANDS AND STATEMENTS are briefly explained. Complete formats and examples are provided.

APPENDICES include quick reference information about the major technical features that programmers and many users need. For an additional in-depth presentation of technical material, consult the Advanced Business Computers Programmer's Reference Guide.

HOW TO USE THIS GUIDE:

Special Considerations

1. As you look at the edge of each page you will notice that there is what we call an "inset tab." The "inset tab" shows you exactly where the seven chapters are located. Note that the beginning of each chapter is a solid blue page. Both of these features make it easy for you to get to the information you need quickly.

2. To help you unpack, hook up, set up, and begin operating your computer, Chapter 2 contains many detailed illustrations that can make the installation of your equipment, with all its options, a quick and easy task.

3. When we discuss a specific key, or want you to press a particular key, we show you a visual cue (*Example:* **RETURN** means press the RETURN key.)

4. **Please note that this manual is not designed to teach the computer language BASIC** (the primary language used in all Commodore computers). If you want to learn BASIC language programming techniques, or any of the other languages available for use with your computer(s): we suggest that you **consult the "Bibliography" (Appendix P) for books that teach programming.**

CHAPTER 2

SETTING UP THE COMPUTER

- Unpacking and Packing
- Installation
- Hookup and Configurations Available
- Expanding Your System with Peripherals
- Additional Microprocessors
- Trouble Shooting

Unpacking/Packing

B and BX Computers

The B and BX systems are shipped in one part. The package also contains a video cable (5-pin DIN or RCA phone-type cable) and an AC power cord (120 volts).

CBM and CBMX Computer

The CBM and CBMX systems are shipped in two parts:

1. Base and video display screen.
2. Keyboard with attached telephone-type cable that plugs into the base.

The package also contains an AC power cable (120 volts).

NOTE: Never try to remove or disconnect the video display screen from the computer base. If the screen must be removed, take the entire unit back to your dealer.

Installation

CBM and CBMX Models

1. Make sure that your computer is turned *off* before starting installation. The CBM and CBMX computers have their power switch located in the **back** of the machine on the **left** hand side.
2. Plug the 25 PIN CABLE attached to the keyboard into the connector on the **lower right** hand side of the base/video display unit. *Make sure that the Commodore Logo* C= *is facing up.*
3. Plug the 3-prong AC power cord into the power cord jack located in the **back** of the base/video display unit, on the **left** hand side. The power cord fits only one way.
4. Plug the 3-prong AC power cord into a standard wall outlet.

Fig. 2-1 'B' Series CBM and CBMX (Front view)

Fig. 2-2. 'B' Series B and BX (Front view)

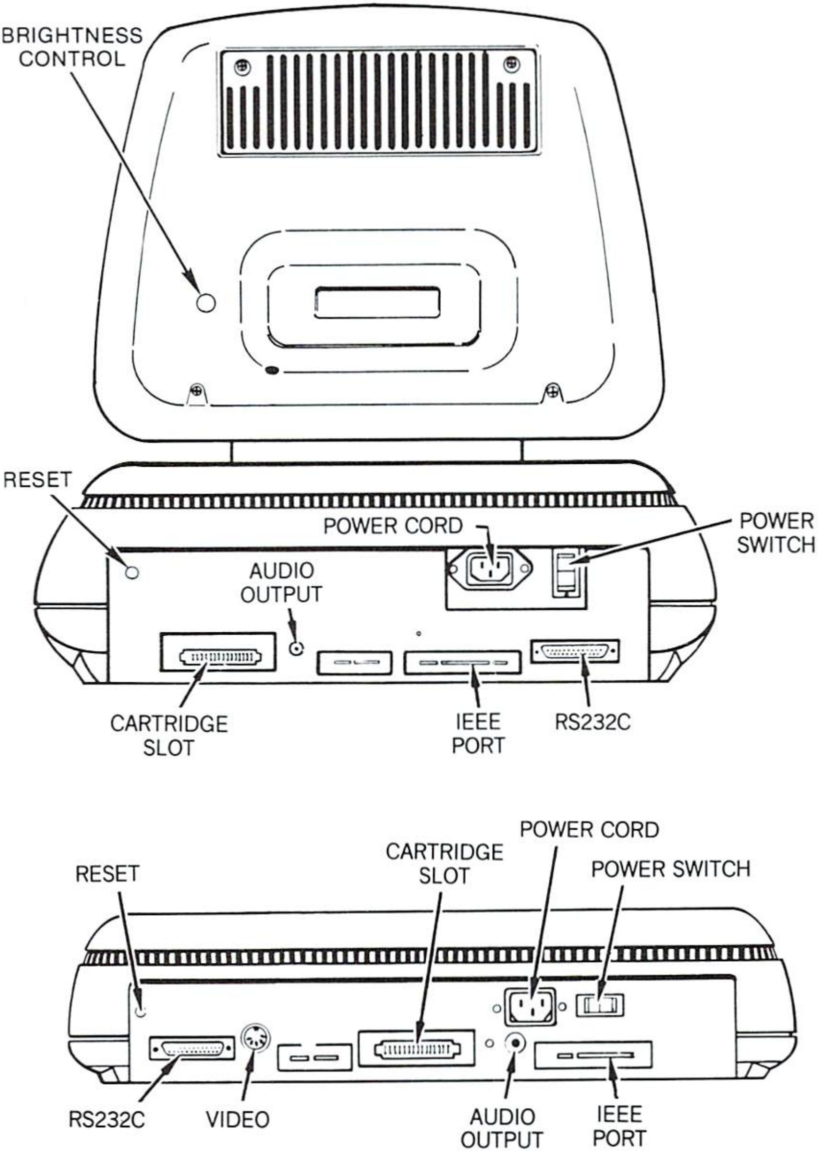


Fig. 2-3 'B' Series (Rear view)

B and BX Models

Connect the computer to your monochrome monitor as described below. See Fig. 2-3 to locate each input and output on the back of your B or BX computer.

1. Make sure that your computer and your monitor are turned *off* before starting installation. The B and BX computers have their power switch located in the **back** of the machine on the **left** hand side.
2. Attach the video cable to the computer at the connector labeled audio /video (5 pin DIN). Line up the pins with the corresponding holes and push the connector in. The cable will only go in one way.
3. Attach the two RCA phone-type jacks to your video monitor inputs. See the monitor's manual for instructions.
4. Plug the computer AC power cord into a 120 volt. 60 Hz AC outlet.

Your B or BX computer should now be connected properly. No additional connections are required to use the computer with your monitor.

NOTE: Save the packing materials that your computer came in. Then, to pack your equipment back in the box for storage or transit, reverse the procedures described above.

Expanding Your System with Peripherals

Printers

A full range of printers is available. designed to match any need. Low cost. high speed dot matrix units such as Commodore's 8023 Tractor Printer are ideal for most applications. Where letter quality printing is required. the Commodore "daisy-wheel" printer produces the best results.

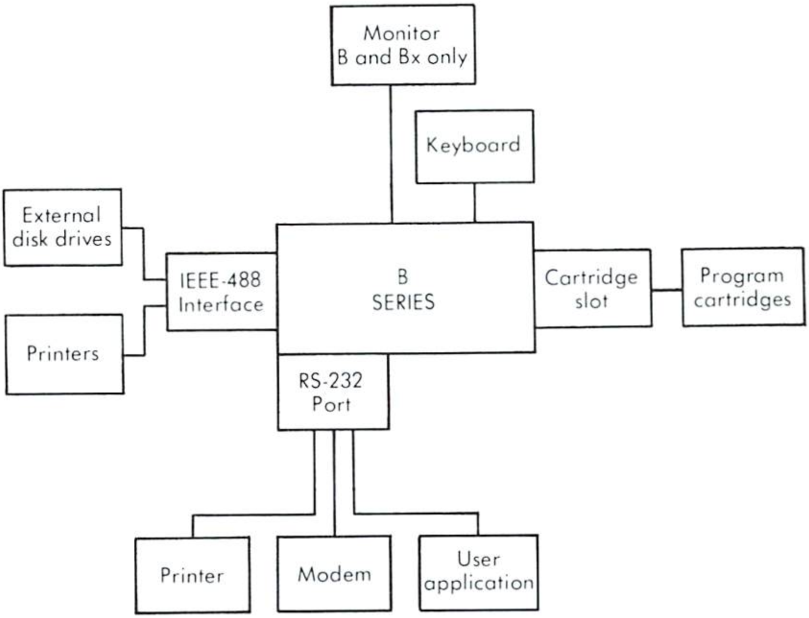


Fig. 2-4 Hook-up and Configuration Available. Accessories and peripherals connect to the expansion ports as shown.

External Disk Drive Units

Single or dual floppy disk units, with storage capacity from 170,000 characters to over 2 million characters, can be easily attached to store programs and data. Commodore's dual floppy disk drives include the 8050 and the 4040. Hard disk units with capacities of 5 and 7.5 million characters can also be used with equal ease. See your Commodore dealer for a complete list of available disk drives.

RS-232C Port

Your computer comes equipped with an industry-standard RS-232C serial interface. This interface provides you with access to a wide variety of peripherals, such as printers, terminals, modems, and data collection equipment.

The RS-232C interface is implemented using the fully programmable 6551 Asynchronous Communications Interface Adapter. With the 6551, you can program your RS-232C interface to match exactly the requirements of the device you're connecting to it.

The Extended BASIC 4.0 interpreter includes file level software support for the RS-232C interface channel. Open the RS-232C channel as you would any other file and access it with standard BASIC input/output statements. The RS-232C Port is device #2. See Appendix N.

CBM IEEE Port

Your advanced computer supports the full range of Commodore CBM peripherals via the built-in IEEE-488 interface. Most disk units are "intelligent," meaning they have their own microprocessor and memory. You can connect up to five disk drives at one time to your computer by "daisy chaining" them together through the IEEE-488 connector port.

NOTE: The device numbers that are used with the IEEE port must be within the range of 4 to 31 inclusive.

Additional Microprocessors and Operating Systems:

Special Options to Increase Your Computer's Power

Each computer in the 'B' Series uses the 6509 microprocessor, which was developed by Commodore's MOS Technology subsidiary. Commodore has designed the 'B' Series computers to be easily expanded to dual processor computers with the addition of the 16-bit 8088 microprocessor or the Z-80 microprocessor. These additional microprocessors give you access to hundreds of software packages that are independently developed for use with the 8088 and Z-80 microprocessors.

In some 'B' series models, the 8088 microprocessor is built-in:

in the others, it can be added. In addition, you can add the Z-80 microprocessor to any 'B' Series computer.

The 16-Bit 8088 microprocessor: MS-DOS and Concurrent CP/M-86

The 8088 microprocessor gives you access to two operating systems that let you increase the software applications available for your 'B' Series computer. These operating systems, MS-DOS and Concurrent CP/M-86, offer a variety of business and personal software programs.

The 8088 microprocessor is built into four models of the 'B' Series of advanced business computers. The presence of the built-in 8088 microprocessor is indicated by the X in the 'B' Series model name (the BX-128-80, the BX-256-80, the CBMX-128-80, and the CBMX-256-80). These machines are dual processor computers.

You can upgrade the B-128-80, the B-256-80, the CBM-128-80, and the CBM-256-80 by adding the 8088 microprocessor. If you have one of these systems, your Commodore dealer can install the 8088 microprocessor for you.

The Z-80 Microprocessor and CP/M® Operating System

The Z-80 microprocessor and CP/M Operating System give you access to a variety of CP/M software applications that you can use on your 'B' Series computer. These applications include:

- widely used business programs, such as CALCSTAR
- word processing programs, such as WORDSTAR
- database programs, such as INFOSTAR
- mailing list programs, such as MAILMERGE
- many other specialized software programs, such as high level computer language compilers

The CP/M Operating System User's Guide that comes with the Z-80 and CP/M package explains how to operate this system.

The Z-80 microprocessor can be installed by your Commodore dealer. If your 'B' Series computer already has an 8088 microprocessor, your Commodore dealer can remove the 8088 and replace it with the Z-80 microprocessor. This installation is reversible: your Commodore dealer can switch the Z-80 and 8088 microprocessors repeatedly. This is possible for every 'B' Series model, including those with the 8088 microprocessor built in.

TROUBLE SHOOTING

PROBLEM	CAUSE	SOLUTION
1. Power indicator light not on	Computer not on	Make sure power switch is in the *On* position
	Power cable not pugged in	Check power socket for loose or disconnected power cable
	Power supply not plugged in	Check connection with wall outlet
	Bad fuse in Computer	Take system to authorized dealer for replacement of fuse
2. No picture on video screen	Incorrect hookup	Computer connects to video and audio inputs on video monitor.
	Video cable not plugged in	Check monitor output cable connection
3. Random pattern on monitor with cartridge in place	Cartridge not inserted properly	*Turn power off* and then reinstert the cartridge

TROUBLE SHOOTING (cont'd.)

PROBLEM	CAUSE	SOLUTION
4. Picture with excess audio background noise	Monitor volume setting is up too high	Lower the monitor's audio volume control
5. Picture is OK but you don't have sound	Monitor volume setting is down too low	Adjust the audio volume control
	Auxillary output not properly connected	Connect the sound jack to the auxillary input on your amplifier and then turn the amp. selector switch to the "Aux." position
6. The picture is too dark or too light	Brightness level is set incorrectly	Adjust the brightness control level on your monitor or built-in video display screen
7. Characters on the screen are hard to read	Contrast ratio between characters and background is too great or too small	Adjust the contrast control on your monitor. or built-in Video Display Screen

CHAPTER 3

USING THE KEYBOARD

- Format Keys
- Editing Keys
- Programmable Function Keys
- Calculator Pad Keys

The 'B' Series 96-key business-style keyboard makes a variety of business applications easy to use. The keyboard resembles a typewriter keyboard, but the computer has additional keys that control special functions. You should be familiar with these special keys before you begin using the computer.

Format Keys

RETURN and ENTER

The RETURN and ENTER keys tell the computer to look at what you've keyed in and put that information into memory. These keys, which have identical functions, also move the cursor to the next line.

When you key in a calculation in direct mode (i.e., not in a program), the solution is immediately displayed when you press either RETURN or ENTER.

NORM/GRAPH

This key lets you switch between the standard character set on your keyboard and the GRAPHICS mode. When you enter the graphics mode by pressing the NORM/GRAPH key, your keyboard's operations undergo the following changes:

- The keys print uppercase letters only. The SHIFT key is not used.
- The SHIFT key lets you print the graphics characters on the fronts of the keys.

Press the SHIFTed NORM/GRAPH key to return to the standard character set of upper and lower case letters. You can't use the graphics characters in this NORMAL mode.

SHIFT

This key works like the SHIFT key on a regular typewriter: it lets you print uppercase letters or the top characters on double character keys. When you are in the NORMAL mode, the standard alphabet of lower and uppercase characters is displayed, and the SHIFT key gets the uppercase characters.

When you are in the GRAPHICS mode, however, the alphabet appears in only uppercase, and the **SHIFT** key gets the graphics characters on the fronts of the keys.

The **SHIFT** key also lets you use an extra set of ten function keys. The **SHIFT**ed function key is ten more than the function key you press. For example, **SHIFT** and **F3** activate function key 13.

OFF/RVS

This key lets you display the REVERSED image of all the characters available on the keyboard. In other words, characters appear on the screen as black on green rather than the usual green on black (your monitor's characters may be a color other than green). When you press the **OFF/RVS** key, all characters you key in appear in reverse. Press the **OFF/RVS** key and the **SHIFT** key to turn off the reverse image display.

Editing Keys

The editing keys let you correct errors easily, move information around on the screen, and place the cursor wherever you want it.

Cursor Control Keys: ↑ ↓ ← →

The cursor is the small rectangle that marks your place on the screen. The four cursor control keys let you move the cursor wherever you want it.

The arrows on the keys show how they move the cursor:

↓ Moves the cursor DOWN.
↑ Moves the cursor UP.
← Moves the cursor LEFT.
→ Moves the cursor RIGHT.

The cursor has a repeat feature that lets it continue to move as long as you hold down the cursor key.

INS/DEL

The **DEL**ete key moves the cursor a space to the left, erasing the previous character you typed. If you're in the middle of a line, the character to the left is deleted and the characters to the right automatically move together to close up the space.

You can insert characters in the middle of text by pressing both the **SHIFT** and the **INS/DEL** keys. To use the insert function, use the cursor control keys to move the cursor to the character immediately to the right of where you want to insert. Hold down the **SHIFT** and **INS/DEL** keys until there is enough space to add missing information.

Like the cursor control keys, the **INS/DEL** key has a repeat feature that lets it continue to work as long as you hold down the key.

CLR/HOME

HOME moves the cursor back to the upper left corner of the screen. This is called the *HOME* position.

You can move the cursor to *HOME* and clear the screen by pressing **SHIFT** and **CLR/HOME**.

Programmable Function Keys

F1 F2 F3 F4 F5 F6 F7 F8 F9 F10

The ten keys on the upper left side of the keyboard are function keys that let you perform a variety of repetitive tasks such as clearing the screen, printing a message, or pausing a program. The keys **F1** through **F10** are predefined:

Key 1, "print"
Key 2, "list"
Key 3, "dload" + chr$ (34)
Key 4, "dsave" + chr$ (34)
Key 5, "dopen"

Key 6, "dclose"
Key 7, "copy"
Key 8, "directory"
Key 9, "scratch"
Key 10, "chr$("

You can display this list by keying in: KEY

In addition, there are ten more function keys available that are not predefined. Keys 11 through 20 are not marked on the key board, but you can use them by pressing the **SHIFT** key while you press one of the function keys. The **SHIFT**ed function key is ten more than the number on the key you press. For example, **SHIFT** and **F5** activate function key 15.

You can redefine these function keys with a simple procedure: just follow this format:

KEY n,["]*definition*[. . . +["]*definition*["]] ["]

1. You must enter the word KEY.
2. *n* is the number for the function key you want to program (1 through 20). Be sure to include the comma.
3. *definition* defines what you want the function key to do.

Here are some examples:

KEY 9, CHR$ (142)	Automatically switches to graphics mode.
KEY 15,"CHR$(14) + CHR$(13) + CHR$(77)"	Automatically switches to normal mode, advances to next line, and prints M.
KEY 1, "OPEN4,4:CMD4:LIST: CLOSE4" + CHR$(13)	Lists program on printer.

Function keys retain your definition only during the current session at the computer. Once you turn the computer off, your definitions are lost for all function keys, unless you save these definitions in a program.

CTRL

The **CTRL** key lets you print the graphics characters on the fronts of the non-alphabetic keys (e.g., number keys, punctuation keys, etc.) These graphics keys are displayed in either GRAPHICS or NORMAL mode. The alphabetic keys' graphics are displayed by using the **SHIFT** key in GRAPHICS mode only.

The **CTRL** key lets you use special control functions. To use a control function, hold down the **CTRL** key while you press the key that gives you the function you want. Here are some examples:

SCREEN EDITOR FUNCTION	CODE	MEANING
Set Window Bottom	CTRL-B	Set the bottom right boundary for the screen window to the current cursor position
Delete Line	CTRL-D	Delete the current line and scroll up all lines below the current line to replace the current line. Blank lines are inserted at the bottom of the screen. The cursor is positioned at the beginning of the new line.
Bell	CTRL-G	Toggle the end of line bell. IF the bell rings, then the end of line bell is enabled. If not, the end of line bell is disabled.
Insert Line	CTRL-I	Insert a blank line *after* the line on which the cursor is currently positioned. Place the cursor at the beginning of the blank line.
Erase To Line End	CTRL-P	Delete all text from the cursor position to the end of the line. Replace all deleted text with blank characters (spaces).
Erase From Line Start	CTRL-Q	Delete all text from the beginning of the line to the cursor position. Replace all deleted text with blank characters (spaces).
Set Window Top	CTRL-T	Set the top left boundary for the screen window to the current cursor position.

ESC

This key is used in conjunction with any of the 26 alphabet keys **A** through **Z** to perform a variety of special functions. To perform any function listed below, press the **ESC** key, release it, and press the appropriate letter.

A	Automatic insert
***B**	Set bottom right corner of window

C	Cancel automatic insert
D	Delete line
E	Nonflashing cursor
F	Flashing cursor
G	Enable (turn on) bell
H	Disable (turn off) bell
I	Insert line
J	Move to the start of the line
K	Move to the end of the line
L	Enable scrolling
M	Disable scrolling
N	Normal screen
O	Cancel insert, quote and reverse modes
P	Erase to the start of the line
Q	Erase to the end of the line
R	Reverse screen
S	Solid cursor
*T	Set the top left corner of window
U	Underscore cursor
V	Scroll up
W	Scroll down
X	Cancel escape sequence
Y	Normal character set
Z	Alternate character set (not currently implemented)

Quote and Insert Modes

When you press the quote key (") once, you enter QUOTE MODE: when you press the INS key, you enter INSERT MODE. In these modes, control and cursor keys are displayed rather than executed. Quote mode is cancelled when you press a second quote. Insert mode ends when the number of characters you entered equals the number of spaces you opened up with the INS key. For example, if you press the INS key six times, insert mode ends when you have entered six characters.

In addition, if your machine is in quote or insert mode, the ESC key cancels the mode and returns you to normal (text) mode. When you are in a mode other than normal mode, you must press the ESC key twice to use any of the ESC key special functions.

* Cancel these two functions by pressing the HOME key twice.

RUN/STOP

You can halt a BASIC program while it is running by pressing the **STOP** key. You can also use this key to halt a print out.

The **RUN** key lets you automatically load the first program on a diskette (drive 0). Just press the **SHIFT** and **RUN** keys to use this function.

C=

The **C=** key stops a program from continuing to scroll down the screen. This key is used most often when you are listing a program and you want to stop to view part of the program. Press any key to restart the scrolling.

Calculator Pad Keys

The calculator keypad on the right side of your keyboard offers all the standard calculator functions. This keypad lets you perform calculations quickly and conveniently. The keypad is not affected when you enter special modes such as the graphics mode.

? Question Mark

The question mark is the standard abbreviation for the PRINT statement in BASIC. To execute a calculation on a computer, you must precede the calculation with a PRINT statement or a question mark. The question mark had been placed on the keypad for your convenience.

For example:

```
?23.45*.06
 1.407
```

7 **8** **9**
4 **5** **6**
1 **2** **3**
0 **.** **00**

The number keys are arranged like a regular calculator. We have included a double zero **00** for your convenience. All numbers located at the top of the main keyboard section can be used in calculations when your computer is operating in the unshifted, normal mode. The keypad numbers work in any mode.

. Decimal Point

This serves as a decimal point for floating point computations. The period, located at the bottom right section of the main keyboard, also works as a decimal point in the unshifted, normal mode.

/ Slash Key

The slash key operates as a symbol for division. The slash key located at the bottom right section of the main keyboard also works as a division symbol, but only when you are in the unshifted, normal mode.

- Minus Sign

This key operates as a symbol for subtraction. It also operates as the unary minus symbol, which is the minus sign preceding negative numbers. The minus sign key located at the top right section of the main keyboard also works as the symbol for subtraction and unary minus in the unshifted, normal mode.

CE Clear/Entry

This key resembles the Clear Entry key found on most calculators. Use this key to eliminate the last number entered. **CE** clears the last number of a computation line, back to the last arithmetic operator. If the last entry is an arithmetic operator, **CE** clears the operator. If the entry is not numeric, **CE** works like the **DELETE** key.

For example:

```
10  PRINT 45*96 + 9.8/52 + 31
```

If you press the **CE** key once, the line looks like this:

```
10  PRINT 45*96 + 9.8/52 +
```

Press once more:

```
     10   PRINT 45*96 + 9.8 /52
```

Press twice more:

```
     10   PRINT 45*96 + 9.8
```

Press once more:

```
     10   PRINT 45*96 +
```

Press five times more:

```
     10   PRIN
```

* Multiplication

This operates as a symbol for multiplication. You can't use the conventional X because the computer can't distinguish the letter X from the multiplication sign. You can also use the * key on the main keyboard when you are in the unshifted, normal mode.

+ Plus Sign

This operates as a symbol for addition. It also serves as the unary plus symbol to represent positive numbers. Unary plus is automatically assumed by the computer, however, and is not necessary. You can also use the plus sign key on the main keyboard when you are in the unshifted, normal mode.

Exponentiation

Use the up arrow (a SHIFTed 6) to raise a number to a power. For example:

```
    ?12 ↑ 5
    248832
```

Execution Order In Calculations

The computer performs multiple calculations in a certain order. Problems are solved from left to right, but within that general movement, some types of calculations take precedence over others. The order of precedence follows these guidelines:

First	-	unary minus (minus sign for negative numbers, not for subtraction)
Second	↑	exponentiation, left to right
Third	* /	multiplication and division, left to right
Fourth	+ -	addition and subtraction, left to right

This means that the computer checks the whole calculation for negative numbers before doing anything else. Then it looks for exponents; then it performs all multiplication and division; then it adds and subtracts. For example:

```
?33 + 11 / 4
 35.75
```

In this example, 11 is divided by 4 and the result is added to 33. To override the order of precedence, enclose any calculations you want solved first in parentheses. All parenthetical calculations are solved before any other calculations. When more than one calculation is enclosed in parentheses, these calculations are solved left to right. Within parentheses, calculations are solved according to the order of precedence. If you add parentheses to the previous example, here's what happens:

```
?(33 + 11) / 4
 11
```

When you have more than one calculation within parentheses, you can further control the order by using parentheses within parentheses. The problem in the innermost parentheses is solved first. For example:

```
?30 + (15 * (2 - 3))
 15
```

CHAPTER 4

SOFTWARE

Commodore software available for your computer will cover a broad range of business and personal applications. These programs include word processing packages, database programs, and a variety of financial applications, such as spread sheet programs and accounting packages. Easy-to-use software will also be available for a variety of professional fields including medicine, law, agriculture, construction, and restaurant management.

Data processing professionals will be able to purchase developmental tools such as assembler software to facilitate machine language level programming. A BASIC compiler will also be available. This program permits compilation of BASIC programs into highly efficient machine language.

The 16-bit 8088 Microprocessor

The 8088 microprocessor gives you access to two widely-used operating systems, MS-DOS* and Concurrent CP/M**, and to the variety of software products these systems support.

The 8088 microprocessor is built into the X Series of advanced business machines (the BX-128-80, BX-256-80, CBMX-128-80, and the CBMX-256-80), and it can be added to all the other 'B' Series models, so the variety of software products the 8088 supports can be available to you.

The Z-80 Microprocessor and the CP/M Operating System

The Z-80 microprocessor lets you use the popular CP/M Operating System, which offers many prepackaged software programs. These programs include widely used business applications, word processing packages, high level language compilers, and more. With the CP/M Operating System, you can use many popular software packages, such as the wordprocessor WORDSTAR, the address manager MAILMERGE, database programs such as INFOSTAR, and many other best-seller industry standards.

The Commodore Software Division

The Commodore Software Division is working with software publishers to develop a high quality library of software products that will fill your computing needs. Products not already on the

---

* MS—DOS is a trademark of Microsoft, Inc.

** Concurrent CP/M is a trademark of Digital Research, Inc.

market will be available soon from your local Commodore dealer. Your dealer has more information about Commodore software, and can keep you informed of the arrival of new software products.

CHAPTER 5

USING YOUR DISK DRIVE

- Connecting Your Disk Drive
- Loading Prepackaged Programs from Diskette
- Preparing New Diskettes: HEADER Command
- Loading Your Own Programs from Diskette
- Saving Programs on Diskette
- Copying Diskettes: BACKUP Command

CONNECTING YOUR DISK DRIVE

Your computer supports the full range of Commodore CBM peripheral devices via the built-in IEEE-488 interface. Most Commodore disk units are intelligent, which means that they have their own microprocessor and memory, so they don't take up memory from your computer.

Your disk drive is easy to install:

1. Attach the PET-to-IEEE cable to the IEEE port on the back of the disk drive (see diagram).
2. Plug the other end of the cable into the IEEE port on the back of the computer. The Commodore logo faces up.
3. Make sure the plugs are securely attached.

If you are also attaching a printer, plug the cables into the disk drive first, then attach the computer and the printer. You can connect up to five disk drive units at one time to your computer by daisy chaining them together. When you attach more than one cable to the disk drive, just plug the additional cables into the first cable (see diagram). Make sure the plugs are secure.

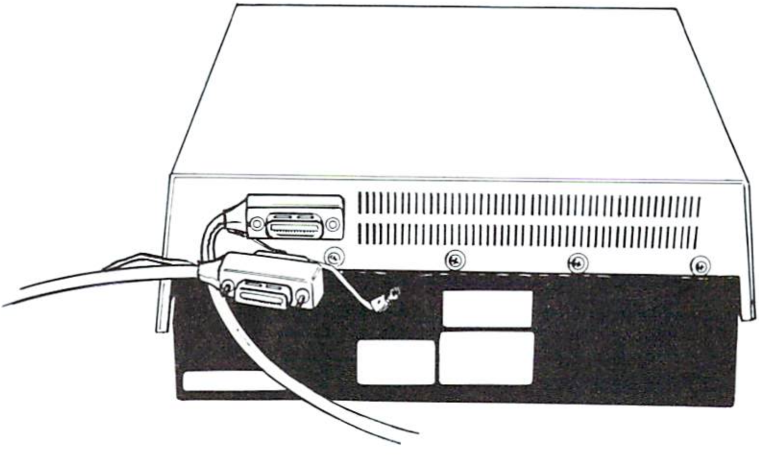


Fig. 5-1 Daisy chained peripherals

Turn on the machines' power; all power lights on all your devices should be ON.

The manual that comes with your disk drive contains more information.

NOTE: Never turn your disk drive OFF when there are disks in any drive. Always remove disks first. If the drive is turned off with disks in place, remove them before turning the drive back on.

Most prepackaged software includes special commands that show you how to load, save, and retrieve programs using your disk drive.

Loading Prepackaged Programs from Diskette

1. Start by carefully inserting the preprogrammed disk into drive zero (0).

NOTE: The computer will always assume that you're putting your disk into drive zero (0) and that you're using disk drive unit number eight (8). These are known as "default values." If you want to use another drive or unit number you must use the optional codes shown in Chapter 6 in square brackets [+1.].

---

2. Make sure that the label on the disk is facing up and is closest to you.
3. Look for a little notch on the disk (it might be covered with a small strip of tape). If you're inserting the disk properly the notch will be on the left side.
4. Close the door on the disk drive to secure the diskette.
5. Key in:

```
DLOAD "program name"
```

6. Press the **RETURN** key.

The disk will make noise and the busy light will turn on. Your screen will say:

```
SEARCHING FOR 0: program name
LOADING
READY
```

7. Wait until the READY message comes on and the cursor appears; then key in:

```
RUN
```

8. Press the **RETURN** key and your prepackaged software is ready to use.

Preparing New Diskettes: HEADER Command

Before you can use a new disk for the first time, you must format it with the HEADER command. This command divides the disk into sections called blocks, and it formats a table of contents, called a directory or catalog, for the disk. You can also reuse a disk by erasing all stored data with the HEADER command.

Follow these steps:

1. Insert the disk in drive 0. Remember to handle the disk carefully. Put the disk in so the label side is **up** and the small notch is on the **left** side as you face the drive unit. The side with the oval exposed area should go in first.
2. Close the disk drive's protective gate to secure the disk.
3. Key in:

```
HEADER "diskname", Ds (,Inn) (,ON Un)
```

diskname is any name for the disk. For example, MYDISK, MEMOS, PAYRECS, etc.
Ds identifies the drive number (0 or 1).

nn is a 2 character identification number for the diskette. The id number should be unique for each disk.
n identifies the drive unit if you have more than one.

4. Press the **RETURN** key and wait until the computer displays this message:

```
ARE YOU SURE?
```

5. Respond by keying in: Y (for Yes), and **RETURN**

The disk drive makes a noise while the new diskette is being headered. This process takes a few minutes. The computer will display a READY message when the diskette is finished.

NOTE: The HEADER command erases any information stored on a diskette and you will not be able to retrieve it. Use this command carefully.

Here are some examples of the HEADER command:

HEADER "LETTERS", D1, I04	Formats a diskette named LETTERS in drive 1 and, gives it the id number 04.
HEADER "NOTES", D0, I24, ON U9	Formats a diskette names NOTES in drive 0 of drive unit number 9 The id number is 24.

Loading Your Own Programs From Diskette

Loading a program from diskette is simple and takes only a few seconds. Once a program is loaded, you can RUN it, LIST it, or make changes and save the new version. Follow these steps to load a program:

1. Key in:

```
DLOAD "program name"
```

NOTE: You can load the first program on a diskette by using * instead of the program name. For example: DLOAD "*"

You can LOAD the first program from a diskette in drive 0 by pressing a SHIFT ed RUN/STOP key.

2. Press the RETURN key and wait for this message to be displayed on your screen:

```
SEARCHING FOR 0: program name
LOADING
READY
```

NOTE: When you load a new program into the computer's memory, any unsaved instructions and programs in memory are erased. Be sure you SAVE any information you want to keep before you key in DLOAD.

Here are some examples of the DLOAD command:

DLOAD "WORDCRAFT"	Loads the program named WORDCRAFT into memory.
DLOAD "*"	Loads the first program on the diskette, regardless of its name.

Saving Programs On Diskette

Follow these simple steps to save a BASIC program on diskette:

1. Key in:

```
DSAVE "program name"
```

2. Press RETURN and wait for this message:

```
SAVING 0: program name
OK
READY
```

NOTE: When you change a saved program and want to replace the old version, add the @ sign before the program name. For example, DSAVE "@OLDPROG" saves the new version of OLDPROG and erases the original version of the program. If you want to keep both versions, use an original name for the changed version.

Copying Diskettes: BACKUP Command

You should keep an extra copy of your stored programs for your protection. Follow these simple steps to make a backup copy of a diskette:

1. Insert a blank disk into drive 1. Insert the master disk into drive 0. Key in:

```
BACKUP Ds TO Dd (,ON Un)
```

s is the drive number of the source drive (i.e., the diskette you want to copy):

d is the drive number of the destination drive (i.e., the blank diskette you're copying on to):

n is the disk drive unit number if you have more than one drive unit connected to your system. ON Uz is an optional part of this command. *Omit it if you have only one disk drive unit in operation.*

2. Press **RETURN** and wait for this message:

```
READY
```

NOTE: Backing up takes a minute or so, but the READY message appears before the process is complete. You can find out when the backup is complete and be sure that it was successful by keying in PRINT DS$. DS$ is a reserved word variable that displays a diagnostic message about disk status, including an error message if an error occurred during a backup.

If DS$ tells you the backup was successful (00,OK,00,00,0) you can list a directory of files on the disk: CATALOG (Dn) (On Uz). Here, n is the drive number of the disk onto which you just copied. This part of the command is required unless you have only a single disk drive. Otherwise, you must name the drive. ON Uz is required when you have more than one disk drive unit. The z names the drive unit where the computer can find the disk whose contents you wish to display.

Here are some examples of the BACKUP command:

BACKUP D0 TO D1	Use when you have one dual disk drive and you are copying from drive 0 to drive 1.
BACKUP D0 TO D1,ON U9	Use when you have more than one disk drive unit and you are copying from drive 0 to drive 1 on drive unit 9.

CHAPTER 6

EXTENDED BASIC 4.0+ COMMANDS AND STATEMENTS

- Conventions in Formats
- Using BASIC Commands
- Using Basic Statements

This chapter provides formats, brief explanations and examples of the BASIC 4.0 commands and statements. It is not intended to teach BASIC. Appendix P lists tutorial books that help you learn BASIC.

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

CONVENTIONS IN FORMATS

The following conventions are used in the formats of the BASIC commands and statements:

- **KEYWORDS,** also called **RESERVED WORDS,** appear in uppercase letters. YOU MUST ENTER THESE KEYWORDS EXACTLY AS THEY APPEAR. However, many keywords have abbreviations that you can also use (see Appendix B).

  Keywords are words that are part of the BASIC language, and that your computer knows. Keywords are the central part of a command or statement. They tell the computer what kind of action you want it to take. These words cannot be used as part of your filenames or other variable names unless they are enclosed in quotation marks. However, we recommend that you NOT use keywords for variable names.

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

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

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

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

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

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

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

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

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

BASIC COMMANDS

BACKUP

This command copies all the files on a diskette to another diskette. You can copy onto a new diskette without first using the HEADER command to format the new diskette because BACKUP also formats diskettes. You should always backup disks in case the original is lost or damaged.

NOTE: Because the BACKUP command also headers diskettes, it destroys any information already stored on the diskette onto which you are copying information. Therefore, be careful when you use this command. If you're copying onto an old diskette, be sure it doesn't contain any programs you wish to keep. See also the COPY command.

BACKUP D*s* TO D*d* [ON U*n*]

s is the number of the source drive (i.e., the drive containing the disk whose files you want to copy).
d is the number of the destination drive (i.e., the drive containing the disk onto which you want to copy).
n is the number of the disk drive unit. Use this argument only if you have more than one unit connected to your system.

Examples:

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

CATALOG

This command displays the names of all the files on a diskette. The catalog of files is also called the directory.

CATALOG [" *filename* "] [D*s*] [ON U*n*]

s is the number of the drive containing the disk whose directory of filenames you want to display.

n is the number of the disk drive unit. Use this argument only if you have more than one disk drive.

Examples:

CATALOG D1	Displays a directory of all files on the disk in drive 1.
CATALOG D0, ON U9	Displays a directory of all files on the disk in drive 0 of drive unit 9 (use when the drive unit number is not 8).
cA "ABC*",D0	Displays all directory files that begin with ABC. cA is the abbreviation for CATALOG. Appendix B lists other BASIC keyword abbreviations.

COLLECT

Use this command to search the files in your directory for improperly closed files. COLLECT frees up space allocated to improperly closed files and deletes their references from the directory.

COLLECT [D*s*] [ON U*n*]

s is the number of the drive containing the diskette whose files you want to COLLECT.
n is the number of the drive unit. Use this only when you have more than one drive unit in operation.

Examples:

COLLECT	Searches files on the last drive accessed.
COLLECT D1	Searches the files on the diskette in drive 1.
COLLECT D0, ON U12	Searches the files on the diskette in drive 0 of drive unit 12.

CONCAT

Merges (concatenates) two sequential data files. When you concatenate files, the second file in your command is deleted and replaced by a new file which is the concatenation of the two files. The first file in your CONCAT command remains unaltered.

CONCAT [D*s*] "*sourcefile*" TO [D*d*] "*destfile*" [ON U*n*]

s is the drive number of the disk drive containing the file you want to add to another file.
"*sourcefile*" is the name of that file which is appended to the "*destination file*", and which remains unaltered.
d is the drive number of the disk drive containing the file to which you want to append the "*sourcefile*".
"*destfile*" is the name of the destination file, which receives the sourcefile and becomes a combination of the two files.
n is the number of the drive unit. Use this only when you have more than one disk drive unit.

Examples:

CONCAT "MYFILE" TO "YOURFILE"	Merges MYFILE and YOURFILE. YOURFILE becomes YOURFILE + MYFILE.
CONCAT "INDEX" TO "MSFILE", ON U9	Merges INDEX and MSFILE on disk drive unit 9. MSFILE becomes MSFILE + INDEX.

CONT

This command restarts the execution of a program that has been interrupted by a STOP or END statement in a program, or when you have pressed the STOP key. Execution resumes at the point where the break in the program occurred. If the break occurred after a prompt from an INPUT statement, execution continues by reprinting the prompt.

CONT is generally used in conjunction with STOP for debugging. When you stop execution, you may examine and change the values of variables (e.g., B = 200) and issue commands in direct mode, such as PRINT B. You can then resume execution with CONT or with a direct mode GOTO, which restarts at a specified line number. However, the changes you can make during a break are limited: if you edit any line of your program during a break, you can't use CONT to restart the program.

```
CONT
```

Example:

```
RUN                              Program begins executing.
?7.9
7.9
4
.0999999996
-3.8
                                 STOP key pressed.
BREAK IN 10                      Break in execution.
READY
LIST                             You can LIST a program
10  INPUT A                      during a break and still
20  X = X - B                    use CONT to resume.
30  B = 3.9
35  PRINT X + A
40  GOTO 10
READY
B = 3.6                          You can change the value
                                 of a variable IF you do
READY                            this in DIRECT MODE.
CONT                             Key in CONT to restart
 - 7.4                           execution.
 - 11.3
```

COPY

This command copies files from one diskette to another. Unlike the BACKUP command, which erases all information on the disk that receives the transfer. COPY does not affect what is already on the destination disk. In addition, COPY lets you transfer just some of the files on a disk while BACKUP transfers the entire contents of the source disk.

COPY [D*s*,] ["*sourcefile*"] TO [D*d*,] ["*destfile*"] [ON U*n*]

s is the drive number of the disk whose file is being copied.
"*sourcefile*" is the name of the file being copied.
d is the drive number of the disk that will receive the transferred file.
"*destfile*" is the name of the file that is the destination of the transferred file.
n is the unit number of the disk drive. Use only when the number is not 8 (the default value).

Examples:

COPY D0, "FILE4" TO D1, "TESTS"	Copies the file named FILE4 from the disk in drive 0 to the file named TESTS in drive 1. Only that file is copied, and all data stored on D1 remains unaffected.
COPY D0 TO D1	Copies all files on drive 0 to drive 1 without deleting any files already on drive 1.

DCLEAR

This command initializes one or more disk drives. The command defaults to drive 0 if you don't name a drive number.

DCLEAR [D*s*] [ON U*n*]

s is the number of the drive you want to initialize.
n is the unit number of the drive. Use if the number isn't 8.

Examples:

DCLEAR	Initializes drive 0.
DCLEAR D1	Initializes drive 1.

DELETE

This command erases from memory a line or group of lines from the BASIC program currently in memory:

DELETE	Erases the entire program currently in memory.
DELETE *linenumber-*	Erases all lines from the line number named to the end of the program.
DELETE *-linenumber*	Erases all lines from the start of the program to the line number named.
DELETE *linenumber-linenumber*	Erases all lines between and including the line numbers named.

DELETE [*linenumber*] [-] [*linenumber*]

Examples:

DELETE -50	Erases all lines of the current program from the first line through line 50.
DELETE 50-	Erases all lines of the current program from line 50 to the last line.

DIRECTORY

This command displays the names of the files on your diskette. If you list a filename or a prefix common to more than one filename, only those files are displayed. For example, all sequential files named SEQFILE can be listed, or you can list all filenames beginning with a common prefix by placing an * after the prefix (e.g., "WORD*" would list files including WORDPRO, WORDCRAFT, WORDLIST, etc.). If you use the ON U argument to name a drive unit and do not specify a disk drive number, the directories of both drives are displayed.

DIRECTORY [Ds] [,"*filename*"] [ON U*n*]

s is the number of the drive containing the disk whose contents you want to display.
"filename" is the name of a file or files with the same prefix that you wish to list.
n is the unit number of the disk drive. Use if the number is not 8, which is the default value.

Examples:

DIRECTORY D1	Displays a list of all the filenames in drive 1.
DIRECTORY D1 "INTRO"	Displays a list of all files named INTRO in drive 1.
DIRECTORY ON U9	Displays a list off all the filenames in both drives on drive unit 9.
DIRECTORY D0, "ABC*"	Displays all directory files that begin with "ABC" on drive 0.

DLOAD

Brings into memory a BASIC program that is stored on disk. You follow the same procedure to load a prepackaged program and a program you wrote and saved yourself. You can use DLOAD as a statement in the body of a program to chain other programs on the same diskette. This automatically runs the program in the DLOAD statement.

DLOAD "*filename*" [,Ds] [ON U*n*]

"filename" is the name of the file you want the load.
s is the number of the drive whose disk contains the file (the default is 0).
n is the number of the drive unit. Use only if this number is not 8, which is the default value.

Examples:

DLOAD "OLDFILE"	Loads a file named OLDFILE from drive 0 into memory.

DLOAD "XFILE", D1, ON U13	Loads XFILE from drive 1 of drive unit 13 into memory.

DSAVE

This command stores a BASIC program on disk. The filename can be up to 16 characters long. If you use a variable or an evaluated expression as a filename, enclose it in parentheses.

DSAVE "*filename*" [,D*s*] [ON U*n*]

"*filename*" is the name of the file you want to save.
s is the number of the drive containing the disk on which you want to store a file. The default is 0.
n is the number of the drive unit. Use only if this number is not 8, which is the default value.

Examples:

DSAVE "BASFILE"	Saves the file BASFILE to drive 0.
DSAVE "FILET1",D1	Saves the file FILET1 to drive 1.

HEADER

Before you can use a new diskette for the first time, you must format it with the HEADER command. This command divides the disk into sections called blocks, and it formats a table of contents, called a directory or catalog, for the disk. You can also reuse a disk because the HEADER command erases all stored data.

See Preparing New Diskettes: HEADER Command, in Chapter 5 for more information.

HEADER "*diskname*", D*s* [,I*nn*] [ON U*n*]

"*diskname*" is the name you give to the diskette.
s is the number of the drive containing the disk you want to HEADER.
nn is the 2 character identification number for the diskette.
n is the number for the drive unit. Use only if the unit number is not 8, which is the default value.

Examples:

HEADER "MEYERDISK", D1, I28	Headers a disk in drive 1, giving it the name MEYERDISK and the id number 28
HEADER "SCMFILE",D0,I07,ON U9	Headers a disk in drive 0, of unit 9, naming it SCMFILE with the id number 07.

KEY

This command displays a list of the current definitions of the function keys and lets you define these keys. Recall that keys F1 through F10 are predefined, but that you can redefine them. Any definition you give is erased at the end of the current session, whether it is a redefinition of an F1 through F10 key or a definition of an F11 through F20 key.

To define a function key, follow these steps:

1. Key in the word KEY and the number of the key you want to define, followed by a comma. For example:

   ```
   KEY 15,
   ```

2. Enter the definition for the key. If you want to print the definition before you execute the function, enclose the definition in quotation marks. To use the function, press the key and then press RETURN to execute. If you don't enclose the definition in quotation marks, the function is executed immediately when you press the function key. If you want the function to do more than one operation, string the operations together with plus signs. For example:

   ```
   KEY 15,"PRINT + CHR$(142)"
   ```

This switches the keyboard to graphics mode.

KEY [keynumber, "*definition* [+ *definition* . . . + *definition*]"]

Examples:

KEY 5,CHR$(34)	PRINTs a quotation mark immediately when you press Key 5.
KEY 17, "PRINT CHR$(142) + CHR$(77) + CHR$(13) + CHR$(65)"	When you press Key 17, the text of what the key does is displayed without quotation marks. Cursor remains at the end of the line until you press RETURN to execute the function. This function does four things: 1. switches to graphics mode 2. PRINTs an M 3. activates RETURN key 4. PRINTs an A

LIST

This command displays a listing of all or part of the program currently in memory. After a LIST command executes, BASIC always returns to the direct mode, also called the command level.

LIST	Lists the entire program.
LIST linenumber-	Lists all lines from the line number named to the end of the program.
LIST-linenumber	Lists all the lines from the beginning of the program to the line number named.
LIST linenumber-linenumber	Lists all the lines between and including the numbers named.

LIST [[*linenumber*] [-] [*linenumber*]]

Examples:

LIST	Lists all the lines in the current program.
LIST-50	Lists the lines from the beginning to line 50.

LOAD

This command brings into memory a program stored on diskette. LOAD closes all open files and deletes all variables and program lines currently in memory, so be sure to save anything you want to keep before you issue the LOAD command.

You can use LOAD as a statement in a program to chain several programs. If you execute a LOAD statement from one program, the loaded program is RUN after it is LOADed, and all data files are kept open. None of the variables is cleared during a chain operation.

LOAD "[D*s*:]*filename*",*Dev#*

s is the drive number containing the disk from which the program will be loaded. The default is 0.
"filename" is the name of the file you want to load into memory.
Dev# is the device number of the disk drive containing the file you want to load. The disk drive device number is 8 unless you change it.

Examples:

LOAD "*",8	Loads the first file on the disk in drive 0.
LOAD "MEYERFILE",8	Loads the file MEYERFILE from drive 0 into memory.
LOAD "1:SCMFILE",8	Loads SCMFILE from drive 1 into memory.
LOAD "1:MY*",8	Loads first file in drive 1 that begins with the letters MY.

NEW

New erases the BASIC program and data currently in memory so that a new program can be entered. Be sure to save anything you want to keep before you issue a NEW command.

You should always use the NEW command before you enter a new program to be sure that memory is clear, otherwise unwanted lines from the previous program could merge with your new program.

```
NEW
```

RENAME

This command changes the name of a file on a diskette without altering the file itself. You cannot execute a RENAME command on a currently open file.

```
RENAME [Ds,] "oldname" TO "newname" [,ON Un]
```

s is the number of the disk drive containing the file you want to RENAME. The default is 0.
"*oldname*" is the current name of the file.
"*newname*" is the name to want to use.
n is the unit number. Use only if this number is not 8, which is the default value.

Examples:

`RENAME D1, "HERFILE" TO "MYFILE"`	Gives the new name MYFILE to HERFILE on drive 1.
`RENAME "DRAFT" TO "BOOKFILE"`	Gives the new name BOOKFILE to DRAFT on drive 0.

RUN

This command executes the BASIC program currently in memory.

`RUN`	Executes the program currently in memory.
`RUN linenumber`	Executes the program beginning at the line number named.

RUN [*linenumber*]

linenumber is the number of the line at which you want to begin execution if you don't want to start at the first line.

Example:

RUN 60	Executes the program from line 60, ignoring previous lines.

SAVE

SAVE stores a program on diskette. If you include the @ sign when you SAVE a program whose name is the same as another program you already have saved on the same diskette, the program that is already saved is replaced by the new program. If you revise a program but want to keep both copies, save them under different names.

SAVE "[[@]D*s*:]*filename*",*Dev#*

@ replaces an existing program that has the same file name.
s is the drive number containing the disk on which you want to save the program. The default is 0.
"filename" is the name of the program you want to save.
Dev# is the device number of the disk drive that contains the disk on which you want to store the program. The disk drive device number is 8 unless you change it.

Examples:

SAVE "1:MEYERFILE",8	Stores the program MEYERFILE on the disk in drive 1.
SAVE "SCMFILE",8	Stores the program SCMFILE on the disk in drive 0.
SAVE "@1:MEYERFILE",8	Replaces the existing program MEYERFILE in drive 1.

SCRATCH

Use this command to delete files from a diskette. When you issue this command, the computer displays the prompt ARE YOU SURE? before executing the SCRATCH. You must respond with YES or Y to begin execution.

SCRATCH "*filename*" [,Ds] [ON U*n*]

"filename" names the file you want to delete.
s is the drive number of the file containing the file you want to SCRATCH. The default is 0.
n is the unit number of the drive. Use only if the number is not 8, the default value.

Examples:

SCRATCH "SCMFILE" ARE YOU SURE? YES	Deletes the file SCMFILE from the disk in drive 0.
SCRATCH "THESIS",D1 ARE YOU SURE? YES	Deletes the file THESIS from the disk in drive 1.

VERIFY

Use this command to check a program on disk against the program currently in memory. VERIFY informs you if there are discrepancies.

VERIFY "[D*s*:]*filename*",*Dev#*

s is the drive number containing the stored program.
The default is 0.
"filename" is the name of the file you want to verify.
Dev# is the device number of the drive containing the stored program you're checking against the current program.
The disk drive device number is 8 unless you change it.

Example:

VERIFY "MEYERFILE",8	Checks the program MEYERFILE stored on drive 0 against the program currently in memory.
VERIFY "1:MYFILE",8	Checks the program MYFILE stored on drive 1 against the program currently in memory.

BASIC STATEMENTS

Statements are BASIC instructions that are issued in programs. They are always preceded by a line number. Most of the statements described here can also be used as BASIC commands in direct mode if you omit the line number. Similarly, most BASIC commands can be used as BASIC statements in program mode if you prefix them with a line number.

APPEND

This statement opens a sequential file and positions the file pointers beyond the current end of file so that you can write additional data to that file. APPEND is like the DOPEN statement, except that APPEND applies only to sequential files.

```
[linenumber] APPEND#fn,"filename" [,Ds] [ON Un]
```

fn is the filenumber of the file you want to reopen and add to (this is called the logical file number).
"filename" is the name of the file you want to APPEND.
s is the number of the drive that contains the file (defaults to 0).
n is the unit number of the disk drive unit (defaults to 8).

Example:

`10 APPEND#3,"MEYERFILE"`	Reopens MEYERFILE, logical file #3, on drive 0 for appending.

BANK

This statement sets the indirection bank number for use with some BASIC commands such as PEEK, POKE, BLOAD, and BSAVE that refer directly to memory bank locations. The BANK statement lets you pick the memory bank into which information will be placed. There are 16 BANKs numbered 0 through 15.

```
[linenumber] BANK expression
```

expression is any number, variable or numeric expression that equals any number between 0 and 15.

Examples:

`10 BANK 3`	Sets the bank number to 3.
`20 POKE 1024,20`	Stores 20 at location 1024 in BANK 3.
`5 FOR A = 0 TO 5`	Starts a loop that gives A a new value (0 through 5) each time the loop executes.
`10 BANK A`	Sets the bank number to the value of A, which progresses from 0 through 5.
`20 BLOAD "TEST"`	Loads the file TEST to the bank whose number is the value A. By the end of the loop, TEST is loaded in BANKS 0 through 5.
`30 NEXT A`	

BLOAD

This BASIC statement loads an executable machine language program into any memory location.

[*linenumber*] BLOAD [*fileoptions*] [,ON U*n*] [,B*z*] [,P*l*]

fileoptions are the arguments that specify the file you want to load. They can include file name, file number, drive number, drive unit number, etc.
z is the number of the memory BANK where you want to load the machine language program. If you don't name a bank, BLOAD loads to the last bank named. If no bank has been named in the program, BLOAD defaults to bank 15.
l is the location (low offset) in the bank where you want to start loading.

Examples:

`10 BLOAD "RATES",D1,ON U9,B3`	Loads RATES from drive 1, drive unit 9, into BANK 3.
`20 BLOAD "TEST",D1, B3, P1024`	Loads TEST into BANK 3 from drive 1 starting at location 1024.

BSAVE

This BASIC statement saves a machine language program from any memory location you name. BSAVE defaults to the last byte in the bank ($0bFFFF, where b = bank 0 through F).

[*linenumber*] BSAVE "*filename*"[,*fileopts*] [,ON U*n*] [,B*z*]
[,P*l*] [TO P*h*]

"*filename*" is the name of the file you want to save.
fileopts include drive number, drive unit number, etc.
z is the number of the BANK where the program is located.
l is the location (low offset) in the bank where you want to start saving.
h is the location (high offset) in the bank where the information you're saving ends.

Example:

10 BSAVE "TEST",D1,B3, P512 TO P1024	Saves file TEST on drive 1, from BANK 3, memory location 512 to 1024.

CLOSE

This statement closes a files that was opened previously with an OPEN statement. You must use the same file number in both the OPEN and the CLOSE statements. A CLOSE for a sequential output file writes the final buffer of output.

[*linenumber*] CLOSE *filenumber*

Example:

100 CLOSE 3	Closes file number 3.

CLR

This command clears all BASIC variables currently in memory, but leaves the program itself intact. The CLR command is automatically executed when you give a RUN command.

[linenumber] CLR

Example:

```
10 FOR X = 1 TO 4              Loop executes 4 times.
20 A = 5: B = X                As X is incremented by 1,
30 C = A + B: PRINT C,X        C and X are PRINTed on the
40 NEXT                        same line until X = 4.
50 CLR                         CLeaRs all variables.
60 PRINT C,X                   PRINTs the CLeaRed variables.
RUN
 6         1                   The values for C and X are
 7         2                   PRINTed as the loop executes
 8         3                   4 times.
 9         4
 0         0                   The zeroes PRINTed for C and X
                               after the CLR statement show
                               that the variables are CLeaRed.
```

CMD

This statement lets you redirect output. For example, output that would normally go to the screen can be redirected with CMD to go instead to a printer or a file. You must use CMD with an OPEN statement that uses the same file number. The device to which output will be redirected is named in the OPEN statement.

[linenumber] CMD *filenumber [,printlist]*

filenumber is the number of the file whose output you want to redirect.
printlist is a list of character strings, numeric variables, or expressions written to the device when the CMD statement is executed.

Example:

```
10  OPEN5,4                    OPENs file number 5 and names
                               the printer as the output device (4).
```

`20 CMD5,"PROGLIST"`	Directs PROGLIST to be written to the printer.
`30 PRINT "TEXT"`	PRINT statements following a CMD are directed to the device named in CMD.

DATA

The DATA statement holds numeric and string constants that are matched with variables in READ statements. The DATA constants are accessed consecutively by READ variables. The variable type (numeric or string) in the READ statement must match the constant type in the corresponding DATA statement. Constants in DATA statements may be reread after you issue a RESTORE statement.

The DATA statement does not have to precede the READ statement. When a READ statement has read all the constants in a DATA statement, it will look for another DATA statement, so the number of items in any DATA statement does not have to equal the number of items in a READ statement. However, the computer will display an OUT OF DATA error message if the total number of DATA constants accessible in a program is fewer than the total number of READ variables.

[*linenumber*] DATA *constant* [*,constant, . . . , constant*]

constant is any numeric (fixed point, floating point, or integer) value or any string value. Numeric expressions are not allowed. String constants do not need to be enclosed in quotation marks unless they contain commas, colons, or leading or trailing spaces.

Examples:

```
10 DATA 1,2,3,4,5
20 READ A,B
30 READ C,D
40 PRINT A;B;C;D
RUN
 1     2     3     4
```

Lists DATA constants. The first READ variable acquires the first DATA constant, etc.

```
NEW
10 DATA 1,2,3,4,5
20 READ A,B,C,D,E,F,G,H
RUN

?OUT OF DATA IN 20
```

You can have more DATA constants than READ variables but not vice versa.

DCLOSE

This command can CLOSE all the files currently open on a disk, or only the logical file specified. If you don't specify a file number, all OPENed filed are CLOSEd.

[*linenumber*] DCLOSE [#*lf*] [ON U*n*]

lf is the number of the logical file you want to CLOSE.
n is the number of the drive unit.

Examples:

`10 DCLOSE`	Closes all files OPEN on default device (8).
`10 DCLOSE#3`	Closes the file with the logical file number 3.
`10 DCLOSE ON U9`	Closes all files OPEN on unit 9.

DEF FN

This statement lets you define your own functions and use them in a program by using only the function name. This statement can save time and space when you want to use a complex formula more than once in a program. You must define the function with the DEF FN statement before you can call the function in a program.

[*linenumber*] DEF FN*na* (*argument*) = *formula*

na is the name of the function. It must be a legal variable name, and you must precede the name with FN when you call the function.

argument can be any numeric variable; it must be enclosed in parentheses.

formula is the expression that performs the function's operations. Any variable name that appears in this formula serves only to define the function; it does not affect program variables that have the same name.

Example:

`10 DEF FNAB (X) = X/Y3`	Defines the function FNAB.
`20 T = FNAB (I)`	Calls FNAB.

DIM

The DIMension statement allocates storage for an array and sets the maximum values for the array variable subscripts. You MUST use the DIM statement to DIMension arrays containing more than 10 elements. To find the number of elements in an array, multiply the values of each subscript plus one. For example, an array DIMensioned (3,2) has (3 + 1)*(2 + 1) elements.

The DIM statement sets the value of all elements of the array to an initial value of zero.

Matrices can have up to 255 dimensions, but the size of each must be less than 32767.

[*linenumber*] DIM *variable*(*subscript* [, . . . ,*subscript*]),
[*variable* (*subscript* [, . . . ,*subscript*]) . . .]

variable is the name of the array.

subscript is the size of the dimension of the array.

Subscripts must be enclosed in parentheses.

Examples:

`10 DIM A(20)`	DIMensions a one-dimensional array with 21 elements.
`20 DIM A$(4,4,4)`	DIMensions a three-dimensional array with 125 elements (4 + 1*4 + 1*4 + 1 = 125).

DISPOSE

Use this statement in error trapping procedures to eliminate unwanted FOR /NEXT loops or GOSUB /RETURN addresses without leaving invalid information on the stack.

[*linenumber*] DISPOSE < FOR| GOSUB >

You must choose either FOR or GOSUB as an argument for a DISPOSE statement.

Example:

30 FOR J = 1 TO 10	Starts a FOR /NEXT loop.
40 PRINT J	
50 IF J = 5 THEN DISPOSE FOR:GOTO 70	Eliminates the loop when J = 5, and moves to line 70.
60 NEXT J	

DOPEN

This statement declares a sequential or random access file for read or write access. A sequential file is opened for read access unless you include the W argument in the statement.

[*linenumber*] DOPEN #*1f*, "*filename*" [,L*y*] [,D*s*] [ON U*n*] [, *W*]

1f is the logical file number of the file you want to open.
"*filename*" is the name of this file.
y is the record length for a nonsequential file. You must include this argument when you create a relative file.
s is the disk drive number. Default is 0.
n is the disk drive unit number. Default is 8.
W indicates write access to a sequential file.

Examples:

10 DOPEN#5,"TEST"	Opens file 5 named TEST on drive 0.

`10 A$ = "RATES"` `20 DOPEN#6,(A$)`	Opens file 6 named RATES. When you use a variable to stand for a file name, you must enclose it in parentheses.
`20 DOPEN#2,"@FILE1,W",D1`	Replaces file 1 with file 2 and opens file 2 on drive 1.

END

END terminates program execution and returns to direct mode.

[*linenumber*] END

FOR/TO/STEP

This compound statement starts a loop that performs a series of instructions a set number of times, and always executes at least once. This statement is always used with a next statement.

FOR names a variable that serves as a counter to control the number of executions of the loop. TO sets the number of executions, such as 1 TO 10, which means that the loop executes 10 times.

STEP is an optional part of the statement that you can use to change the amount the counter is incremented from the default of 1. For example, 1 TO 10 STEP 2 makes the loop execute only 5 times, since the counter is now incremented by 2 each time the loop executes.

You can also count backwards in a FOR loop by reversing the order of the numbers in the TO arguments and by using a negative value as the STEP argument.

You can also nest FOR/NEXT loops, that is, a FOR/NEXT loop can be placed inside another FOR/NEXT loop. When you do this, the inside loop must end before the outside loop, and the loops must have a different variable as the counter.

[*linenumber*] FOR *variable* = expression1 TO *expression2*
[STEP *expression3*]

variable is the name of the loop counter.
expression1 is the beginning value of the counter.
expression2 is the ending value of the counter.
expression3 is the value of the increment of the counter. Defaults to 1.

Examples:

`10 FOR X = 1 TO 5`	Sets X as the counter and limits to 5 the number of executions of the loop.
`20 A = A + X:PRINT A`	Each time the loop runs, this statement will execute again.
`30 NEXT`	Tells the computer to get the next value of X.
`10 FOR G = 1 TO 10 STEP 2` `20 PRINT G: NEXT G`	Starts a loop whose counter increments by 2 each time the loop executes.
`10 FOR R = 25 TO 5 STEP -.5`	Starts a loop whose counter decrements by -.5 each time the loop executes.

GET

This statement provides another way to assign data values to variables. GET scans the keyboard buffer and reads a single character. If you don't type a character, a null character is automatically assigned. The GET statement is often placed in a loop that continues until you type a character that is assigned to the GET variable.

The GET variable is usually a string variable, which can accept either string or numeric input. A numeric variable can only accept numeric input.

[*linenumber*] GET *variable*

Example:

`10 GETA$: IF A$ = ""THEN 10`	GET asks you to type a single character that is assigned to A$. The IF tells the computer to keep checking until you enter a character.

GET#

GET# reads a single character from a file. You must have already OPENed the file with the same logical file number before you can use GET#.

[*linenumber*] GET# *filenumber, variable*

filenumber is the logical file number of the OPENed file from which your GET# is reading a character.
variable is the variable to which the character read by GET# is assigned.

Example:

`10 DOPEN#5, "TEST"`	Opens logical file 5.
`30 GET#5, F$`	Reads a single character from file 5 and assigns it to F$.

GOSUB

The GOSUB statement lets you branch to a subroutine. The subroutine must be terminated by a RETURN statement that sends control back to the body of the program. You can nest GOSUB /RETURN statements up to 23 deep.

[*linenumber*] GOSUB *linenumber2*

linenumber2 is the line where the subroutine starts.

Examples:

`75 GOSUB 10`	Sends control to a subroutine starting at line 10.
`95 GOSUB 125`	Sends control to a subroutine at line 125.

GOTO

GOTO unconditionally branches the program to a specified line. GOTO does not require any sort of return statement. If you want to stop a loop begun by a GOTO statement, you must break into execution with a STOP or include another statement that ends the loop.

[*linenumber*] GOTO *linenumber2*

Examples:

```
10 INPUT A$: PRINT A$
20 GOTO 10
```

The GOTO in line 20 causes line 10 to execute repeatedly.

```
10 INPUT A: PRINT A*1.06
20 IF A<100 GOTO 10
30 IF A=>100 THEN END
```

The IF statements provide a way to end the GOTO loop in line 20, which stops executing when line 20 is false.

IF/THEN/ELSE
IF/GOTO

The IF statement is another way to control program execution. This statement tells the computer to check IF a condition is true, and IF it is, follow the instructions following THEN. IF that condition is false, the program skips to the next line to continue. You can use an IF statement to start a loop or to decide whether certain parts of a program will execute. IF statements may be nested.

[linenumber] IF *expression* THEN *tclause* [:ELSE *eclause*]

expression sets the condition to be verified. The THEN clause instructions are executed only if the expression is true.
tclause is the set of instructions to be performed when the expression is true.
eclause is another set of instructions to be performed when the expression is false.

Expressions in IF statements usually include one of the following relational operators:

SYMBOL	MEANING	SYMBOL	MEANING
>	greater than	<>	not equal to
<	less than	>=	equal to or greater than
=	equal to	<=	equal to or less than

Examples:

`10 IF A> B THEN PRINT A,B`	A and B are printed only if A is greater than B.
`10 IF A> 100 GOTO 125`	If A is greater than 100, execution goes to line 125.
`10 IF A< = 99 THEN A = A*1.5:ELSE A = 2`	If A is less than or equals 99, instructions after THEN are executed and the ELSE clause is not. If A is greater than 99, THEN's argument isn't executed, and ELSE's is.

INPUT

This statement lets you input values from the keyboard during execution. When you execute the program, you are automatically prompted by a question mark for INPUT. You can also write a prompt message. Program execution does not continue until you respond to an INPUT prompt.

The number of data items you supply in response to an INPUT prompt must equal the number of variables in the INPUT statement. INPUT variables may be either string or numeric. INPUT assumes that commas and colons signal the end of a data item.

[*linenumber*] INPUT ["*promptstring*";] *variable list*

"*promptstring*" is optional text you can add to precede the question mark prompt.
variable list is one or more variables whose values you are INPUTting.

Example:

```
10 INPUT A$: PRINT "CONTINUE"
20 IF A$ <> "STOP" GOTO 10
RUN
 ? COMMODORE
 CONTINUE
 ? B SERIES
```

As long as you don't enter STOP when you are prompted for INPUT, execution continues and you are prompted again.

```
CONTINUE
? STOP
READY
```

INPUT#

INPUT# is similar to the INPUT statement, except it reads data from an OPENed disk file. Leading spaces are ignored. INPUT# assumes that commas, colons, and carriage returns signal the end of a data item.

[*linenumber*] INPUT# *filenumber, variable list*

filenumber indicates the file from which INPUT# is reading data. *variable list* is one or more variables whose values you are INPUTting.

Example:

`10 INPUT#3,A$,A`	Reads values for A$ and A from file 3

LET

LET assigns a value to a variable. The word LET, however, is always optional. In other words, LET A = 3 is the same as A = 3. The presence of the equal sign is sufficient when you are assigning an expression to a variable.

[*linenumber*] [LET] *variable* = *expression*

Examples

```
10 LET A$ = "STRING"
20 A = 32-28
30 B$ = "STRING"
```

NEXT

NEXT is the statement that does the following:

- indicates where a FOR /NEXT loop ends
- increments the value of the FOR value by the amount declared in the STEP argument (default = 1) when the loop is not finished
- sends execution out of the FOR loop when the loop is finished.

NEXT only appears as the complement of a FOR loop, and every FOR loop must have a NEXT statement. These loops may be nested.

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

variable is optional; when loops are nested the first NEXT is assumed to go with the last FOR statement. When the NEXT variable is included, it must match the FOR variable.

Example:

```
10 FOR A = 1 TO 2: PRINT A  Loop A executes twice.
20 FOR F = 99 TO 97
   STEP-1:PRINT F           Loop F. executes 3 times.
30 NEXT F,A                 Loop F, the last named, is
RUN                         the first finished.
 1                          Loop A runs once.
 99                         Loop F runs all three times
 98                         because it finishes before A
 97                         can execute a second time.
 2                          Loop A runs another time.
 99                         Loop F runs three times again
 98                         because it is inside A.
 97
```

ON/GOSUB

This compound statement branches the program to one of several subroutines specified by the line numbers listed as GOSUB arguments. The destination depends on the value returned when the ON expression is evaluated. If the value is 1, control branches to the first subroutine; if it's 2, control goes to the second, etc. If the value of the expression is negative, you receive an error message. If the expression is zero or greater than the number of items in the list, control passes to the line following the ON /GOSUB statement.

[*linenumber*] ON *expression* GOSUB *list of linenumbers*

expression determines which subroutine receives control when the expression is evaluated.
list of linenumbers corresponds to the subroutine to which the program might branch.

Example:

```
10 FOR A = 1 TO 3
20 ON A GOSUB 75,95,115
30 NEXT
```

The first time the FOR loop executes, control passes to the first subroutine (at line 75) because A = 1, etc.

ON/GOTO

ON /GOTO resembles ON /GOSUB, except that ON /GOTO sends control to one of several specified line numbers rather than to subroutines. All other conditions are the same.

[*linenumber*] ON *expression* GOTO *list of linenumbers*

expression determines which lines receives control when the expression is evaluated.
list of linenumbers corresponds to the line numbers to which the program might branch.

Example:

```
50 ON X-1 GOTO 125,150,200
```

When X-1 = 1, control goes to line 125; when X-1 = 2, control goes to 150, etc.

OPEN

This statement establishes an Input/Output (I/O) channel to the screen or to an external device such as a disk drive, a printer, or the IEEE bus.

[*linenumber*] OPEN *filenumber* [,*devicenumber* [,*secondary address* [,"*filename*"]]]

filenumber of the logical number of the file you want to OPEN. This number must be between 0 and 255.
devicenumber designates the external device to which you want to OPEN a channel. The device numbers for external devices are: disk = 8 through 15 (default 8); printer = 4; screen = 3.
secondary address (0 through 15) is required in some cases. The addresses are: 0 through 1 = commands other than OPEN; 2 through 14 = data files; 15 = command channel.
"filename" is the name of the file referred to in the secondary address.

Examples:

10 OPEN 1,3	OPENs the screen as a device.
20 OPEN 2,4	OPENs a channel to the printer.
30 OPEN 4,8,15	OPENs a command channel on the disk.

PEEK

PEEK* lets you read the information at a specific memory location. PEEK returns the value (0 - 255) of a single byte.

[*linenumber*] PEEK (*memorylocation*)

memory location gives the memory address of the byte whose value you want to read.

* PEEK and POKE default to the BASIC text bank. If you want to access another bank, you must issue the BANK command first.

Example:

`10 A = PEEK(59468): PRINT A`	PRINTs the value of the byte located in memory at 59468.

POKE

POKE* lets you write a byte into a specific memory location. POKE is complemented by the function PEEK. Use PEEK and POKE for efficient and specific data storage, and for assembly language subroutine operations such as loading and passing arguments.

You can only POKE to RAM (Random Access Memory), though no error is flagged if you POKE to ROM (Read Only Memory).

[*linenumber*] POKE *location, value*

location is the place in memory where you want to place a value.
value is what you want to place in a specific memory location.

Example:

`10 POKE 59468,14`	Sets the character set to upper /lower case mode.
`20 A = PEEK(59468):PRINT A`	PRINTs 14 as the value for A since you previously POKEd 14 into location 59468.

PRINT

PRINT displays on the screen any information you specify. The punctuation you use in the PRINT statement determines the position of PRINTed items. BASIC divides each line into print zones of ten spaces each. When you separate PRINT items with a comma, each item is PRINTed in a new print zone. A semicolon PRINTs items right next to each other (however, PRINTed numbers are always followed by a space).

If you end a PRINT statement with either a comma or a semicolon, the next PRINT statement begins on the same line. If there is no punctuation at the end of the statement, a carriage

---

* PEEK and POKE default to the BASIC text bank. If you want to access another bank, you must issue the BANK command first.

return is assumed, and the next PRINT statement begins on the next line.

[*linenumber*] PRINT [*printlist*]

printlist can include any of the following:

1. Text, which must always be enclosed in quotation marks.
2. Variable names: if enclosed in quotation marks, the value of the variable PRINTs; if not enclosed, the variable name PRINTs.
3. Functions.
4. Punctuation marks (used for formatting output.

Examples:

Statement	Prints
10 A = 3*4: PRINT "A = ";A	A = 12
20 PRINT "REPORT TITLE"	REPORT TITLE
30 A = 3:PRINT "A = ";A,"B = ";A*2	A = 3 B = 6
40 PRINT 1,2,3	1 2 3
50 PRINT 1;2;3	1 2 3

PRINT#

PRINT# resembles PRINT, but PRINT# writes the values listed to the file associated with the file number in the PRINT# statement. Recall that the file must have been previously OPENed with the same file number.

[*filenumber*] PRINT# *filenumber, printlist*

filenumber identifies the logical file into which you want to write data.
printlist contains the data you want to write to the file.

Example:

10 PRINT#3,"TEST DATA :"	Writes this information to file number 3.

PRINT USING
PRINT# USING

These statements let you define the format of the string and numeric output you want to print.

[*linenumber*] PRINT [#*filenumber*,] USING "*formatlist*"; *printlist* [;]

filenumber names the file into which you wish to write formatted data. The file must have been previously OPENed.
"*formatlist*" defines the format of your output.
printlist is the data you want to PRINT in the defined format.
The format symbols are:

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

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

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

```
10  PRINT USING "####" ,X
```

For these values for x, this format displays:

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

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

For example, if you want a field to contain a maximum of seven characters, you can use this PRINT USING statement to print a string variable:

```
PRINT USING "#######" ; NAME,$
```

If the string NAME$ contained more than seven characters, the characters after the seventh character will be truncated when the string is printed. For example, if NAME$ = "SHABINGER", this format will print SHABING.

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

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

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

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

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

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

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

A *dollar sign* ($) symbol shows that a dollar sign will be printed in the number. You must specify at least one # before the dollar sign or else the dollar sign will not *float*. If you specify a dollar sign without a leading #, the dollar sign is printed in the position shown in the *format field*. If you specify at least one # before the dollar sign, the dollar sign *floats* to be placed just before the number.

If you specify commas and /or a plus or minus sign in a *format field* with a dollar sign, your program will print a comma or sign before the dollar sign.

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

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

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

A *greater than sign* (>) is used to *right justify* a string in a field. You specify the field width by the number of characters (# and =)

in the *format field*. If the string contains fewer characters than the field width, the string is right justified in the field. If the string contains more characters than can be fit into the field, the rightmost characters are truncated and the string fills the entire field.

Examples:

Field	Expression	Result	Comment
+##	1	+ 1	Fill character between sign and number.
#.##+	−.01	0.01−	Leading zero added.
−.##	−.1	−.10	Leading zero suppressed by minus sign.
##.#−	1	1.0	Trailing zero added.
+##+	1	ERROR	Two plus symbols.
+##.#−	1	ERROR	Plus and minus symbols.
####	−100.5	−101	Rounded to no decimal places.
####	−1000	****	Overflow because four digits and minus sign cannot fit in field.
#.##	−4E−03	−.00	Rounded to −0
###.	10	10.	Decimal point added.
#.#.	1	ERROR	Two decimal points.
##,##	100	1,00	
##,##	10.4	10	Comma suppressed and value rounded.
#,###.##	1000.009	1,000.01	Rounded.
##,##	−1	−1	Comma suppressed.
##,##	−10	−10	Minus overrules comma. No leading digit before the comma.
##=>.>	1000	1000.0	> and = treated as # since in numeric field.
+>==,#	1	+>==,1	At least one # must precede the comma. >, =, and comma are treated as symbols to print, not as format field items.
+>=#,#	1	+ 1	> and = treated as # since in numeric field.
#$##	1	$1	Leading $ sign.

Field	Expression	Result	Comment
###$	−1	−$1	Sign precedes $.
##$##	−1	−$1	Sign precedes $.
###$−	−1	$1−	Sign in last position.
+$###	−1	+$ −1	At least one # must precede $. + and $ treated as symbols to print, not as format field items.
+#$##	−1	−$1	
+#.#↑↑↑↑	1	+1.0E+00	E−format output
##.##↑↑↑↑	−100000	−1.00E+05	
##↑↑↑	1	ERROR	Only three carets.
##↑↑↑↑↑	34	34E+00↑	Fifth caret seen as text char−acter and is always printed.
##.##↑↑↑↑	cbm	cbm	String data item, printed left justified in nine character field.
###>#	cbm	cbm	Printed right justified in five character field.
=####	cbm	cbm	Centered in eight character field.
#,$#=+	cbm	cbm	Only + affects centering in six character field. Other symbols are translated to #.

PUDEF

PUDEF lets you use characters in a PRINT USING statement that are not permitted in the PRINT USING format list. PUDEF let you redefine up to 4 symbols in the PRINT USING statement. You can change blanks, commas, decimals points, and dollar signs into some other character by placing the new character in the correct position in the PUDEF control string.

[linenumber] PUDEF "*controlstring*"

controlstring is a list of new characters you want to place in your PRINT USING format. The control string can contain up to four new characters:

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

- Character position 2 is the comma character. Default is a comma.
- Character position 3 is the decimal point.
- Character position 4 is the dollar sign.

Examples:

`10 PUDEF "*"`	PRINTs * in the place of blanks.
`20 PUDEF " @"`	PRINTs @ in place of commas.
`30 PUDEF " .,"`	PRINTs decimal points in place of commas, and commas in place of decimal points.

READ

This statement assigns values from DATA statements to variables listed as READ arguments. The data types must be the same in both statements. A single READ statement may read data from several DATA statements, and several READ statements may read from one DATA statement. DATA lists must contain enough values to assign one value to each READ variable, but any extra DATA values are ignored.

You can reREAD data by using the RESTORE statement.

[*linenumber*] READ *variable list*

variable list is the list of variables whose values are assigned from DATA statement constants.

Examples:

```
10 DATA 1,2,3
20 READ A,B,C
```

Assigns 1 to A, 2 to B, and 3 to C.

```
10 DATA 1,2,3,4
20 READ A,B:PRINT A;B
30 RESTORE
40 READ C,D:PRINT C;D
RUN
  1  2
  1  2
```

Assigns 1 to A; 2 to B. Moves pointer reading data back to beginning, so 1 is assigned to C; 2 to D.

RECORD

RECORD adjusts a relative file pointer to select any byte (character) of any record in the relative file. The file must have been previously OPENed.

[*linenumber*] RECORD# *filenumber,recordnumber*[*,bytenumber*]

filenumber is the logical number of the relative file.
recordnumber is the number of the relative file record in which the byte you want to select is located (must be between 0 and 65535). 0 and 1 both index the first relative file record.
bytenumber indicates at which byte (1 through 254) you want to select.

Examples:

```
10 DOPEN#2,"RELFILE",L50
```

OPENs a relative file with a record length of 50.

```
20 RECORD#2,10,50
25 PRINT#2,CHR$(255)
30 DCLOSE#2
```

Allocates space for 10 records and moves the pointer to the end.

```
10 FOR J = 1 TO 10
20 RECORD#2,(J),1
30 PRINT#2,"RECORD";J
40 NEXT
```

Writes ten records to position 1 in each record.

REM

The REMark statement lets you insert explanatory remarks in your programs. These remarks are not executable and do not affect the program.

[linenumber] REM [text]

text can be any commentary that clarifies your program.
REMarks do not need to be enclosed in parentheses.

Examples:

```
10  PRINT X: REM X IS TAXABLE TOTAL   All text following REM
20  REM REMARKS MAKE PROGRAMS         does not execute.
    EASY TO READ
```

RESTORE

RESTore lets you reREAD the values in a DATA statement from the beginning.

[*linenumber*] RESTORE [*linenumber2*]

linenumber2 is the line number where the pointer is moved back for DATA to be reREAD.

Examples:

```
10  DATA 1,2
15  DATA 8,9,10
20  READ A,B,C,D:PRINT A;B;C;D Asigns first 4 DATA values.
30  RESTORE 15                 Moves pointer to start of 15.
40  READ E,F,G: PRINT E;F;G    Assigns data from start of 15.
50  RESTORE                    Moves pointer to start of first
                               DATA statement.
60  READ A,B,C:PRINT A;B;C     Assigns first 3 DATA values.
RUN
 1  2  8  9
 8  9  10
 1  2  8
```

RESUME

This statement lets you continue with program execution after an error has been trapped and processed by your error handling routine. If you do not name a specific line at which execution is to RESUME, the program will attempt to re-execute the statement in error. If you select the NEXT argument, execution resumes at the line following the error. If you select some other line number, execution continues there.

[*linenumber*] RESUME [NEXT | *linenumber2*]

linenumber2 is any line you select for execution to resume.

Example:

70 TRAP 100: REM IF AN ERROR OCCURS GOTO LINE 100 75 PRINT VAL (L): REM THIS IS AN ERROR BECAUSE L = 0	Sends program to line 100 if there is any error in the program.
120 RESUME NEXT	Restarts program at line after error.

RETURN

RETURN ends a subroutine and branches the program back to the statement following the GOSUB statement that started the subroutine.

[*linenumber*] RETURN

Example:

50 GOSUB 70 60 PRINT "* SUBROUTINE OVER *" 65 END	Passes control to subroutine at line 70.
70 PRINT "SUBROUTINE STARTS" 80 PRINT "MORE SUBROUTINE" 90 PRINT "ENDING SUBROUTINE"	Subroutine begins.
100 RETURN RUN SUBROUTINE STARTS MORE SUBROUTINE ENDING SUBROUTINE * SUBROUTINE OVER *	Ends subroutine and passes control back to the line following GOSUB, line 60, which executes only after the subroutine is over.

STOP

This statement terminates program execution and returns control to command level, also called direct mode. You can resume execution with the CONT statement if you follow the restrictions detailed in the description of CONT.

[*linenumber*] STOP

SYS

Use this statement to call a machine language subroutine. This subroutine is located at the jumpaddress named as the SYS argument. This address is decimal, not hexidecimal.

SYS jumps to the last bank named in the program. If no bank has been named, SYS jumps to bank 15. If SYS jumps to any bank other than 15, RAM-loaded transfer of execution routines must be present in the bank.

NOTE: All machine language programs must end with an RTS (ReTurn from Subroutine) statement, which returns to the BASIC program.

[*linenumber*] SYS *jumpaddress*

jumpaddress is the decimal address of the machine language subroutine being called by the program.

Example:

40 SYS 512	Calls the machine language subroutine at decimal address 512.

TRAP

This statement prevents BASIC's normal error handling functions from taking control. When an error occurs, TRAP lets your program perform its own error handling routines that you've written into the program. Three error-handling functions, EL, ER, and ERR$, are explained in Appendix A.

[*linenumber*] *TRAP* [*linenumber2*]

linenumber2 is the line where your error handling procedures begin.

Example:

```
360 INPUT B
370 IF B = 0 THEN TRAP 550
380 X = A/B: PRINT X
```

If B = 0, an error occurs because BASIC won't divide by 0. TRAP passes to line 550 where this error is fixed without the program being stopped because of the error.

WAIT

WAIT suspends program execution while monitoring the status of data input from the specified location. The values of selected bits at the specified location determine whether the WAIT statement is re-executed, or control passes to the next executable statement.

When you use the WAIT statement, the program is on hold, waiting until a machine address you name develops a specific bit pattern. The data read at the address is exclusive Ored with mask2, whose default is 0. Then the data is ANDed with mask1. If the result is zero, BASIC loops back to reread the data, making execution WAIT. If the result of the OR and AND operations is not zero, execution continues with the next executable statement.

NOTE: If you enter an indefinite loop with a WAIT statement, you must manually reset the machine.

[*linenumber*] WAIT *location, mask1* [*,mask2*]

mask1 is the value with which the specified data is ANDed.
mask2 is the value with which the specified data is exclusive ORed.

Example:

```
55 PRINT "PROGRAM WAITS TIL ANY
    KEY IS PRESSED"
60 POKE 209,0
```

Puts 0 in memory location 209.

```
70 WAIT 209,1
```

Makes program wait until any key is pressed before resuming.

```
80 PRINT "SOME KEY WAS PRESSED"
```

APPENDICES

APPENDIX A

BASIC 4.0 FUNCTIONS

ABS

```
ABS (expression)
```

Returns the absolute value of (expression).

Example.

```
PRINT   ABS (7*(-5))
 35
```

ASC

```
ASC (expression)
```

Returns the numeric value that represents the ASCII code of the first character of (expression). which is a string value. The CHR$ function performs ASCII-to-string conversion.

Example.

```
10 X$ = "TEST"
20 PRINT ASC (X$)
RUN
 84                          T is ASCII code 84.
```

ATN

```
ATN (expression)
```

Returns the arctangent of the (expression) in radians. The result is in the range -pi /2 to pi /2. The expression can be any numeric type, but the evaluation of ATN is always performed in floating point binary.

Example.

```
10 INPUT X
20 PRINT ATN (X)
RUN
? 3
 1.24904577
```

CHR$

CHR$ (expression)

Returns a string containing a single character whose value is the character with the ASCII code represented by (expression). These codes are listed in Appendix D. The expression can be any integer between 0 and 255.

CHR$ is often used to send a special character to the terminal. For example, CHR$(14) switches the screen to upper /lower case (normal) mode.

The ASC function performs ASCII-to-numeric conversion.

Examples:

```
10 E$ + CHR$ (147) + "ERROR MESSAGE"
20 PRINT E$: REM CLEARS SCREEN AND PRINTS MESSAGE

10 N$ = CHR$(83) + CHR$(77)
20 PRINT N$
RUN
SM
```

COS

COS (expression)

Returns the cosine of (expression) in radians. Expression is any valid numeric expression. The evaluation of COS is always performed in floating point binary.

Example:

```
PRINT COS(5-1)
 -.65364362

10 X = 2*COS(.4)
20 PRINT X
RUN
 1.84212199
```

ERR$

ERR$ (*expression*)

Returns a character string which contains the text of the error message represented by (*expression*). The value of expression must be between 0 and 127.

When used with the TRAP statement, ERR$ helps you process error messages within your program.

Example:

```
35   REM IF USED WITH TRAP EL HOLDS THE ERROR LINE
     WHILE ER HOLDS THE ERROR #
50   PRINT ERR$(1):REM THIS WILL PRINT AN ERROR
     MESSAGE
70   TRAP 110:REM GO TO LINE 110 IF AN ERROR OCCURS
80   PRINT VAL(K):REM THIS IS AN ERROR
90   PRINT "WE HAVE RETURNED FROM OUR TRAP
     ROUTINE"
100  END
110  PRINT "ERROR IN LINE"EL: REM PRINT THE LINE WITH
     THE ERROR
120  PRINT "THE ERROR IS "ERR$(ER) : REM DISPLAY THE
     ERROR
130  RESUME NEXT:REM RESUME EXECUTION AFTER LINE
     WITH ERROR IN IT
```

EXP

EXP (*expression*)

Returns the value of e (approx. 2.71828183) raised to the power represented by (*expression*). Expression must be less than or equal to 88.02969191.

Examples:

```
?EXP(1)
 2.71828183

?EXP(3.5)/2
 16.557726

?EXP(89)

?OVERFLOW
```

FRE

FRE (*expression*)

Returns the number of free bytes in a memory segment or bank indicated by (*expression*). If you have a 128K machine. banks 1 and 2 contain 64K each. and the other banks are empty. If you have a 256K machine. banks 1. 2. 3. and 4 contain 64K each. and the other banks are empty.

Example:

```
?FRE(1)
 63908

?FRE(1) + FRE(2)
 128095
```

INSTR

INSTR (*expression1,expression2*[*,expression3*])

The INSTR function performs a substring search. The text of string (*expression 1*) is searched. beginning at character position (*expression3*). for the occurrence of string (*expression2*). Numeric expression3 must be a value between 1 and 255. The default for expression 3 is 1.

INSTR returns these values:

- If expression 2 is NOT found in expression1. INSTR returns zero (0).

- If expression2 is found. INSTR returns the position in string expression1 that contains the first character of expression2.

Example:

```
10  A$ = "TEST TEXT"
20  B$ = "TEXT"
30  PRINT B$;"TEXT STARTS AT CHAR";INSTR(A$,B$)
RUN
TEXT STARTS AT CHAR 6
```

INT

INT (*expression*)

Returns the largest integer which is less than or equal to the value of (*expression*).

Example:

```
PRINT INT (99.89)
 99
PRINT INT ( -28.8)
 -29
```

LEFT$

LEFT$ (*expression1*, *expression2*)

Returns a string that consists of a number (*expression2*) of characters from a string (*expression1*) starting from the leftmost character in (*expression1*). Expression2 must be an integer between 1 and 255.

If expression2 is greater than the length of expression1, then the LEFT$ function returns the entire string. Use the LEN function to find the length of expression1.

Example:

```
10   A$ = "COMMODORE COMPUTERS"
20   B$ = LEFT$(A$,9)
30   PRINT B$
RUN
COMMODORE
```

LEN

LEN (*expression*)

Returns the number of characters in (*expression*). Non-printing characters and blanks are counted.

Example:

```
10  X$ = "COMMODORE COMPUTERS"
20  PRINT LEN(X$)
RUN
 18
```

LOG

LOG (*expression*)

Returns the natural logarithm of (*expression*). Expression must be greater than zero.

Example:

```
PRINT LOG (45/7)
 1.86075234
```

MID$

MID$ (*expression1*, *expression2* [,*expression3*])

Returns a string that contains a number (*expression3*) of characters from string (*expression1*), starting at the character position, named in (*expression2*). Expression2 and expression3 must be between 1 and 255.

If you do not supply a value for expression3 or if there are fewer than expression3 characters in the string expression1, then the MID$ function returns all of the rightmost characters of expression1, beginning with the expression2 character.

If you specify a value for expression2 that is greater than the length of the string expression1, then the MID$ function returns a null string.

Example:

```
10   A$ = "GOOD"
20   B$ = "MORNING EVENING, FRIENDS"
30   PRINT A$;MID$(B$,9)
40   PRINT A$;MID$(B$,9,7)
RUN
GOOD EVENING, FRIENDS
GOOD EVENING
```

PEEK

PEEK (*expression*)

Returns the byte read from memory location (*expression*) in the bank selected by a previously executed BANK instruction. Expression must be between 0 and 65535.

PEEK is the complementary function to the POKE statement. See the POKE statement for more information.

Example:

```
20  PRINT PEEK (36879)
RUN
 46
```

POS

POS (*expression*)

Returns the column number of the current cursor position. The leftmost position is 0: the rightmost position is 80. *Expression* is a dummy argument, which means that you can give it any value because it doesn't affect the function evaluation.

Example:

```
50  IF POS(X) > 60 THEN PRINT CHR$(13)
60  REM CHR$(13) IS THE RETURN KEY
```

RIGHT$

RIGHT$ (*expression1*,*expression2*)

Returns a string that consists of a number (*expression2*) of characters from a string (*expression1*) starting from the rightmost character in expression1. Expression2 must be an integer between 1 and 255.

If expression2 is greater than the length of expression1, the RIGHT$ function will return the entire string. You can use the LEN function to see how long expression1 is.

If expression2 is zero, then RIGHT$ returns the null string. A null string is a string with a length of zero.

The LEFT$, MID$, and RIGHT$ string handling functions and the INSTR function can be used to perform complicated string handling operations.

Example:

```
10  T$ = "BEGINNING,MIDDLE, AND END OF TEXT"
20  E$ = RIGHT$(T$,3):REM E$ = 3 RIGHTMOST CHARS OF
    T$,
30  IF E$ <> "END" THEN PRINT RIGHT$(A$,8)
40  REM CHECKS IF 3 RIGHTMOST CHARS = END;
    IF NOT, PRINTS 8 RIGHTMOST
RUN
 OF TEXT.
```

RND

RND (*expression*)

Returns a random number between 0 and 1. *Expression* is the seed value.

Example:

```
10   FOR A = 1 to 5
20   PRINT INT (RND(X)*100)
30   NEXT A
RUN
 24   30   31   51   5
```

SGN

SGN (*expression*)

Returns a value that indicates whether the value of (*expression*) is positive, negative, or zero. The SGN function values are:

- For X > 0. SGN returns +1
- For X = 0. SGN returns 0
- For X < 0. SGN returns -1

Example:

```
10   ON SGN(X) + 2 GOTO 75, 125, 180
20   REM IF X < 0 GOES TO 75; IF X = 0 GOES TO 125
30   REM IF X > 0 GOES TO 180
```

SIN

SIN (*expression*)

Returns the sine of (*expression*) in radians.
Example:

```
PRINT SIN(1.5)
 .997494987
```

SPC

SPC (*expression*)

Prints the number of blank spaces on the screen (or printer, if opened) indicated by the number in (*expression*). SPC can only be used with PRINT. Expression must be between 0 and 155.

Example:

```
PRINT "TOTAL SALES"; SPC(15);X
TOTAL SALES                    12345.67
```

SQR

SQR (*expression*)

Returns the square root of (*expression*). Expression must be greater than or equal to zero.

Example:

```
PRINT 10, SQR(10)
10           3.16227766
```

STATUS

Status

Returns a completion STATUS for the last input /output operation which was performed on an open file. The STATUS can be read from any peripheral device.

The value of the status function depends on the operation and device checked.

Use the STATUS function to:

- check for errors during the processing of a program on disk
- see if you are at the end of a file during the read processing
- check on a verify operation

A table of STATUS code values for printer, disk (IEEE peripherals) and RS-232 file operations is shown below:

ST Bit Position	ST Numeric Value	IEEE Bus	RS-232C Channel*
0	1	time out write	parity error (receive only)
1	2	time out read	framing error (receive only)
2	4		overrun (receive only)
3	8		
4	16		input buffer empty
5	32		DCD error
6	64	EOI	DSR error
7	-128	device not present	

*Meaning when bit is set to 1.

STR$

STR$ (*expression*)

Returns a string representation of the value of (*expression*).

Example:

```
PRINT "$" + STR$(2.77)
```
Prints $2.77

```
          or
PRINT"$";STR$(2.77)            Prints $2.77
PRINT STR$(150) + ".00"        Prints 150.00
```

TAB

TAB (*expression*)

Positions the cursor in the column represented by (*expression*). You can only use TAB with a PRINT statement. Expression must be between 0 and 155. The first column on the screen is column 0.

Example:

```
PRINT "TOTAL"; TAB(29);"123456"
TOTAL                            123456
```

TAN

TAN (*expression*)

Returns the tangent of (*expression*) in radians.

Example:

```
10  X = .785398163
20  Y = TAN(X)
30  PRINT Y
RUN
 1
```

TI$

TI$

Returns the internal interval timer as a character string. The string contains seven characters showing hours, minutes, seconds, and tenths of seconds (hhmmsst). Set the timer with this statement:

```
10  TI$ = "0000000"
```

USR

USR (*expression*)

Calls the user written machine language subroutine which has starting address stored in locations 3 and 4 of bank 15. The argument (*expression*) is stored in the floating point accumulator prior to entering the subroutine.

VAL

VAL (*expression*)

Returns the numeric value of the string (*expression*). The STR$ function performs the complementary task, numeric to string conversion.

Example:

```
30  IF VAL(ZIP$)<90000 OR VAL(ZIP$)>96699 THEN
40  PRINT "OUT OF STATE"
```

RESERVED SYSTEM VARIABLES

AND	Logical operator.
DS$	Disk status reserved word.
EL	Line number last error occurred.
ER	Error# of last error occurrence.
OR	Logical operator.
NOT	Logical operator.
STatus	The system status for the last Input /Output operation.
TI$me	The character string representation of the current time-of-day registers.

RESERVED SYSTEM SYMBOLS

+ Plus sign	arithmetic addition or string concatenation
- Minus sign	arithmetic subtraction and unary minus
* Asterisk:	arithmetic multiplication
/ Slash:	arithmetic division

(blank) Blank:	separates keywords and variable names
= Equal sign:	value assignment and relationship testing
< Less than	used in relationship testing
> Greater than:	used in relationship testing
↑ Up arrow:	arithmetic exponentiation
. Comma:	used in variable lists to format output: also separates command parameters
. Period:	decimal point in floating point constants
: Semicolon:	used in variable lists to format output
: Colon:	separates multiple BASIC statements on a program line
" Quotation mark:	encloses string constants
? Question mark:	abbreviation for the keyword PRINT
(Left parenthesis:	expression evaluation and functions
) Right parenthesis:	expression evaluation and functions
% Percent:	declares a variable name as an integer
# Number:	comes before the logical file number in input / output statements
$ Dollar sign:	declares a variable name as a string
π Pi:	the numeric constant 3.14159265

APPENDIX B

BASIC 4.0 ABBREVIATIONS

KEYWORD	ABBREVIATION	TYPE
ABS	a SHIFT B	function—numeric
APPEND	a SHIFT P	statement
ASC	a SHIFT S	function—numeric
ATN	a SHIFT T	function—numeric
BACKUP	b SHIFT A	command
BANK	ba SHIFT N	statement
BLOAD	b SHIFT L	command
BSAVE	b SHIFT S	command
CHR$	c SHIFT H	function—string
CATALOG	c SHIFT A	command
CLOSE	cl SHIFT O	statement
CLR	c SHIFT L	statement
CMD	c SHIFT M	statement
COLLECT	co SHIFT L	command
CONCAT	con SHIFT C	statement
CONT	c SHIFT O	command
COPY	co SHIFT P	command
COS	none	function—numeric
DATA	d SHIFT A	statement
DCLEAR	none	command
DCLOSE	d SHIFT C	statement
DEF FN	d SHIFT E	statement
DELETE	de SHIFT L	command
DIM	d SHIFT I	statement
DIRECTORY	di SHIFT R	command

DISPOSE	di SHIFT S	statement
DLOAD	d SHIFT L	command
DOPEN	d SHIFT O	statement
DSAVE	d SHIFT S	command
END	e SHIFT N	statement
ERR$	none	function—string
EXP	e SHIFT X	function—numeric
FOR	f SHIFT O	statement
FRE	f SHIFT R	function—numeric
GET	g SHIFT E	statement
GET#	none	statement
GOSUB	go SHIFT S	statement
GOTO	g SHIFT O	statement
HEADER	h SHIFT E	command
IF...GOTO	none	statement
IF...THEN...ELSE	none	statement
INPUT	none	statement
INPUT#	i SHIFT N	statement
INSTR	in SHIFT S	function—numeric
INT	none	function—numeric
KEY	k SHIFT E	command
LEFT$	le SHIFT F	function—string
LEN	none	function—numeric
LET	l SHIFT E	statement
LIST	l SHIFT I	command
LOAD	l SHIFT O	command
LOG	none	function—numeric
MID$	m SHIFT I	function—string
NEW	none	command
NEXT	n SHIFT E	statement
ON...GOSUB	none	statement
ON...GOTO	none	statement
OPEN	o SHIFT P	statement
PEEK	p SHIFT E	function—numeric
POKE	p SHIFT O	statement
POS	none	function—numeric
PRINT	?	statement
PRINT#	p SHIFT R	statement
PRINT USING	?us SHIFT I	statement
PUDEF	none	statement
READ	r SHIFT E	statement

RECORD	re SHIFT C	statement
REM	none	statement
RENAME	re SHIFT N	command
RESTORE	re SHIFT S	statement
RESUME	res SHIFT U	statement
RETURN	re SHIFT T	statement
RIGHT$	r SHIFT I	function—string
RND	r SHIFT N	function—numeric
RUN	r SHIFT U	command
SAVE	s SHIFT A	command
SCRATCH	s SHIFT C	command
SGN	s SHIFT G	function—numeric
SIN	s SHIFT I	function—numeric
SPC	s SHIFT P	function—special
SQR	s SHIFT Q	function—numeric
STATUS	st	function—numeric
STOP	s SHIFT T	statement
STR$	st SHIFT R	function—string
SYS	s SHIFT Y	statement
TAB	t SHIFT A	function—special
TAN	none	function—numeric
TI$	none	function—string
TRAP	t SHIFT R	statement
USR	u SHIFT S	function—special
VAL	none	function—numeric
VERIFY	v SHIFT E	command
WAIT	w SHIFT A	statement

NOTE: The character printed is the same in normal (text) mode and graphics mode unless otherwise indicated.

APPENDIX C

SCREEN DISPLAY CODES

SET 1	SET 2	POKE
@		0
A	a	1
B	b	2
C	c	3
D	d	4
E	e	5
F	f	6
G	g	7
H	h	8
I	i	9
J	j	10
K	k	11
L	l	12
M	m	13
N	n	14
O	o	15
P	p	16
Q	q	17
R	r	18
S	s	19
T	t	20
U	u	21
V	v	22
W	w	23
X	x	24
Y	y	25
Z	z	26
[		27
£		28
]		29
↑		30
←		31
SPACE		32
!		33
"		34
#		35
$		36
%		37
&		38
'		39
(		40
)		41
*		42
+		43
,		44
−		45
.		46
/		47
0		48
1		49
2		50
3		51
4		52
5		53
6		54
7		55
8		56
9		57
:		58
;		59
<		60
=		61
>		62

SET 1	SET 2	POKE
?		63
─		64
♠	A	65
│	B	66
─	C	67
─	D	68
▔	E	69
▁	F	70
▏	G	71
▕	H	72
╮	I	73
╰	J	74
╯	K	75
└	L	76
╲	M	77
╱	N	78
┌	O	79
┐	P	80
●	Q	81
▁	R	82
♥	S	83
▏	T	84
╭	U	85
╳	V	86
○	W	87
♣	X	88
▕	Y	89
♦	Z	90
┼		91
▒		92
│		93
π	▒	94
◥	▧	95
SPACE		96
▌		97
▄		98
▔		99
▁		100
▏		101
▒		102
▕		103
▒		104
◤	▨	105
▕		106
├		107
▗		108
└		109
┐		110
▁		111
┌		112
┴		113
┬		114
┤		115
▎		116
▍		117
▐		118
▔		119
▀		120
▃		121
┘	✓	122
▖		123
▝		124
┘		125
▘		126
▚		127

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

APPENDIX D

CHR$ CODE

				PRINTS			PRINTS		
PRINTS	CHR$	PRINTS	CHR$	TEXT	GRAPHICS	CHR$	TEXT	GRAPHICS	CHR$
	0		23	.	.	46	e	E	69
	1		24	/	/	47	f	F	70
	2		25	0	0	48	g	G	71
	3		26	1	1	49	h	H	72
CE	4	**ESC**	27	2	2	50	i	I	73
	5		28	3	3	51	j	J	74
	6		29	4	4	52	k	K	75
BELL	7		30	5	5	53	l	L	76
	8		31	6	6	54	m	M	77
TAB	9	**SPACE**	32	7	7	55	n	N	78
	10	!	33	8	8	56	o	O	79
	11	"	34	9	9	57	p	P	80
	12	#	35	:	:	58	q	Q	81
RETURN **ENTER**	13	$	36	;	;	59	r	R	82
SWITCH TO TEXT MODE	14	%	37	<	<	60	s	S	83
WINDOW TOP	15	&	38	=	=	61	t	T	84
	16	'	39	>	>	62	u	U	85
CRSR ↓	17	(	40	?	?	63	v	V	86
RVS ON	18	)	41	@	@	64	w	W	87
HOME	19	*	42	a	A	65	x	X	88
DEL	20	+	43	b	B	66	y	Y	89
	21	,	44	c	C	67	z	Z	90
	22	–	45	d	D	68	[	[	91

NOTE: The character printed is the same in normal (text) mode and graphics mode unless otherwise indicated.

PRINTS	CHR$
£	92
]	93
↑	94
←	95
SEE NOTE BOTTOM OF PAGE	
	128
	129
	130
RUN	131
CLEAR ENTRY	132
	133
	134
	135
	136
TAB	137
	138
	139
	140
SHIFT **RETURN**	141
SWITCH TO GRAPHICS MODE	142
WINDOW BOTTOM	143
	144
↑ CRSR	145
RVS OFF	146
CLR	147

PRINTS	CHR$
INST	148
	149
	150
	151
	152
	153
	154
	155
	156
← CRSR	157
	158
	159
SPACE	160
▌	161
▄	162
▔	163
▁	164
▏	165
▒	166

PRINTS		
TEXT	GRAPHICS	CHR$
	▕	167
	▒	168
▨	◤	169
	▐	170
	├	171

PRINTS		
TEXT	GRAPHICS	CHR$
	▗	172
	└	173
	┐	174
	▂	175
	┌	176
	┴	177
	┬	178
	┤	179
	▎	180
	▍	181
	▐	182
	▔	183
	▀	184
	▃	185
√	┘	186
	▖	187
	▝	188
	┘	189
	▘	190
	▚	191
	─	192
A	♠	193
B	│	194
C	─	195
D	─	196
E	─	197
F	─	198

PRINTS		
TEXT	GRAPHICS	CHR$
G	│	199
H	│	200
I	╮	201
J	╰	202
K	╯	203
L	⌞	204
M	╲	205
N	╱	206
O	⌜	207
P	⌝	208
Q	●	209
R	▁	210
S	♥	211
T	▏	212
U	╭	213
V	╳	214
W	○	215
X	♣	216
Y	▕	217
Z	♦	218
	┼	219
	▒	220
	│	221
π	π	222
▧	◥	223
SEE NOTE BOTTOM OF PAGE		

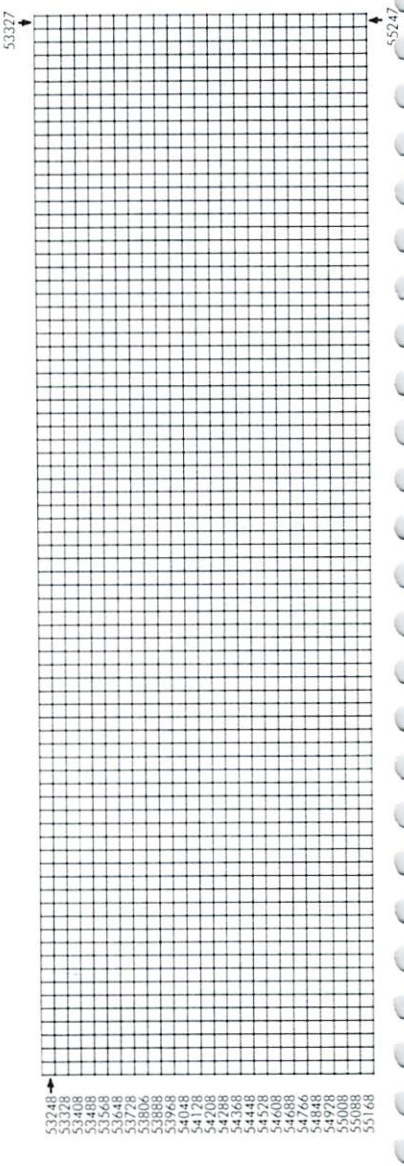
CODES 96-127 ARE THE SAME AS 32-63
CODES 224-254 ARE THE SAME AS 160-190
CODE 255 IS THE SAME AS 222

APPENDIX E

SCREEN MEMORY MAP

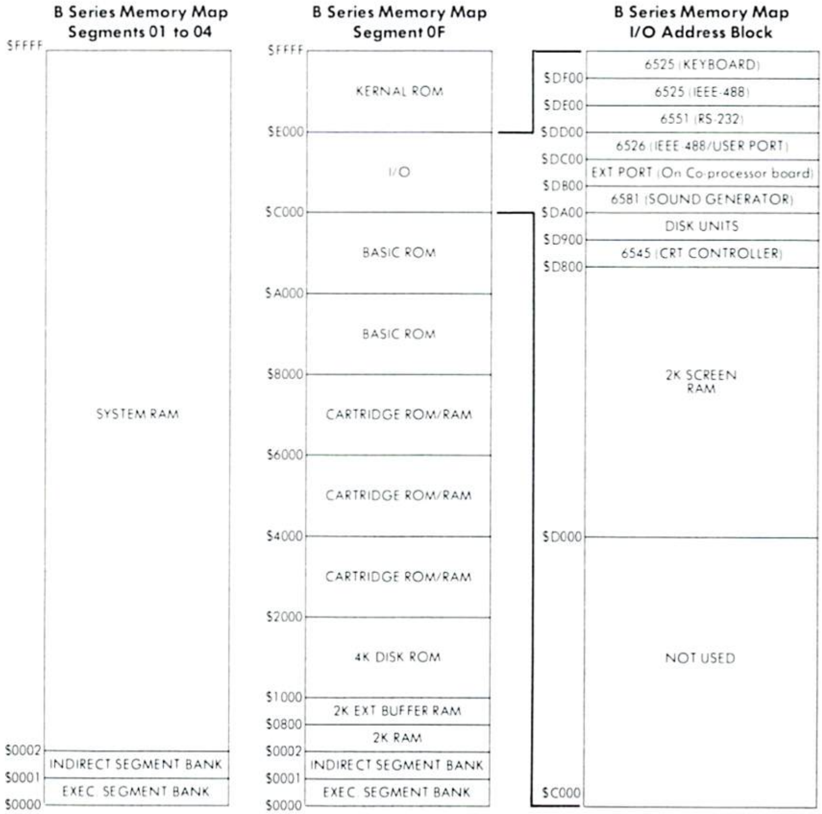
Your computer's memory stores the characters currently displayed on the screen and automatically updates changes. Your 'B' Series computer screen has 25 lines by 80 columns, so it has positions for 2000 characters. Each of these positions has its own screen memory address by which you can refer to the screen position and the character currently located there. You can access a specific location by supplying the address in PEEK and POKE statements. PEEKs let you see what is in a screen memory location, and POKEs let you put a value into a screen memory location.

Each character position is represented by one byte, starting at hexadecimal address D000 (decimal 53248) and ending at hexadecimal address D7CF (decimal 55247).



APPENDIX F

B SERIES MEMORY MAP



APPENDIX G

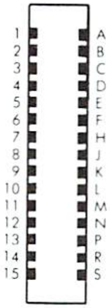
MATHEMATICAL FUNCTIONS TABLE

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 contangent	arccoth(x) = log((x + 1) /(x - 1)) /2

APPENDIX H

PINOUTS FOR INPUT/OUTPUT DEVICES

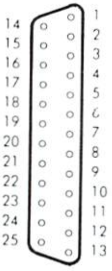
Your computer is equipped with several specialized chips all in BANK 15. The 6526 Complex Interface Adapter is located at 56320 ($DC00). The 6551 Asynchronous Communications Interface Adapter is located at 56576 ($DD00). Your computer has two 6525 Tri-port Interface chips located at 56832 ($DE00) and 57088 ($DF00). For more information, consult your *Programmer's Reference Guide.*



Connector Pin—Outs

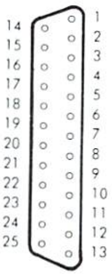
Pin	Type	Pin	Type
1	A0	A	BD0
2	A1	B	BD1
3	A2	C	BD2
4	A3	D	BD3
5	A4	E	BD4
6	A5	F	BD5
7	A6	H	BD6
8	A7	J	BD7
9	A8	K	GND
10	A9	L	GND
11	A10	M	SR/W
12	A11	N	S02
13	A12	P	NOT CSBank 1
14	+5 VDC	R	NOT CSBank 2
15	+5 VDC	S	NOT CSBank 3

Keyboard Connector



Pin	Type	Pin	Type
1	PA0	2	PA2
3	PA4	4	PA6
5	PB0	6	PB1
7	PB2	8	PB3
9	PB4	10	PB5
11	PB6	12	PB7
13	PC5	14	PA1
15	PA3	16	PA5
17	PA7	18	PC0
19	PC1	20	PC2
21	PC3	22	GND
23	GND	24	GND
25	PC4		

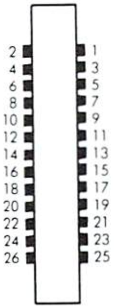
RS 232C Connector



Pin	Type
1	SHIELD
2	T × D
3	R × D
4	RTS
5	CTS
6	DSR
7	GND
8	DCD
11	+5 VDC
18	-12 VDC
20	DTR
24	R × C

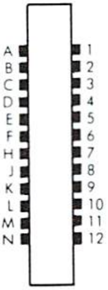
All others N.C.

User Connector



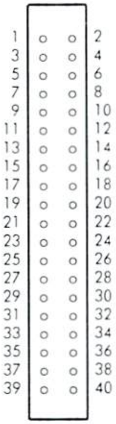
Pin Type	Pin Type
1 - GND	2 - PB2
3 - GND	4 - PB3
5 - NOT PC	6 - NOT FLAG
7 - 2D7	8 - 2D6
9 - 2D5	10 - 2D4
11 - 2D3	12 - 2D2
13 - 2D1	14 - 2D0
15 - 1D7	16 - 1D6
17 - 1D5	18 - 1D4
19 - 1D3	20 - 1D2
21 - 1D1	22 - 1D0
23 - NOT CNT	24 - +5 VDC
25 - NOT IRQ	26 - SP

IEEE Connector



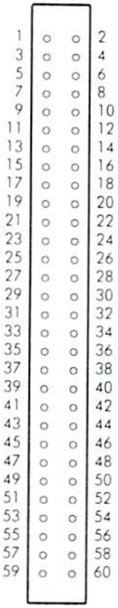
Pin	Type	Pin	Type
1	D1	A	D5
2	D2	B	D6
3	D3	C	D7
4	D4	D	D8
5	EOI	E	REN
6	DAV	F	GND
7	NRFD	H	GND
8	NDAC	J	GND
9	IFC	K	GND
10	SRQ	L	GND
11	ATN	M	GND
12	SHIELD	N	GND

Co-Processor Connector



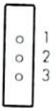
Pin	Type	Pin	Type
1	EXTMA3	2	DRAMO0
3	EXTMA2	4	DRAMO1
5	EXTMA7	6	DRAMO2
7	EXTMA6	8	DRAMO3
9	EXTMA5	10	DRAMO4
11	EXTMA4	12	DRAMO5
13	EXTMA1	14	DRAMO6
15	EXTMA0	16	DRAM07
17	GND	18	GND
19	GND	20	GND
21	GND	22	NOT BUSY1
23	GND	24	NOT P2REFREQ
25	GND	26	NOT P2REFGRNT
27	GND	28	BP0
29	GND	30	BP1
31	GND	32	BP2
33	N.C.	34	BP3
35	NOT-PROCRES	36	NOT BUSY2
37	EXTBUFR/W	38	NOT ERAS
39	DRAM R/W	40	NOT ECAS

Expansion Connector



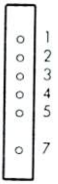
Pin	Type	Pin	Type
1	+5 VDC	2	+5 VDC
3	+5 VDC	4	+5 VDC
5	GND	6	GND
7	GND	8	GND
9	GND	10	GND
11	NOT BRAS	12	IRQ3
13	-12 VDC	14	NOT EXTRES
15	+12 VDC	16	NOT S.O.
17	NOT RES	18	LPEN
19	SR/W	20	NOT EXTBUFCS
21	TODCLK	22	NOT DISKROMCS
23	BOOTCLK	24	N.C.
25	S02	26	NOT BCAS
27	S01	28	NOT CS1
29	BD3	30	NOT EXTPRTCS
31	BD4	32	BD2
33	BD5	34	BD1
35	DB7	36	BD0
37	BA13	38	BD7
39	BA14	40	BA15
41	BA1	42	BA0
43	BA2	44	BA11
45	BA3	46	BA10
47	BA12	48	BA4
49	BA9	50	BA5
51	BA8	52	BA6
53	BP0	54	BA7
55	BP1	56	BP2
57	NOT NMI	58	BP3
59	RDY	60	NOT IRQ

Audio Jack



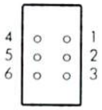
Pin	Type
1	TO SPEAKER
2	N.C.
3	TO SPEAKER

Video Connector



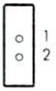
Pin	Type
1	VIDEO
2	GND
3	VERTICAL SYNC
4	GND
5	HORIZONTAL SYNC
6	KEY
7	GND

Power Connector



Pin	Type
1	50/60 Hz
2	-12 VDC
3	+12 VDC
4	GND
5	GND
6	+5 VDC

Reset Connector



Pin	Type
1	TO RESET SWITCH
2	TO RESET SWITCH

APPENDIX I

CONVERTING FROM STANDARD BASIC TO EXTENDED BASIC 4.0

If you have programs written in a BASIC other than Commodore BASIC, some minor adjustments may be necessary before running them with Commodore BASIC. Here are some specific things to look for when converting BASIC programs.

String Dimensions

Delete all statements that are used to declare the length of strings. A statement such as DIM A$(I, J), which dimensions a string array for J elements of length I, should be converted to the Commodore BASIC statement DIM A$(J).

Some BASICs use a comma or ampersand for string concatenation. Each of these must be changed to a plus sign, which is the operator for Commodore BASIC string concatenation.

In Commodore BASIC, the MID$, RIGHT$, and LEFT$ functions are used to take substrings of strings. Forms such as A$(I) to access the "Ith" character in A$, or A$(I, J) to take a substring of A$ from position I to position J, must be changed as follows:

Other BASIC	Commodore BASIC
A$(I) = X$	A$ = LEFT$(A$, I - 1) + X$ + MID$(A$, I + 1)
A$(I,J) = X$	A$ = LEFT$(A$, I - 1) + X$ + MID$(A$,J + 1)

Multiple Assignments

Some BASICs allow statements of the form:

```
10 LET B = C = 0
```

to set B and C equal to zero. Commodore BASIC would interpret the second equal sign as a logical operator and set B equal to -1 if C equaled 0. Instead, convert this statement to two assignment statements:

```
10 C = 0:B = 0
```

Multiple Statements

Some BASICs use a backslash to separate multiple statements on a line. With Commodore BASIC, be sure all statements on a line are separated by a colon.

MAT Functions

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

Differences From Older Commodore BASIC

TI references must be changed. The current smallest unit of time is 1 /10 sec. rather than 1 /60 sec. TI$ now has seven characters instead of six. The seventh character is tenths of seconds. ER is now a reserved variable. All references must be changed to use a new variable name. ER returns the error number (127 is no error).

EL is now a reserved variable. All references must be changed to use a new variable name. EL returns the line number of the last error (65535 is no error).

APPENDIX J

ERROR MESSAGES

# MESSAGE	EXPLANATION
0. ?stop key detected	Occurs when doing a KERNAL I /O function and the STOP key is pressed. May occur during LOAD or SAVE (or OPEN, CLOSE, GET#, INPUT#, PRINT#). Disk files are not damaged.
1. ?too many files	You are trying to OPEN more than 10 files at a time. Decrease the number of OPEN or DOPEN files by CLOSING them.
2. ?file open	An attempt was made to redefine file parameter information by repeating an OPEN command on the same file twice.
3. ?file not open	The operating system must have information provided by the OPEN statement. If an attempt is made to read or write a file without having done this previously, then this message appears.
4. ?file not found	The named file specified in OPEN or LOAD was not found on the device specified.

5. ?device not present	No device on the IEEE was present to handshake an attention sequence. May happen on OPEN, CLOSE, CMD, INPUT#, GET#, PRINT#. If filename is not specified with OPEN, this error will not occur.
8. ?missing filename	LOADs and SAVEs from the IEEE port (e.g., the disk) require a filename to be specified. Supply the filename.
9. ?illegal device number	Occurs if you try to access a device in an illegal manner. For example, LOADing or SAVING on the keyboard, screen, or RS-232.
10. are you sure ?	This is a prompt for BACKUP, SCRATCH, and HEADER. It is not an error message and should not occur during BASIC program execution.
11. ?bad disk	Media failure on HEADER command.
14. break	This occurs when the STOP key is pressed during normal BASIC execution. The CONTinue command can be used to restart the program.
15. extra ignored	Too many items of data or separators (,) were typed in response to an INPUT statement. Only the first few items were accepted.
16. redo from start	Is not actually a fatal error printed in the standard format but is a diagnostic which is printed when

	data in response to INPUT is nonnumeric where a numeric quantity is required. The INPUT continues to function until acceptable data has been received.
20. ?next without for	Either a NEXT is improperly nested or the variable in a NEXT statement corresponds to no previously executed FOR statement.
21. ?syntax error	BASIC cannot recognize the statement you have typed. Caused by such things as missing parentheses, illegal characters, incorrect punctuation, misspelled keyword.
22. ?return without gosub	A RETURN statement was encountered without a previous GOSUB statement being executed.
23. ?out of data	A READ statement was executed but all of the data statements in the program have been read. The program tried to read too much data, or insufficient data was included in the program. Carriage returning through a line READY on the B Series video display yields this error because the message is interpreted as READ Y.
24. illegal quantity	Occurs when a function is accessed with a parameter out of range caused by: 1. A matrix subscript out of range ($0 < X < 32767$)

2. LOG (negative or zero argument)
3. SQR (negative argument)
4. A↑B where A < 0 and B not integer.
5. Call of USR before machine language subroutine has been patched in.
6. Use of string functions MID$, LEFT$, RIGHT$, with length parameters out of range (1 < X < 255).
7. Index ON . . . GOTO out of range.
8. Addresses specified for PEEK, POKE, WAIT, and SYS out of range (1 < X < 255).
9. Byte parameters of WAIT, POKE, TAB and SPC out of range (0 < X < 255).

25. overflow

Numbers resulting from computations or input that are larger than binary 1.70141184E + 38 cannot be represented in BASIC's number format. Underflow is not a detectable error but numbers less than binary 2.93873587E-39 are indistinguishable from zero.

26. ?out of memory

May appear while entering or editing a program as the text completely fills memory. At run time, assignment and creation of variables may also fill all variable memory. Array available declarations consume large areas of memory even though a program may be rather short. The maximum number of FOR loops and simultaneous GOSUBs are dependent on each other. This context is stored on the microprocessor hardware stack whose capacity may be exceeded. To determine the type of memory error, examine the results of FRE. If there is a large number of bytes available, it is most likely a FOR-NEXT or GOSUB problem. A subroutine which terminates in GOTO rather than RETURN will eventually cause an out of memory error as stack pointers build up.

27.?undefined statement	An attempt was made to GOTO, GOSUB, or THEN to a statement which does not exist.
28. ?bad subscript	An attempt was made to reference a matrix element which is outside the dimensions of the matrix. This may happen by specifying the wrong number of dimensions or a subscript larger than specified in the original dimension.
29. ?redim'd array	After an array was dimensioned, another dimension statement for the same array was encountered. For example, an array variable is defined by default when it is first used, and later a DIM statement is encountered.
30. ?division by zero	Zero as a divisor would result in numeric overflow-thus it is not allowed. When this message appears, it is most expedient to list

	the statement and look for division operators.
31. ?illegal direct	A single buffer area is used by BASIC to process incoming characters. This same buffer is used to hold a statement that is being interpreted in direct mode. INPUT will not work because incoming characters would overwrite the variable list following INPUT to be processed. DEF cannot be used in direct mode for a different but similar reason. The name of a function is stored in the BASIC variable area with pointers to the string of characters which define the function. Since the function exists only in the input buffer, it is wiped out the first time a NEW command is typed in.
32. ?type mismatch	The left-hand side of an assignment statement was a numeric variable and the right-hand side was a string, or vice versa; or a function which expected a string argument was given a numeric one, or vice versa.
33. ?string too long	Attempt by use of the concatenation operator to create a string more than 255 characters long.
34. ?file data	Occurs when an INPUT# statement finds a string while attempting to read a numeric value.
35. ?formula too complex	This indicates that BASIC has run

	out of string temporary pointers to keep track of substrings in evaluating a string expression. Break the string expression into two smaller parts to cure the problem.
37. ?undefined function	Reference was made to a user defined function which had never been defined.
39. ?verify error	The contents of memory and a specified file do not compare.
40. ?out of stack	Too many levels of FOR . . . NEXT or GOSUBs have been executed. No recovery possible.
41. ?unable to resume	A fatal error has occurred, such as running out of stack.
42. ?unable to dispose	All of the DISPOSE type items have been disposed of or none exist.
43. ?out of text	If any LOAD or DLOAD exceeds the end of the text bank of (64K) this error will result. This error will not occur when using the BLOAD command.
44. ?cannot continue	The CONT command will not work because the program was never RUN, there has been an error, or a line has been edited.

APPENDIX K

NONERROR MESSAGES

The messages listed below are available through the ERROR MESSAGE code numbers by using the ERR$ calling codes listed next to each message. However, these messages are *not* Error Messages so they *will not appear* on the screen unless you specifically call for them in your programming or call for them as a standard operating procedure.

MESSAGE	EXPLANATION
12. (carriage return) ready (carriage return)	This message lets you know that your system is ready to use.
13. (space) in (space)	This message is similar to ready.
17. your last "evaluated" number	This is the last number that has been evaluated through the numerical output buffer. (e.g., print 10*10: if you use an ERS code 17, the number on your screen will equal the last evaluation—in this case, 100.)
18. more (carriage return)	
19. power on message	***COMMODORE BASIC 128, V4.0*** ***COMMODORE BASIC 256, V4.0***

APPENDIX L

6581 (SID) CHIP REGISTER MAP

The 6581 Sound Interface Device is located starting at location 55808 ($DA00). Below is a brief register map. For detailed information, consult the *Programmer's Reference Guide.*

REG # (DEC)	ADDRESS A_4	A_3	A_2	A_1	A_0	REG # (HEX)	DATA D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REG NAME	REG TYPE
															Voice 1	
0	0	0	0	0	0	00	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ LO	WRITE ONLY
1	0	0	0	0	1	01	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ HI	WRITE ONLY
2	0	0	0	1	0	02	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW LO	WRITE ONLY
3	0	0	0	1	1	03	—	—	—	—	PW_{11}	PW_{10}	PW_9	PW_8	PW HI	WRITE ONLY
4	0	0	1	0	0	04	NOISE	⎍⎍	◿◿	⋀⋀	TEST	RING MOD	SYNC	GATE	CONTROL REG	WRITE ONLY
5	0	0	1	0	1	05	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	WRITE ONLY
6	0	0	1	1	0	06	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	SUSTAIN/RELEASE	WRITE ONLY
															Voice 2	
7	0	0	1	1	1	07	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ LO	WRITE ONLY
8	0	1	0	0	0	08	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ HI	WRITE ONLY
9	0	1	0	0	1	09	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW LO	WRITE ONLY
10	0	1	0	1	0	0A	—	—	—	—	PW_{11}	PW_{10}	PW_9	PW_8	PW HI	WRITE ONLY
11	0	1	0	1	1	0B	NOISE	⎍⎍	◿◿	⋀⋀	TEST	RING MOD	SYNC	GATE	CONTROL REG	WRITE ONLY
12	0	1	1	0	0	0C	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	WRITE ONLY
13	0	1	1	0	1	0D	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	SUSTAIN/RELEASE	WRITE ONLY
															Voice 3	
14	0	1	1	1	0	0E	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ LO	WRITE ONLY
15	0	1	1	1	1	0F	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ HI	WRITE ONLY
16	1	0	0	0	0	10	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW LO	WRITE ONLY
17	1	0	0	0	1	11	—	—	—	—	PW_{11}	PW_{10}	PW_9	PW_8	PW HI	WRITE ONLY
18	1	0	0	1	0	12	NOISE	⎍⎍	◿◿	⋀⋀	TEST	RING MOD	SYNC	GATE	CONTROL REG	WRITE ONLY
19	1	0	0	1	1	13	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	WRITE ONLY
20	1	0	1	0	0	14	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	SUSTAIN/RELEASE	WRITE ONLY
															Filter	
21	1	0	1	0	1	15	—	—	—	—	—	FC_2	FC_1	FC_0	FC LO	WRITE ONLY
22	1	0	1	1	0	16	FC_{10}	FC_9	FC_8	FC_7	FC_6	FC_5	FC_4	FC_3	FC HI	WRITE ONLY
23	1	0	1	1	1	17	RES_3	RES_2	RES_1	RES_0	FILTEX	FILT 3	FILT 2	FILT 1	RES/FILT	WRITE ONLY
24	1	1	0	0	0	18	3 OFF	HP	BP	LP	VOL_3	VOL_2	VOL_1	VOL_0	MODE/VOL	WRITE ONLY
															Misc.	
25	1	1	0	0	1	19	PX_7	PX_6	PX_5	PX_4	PX_3	PX_2	PX_1	PX_0	POT X	READ ONLY
26	1	1	0	1	0	1A	PY_7	PY_6	PY_5	PY_4	PY_3	PY_2	PY_1	PY_0	POT Y	READ ONLY
27	1	1	0	1	1	1B	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0	OSC_3/RANDOM	READ ONLY
28	1	1	1	0	0	1C	E_7	E_6	E_5	E_4	E_3	E_2	E_1	E_0	ENV_3	READ ONLY

APPENDIX M

PRINTER COMMANDS

6400 Word Processor Printer/ 8023P CBM Bi-Directional Printer

COMMAND	SYNTAX	FUNCTION
OPEN	OPEN lfn,dn,(sa)	sets correspondence between file number and physical device. The lfn or logical file number may be any number from 1 to 255. The dn or device number refers to the device you wish to send the file to. The sa or secondary address alerts the printer's microprocessor system that formatting is to occur.
CMD	CMD lfn	transfers control from computer to printer. The lfn must be the same as that in the OPEN statement. When you give the CMD command, the printer prints READY and is awaiting further commands. The CMD command followed by a PRINT or LIST command directs the output to the printer.
PRINT#	PRINT# lfn, data	PRINT# works like PRINT except that output is directed to the printer instead of video. Using the CMD command opens a "listening" channel to

		the printer, and when followed by a PRINT# command, the connection between the printer and computer is shut down or is said to be "unlistening".
CLOSE	CLOSE lfn	You should always close a file after printing from it. You may not exceed ten open files so you should close files when you are finished with them.

APPENDIX N

USING THE RS-232C CHANNEL

The OPEN statement for an RS-232C channel has some special arguments that you must understand before you can use it. You must match the operating parameters of the RS-232C interface to those of the device you're connecting to the computer.

When you open the RS-232C channel, your OPEN statement must look like this:

OPEN *filenumber,2,secondary-address,openstring*

Where:

filenumber is the logical file number to be associated with the RS-232C channel.

secondary-address determines the direction of the RS-232C channel. It can be input, output, or bidirectional and may or may not convert between CBM and ASCII character codes.

openstring is a *four-byte* command string that establishes the operating parameters for the RS-232C channel.

The *secondary-address* may take any of the values shown in Table 8.1.

TABLE 8.1 RS-232C DIRECTIONAL SECONDARY ADDRESSES

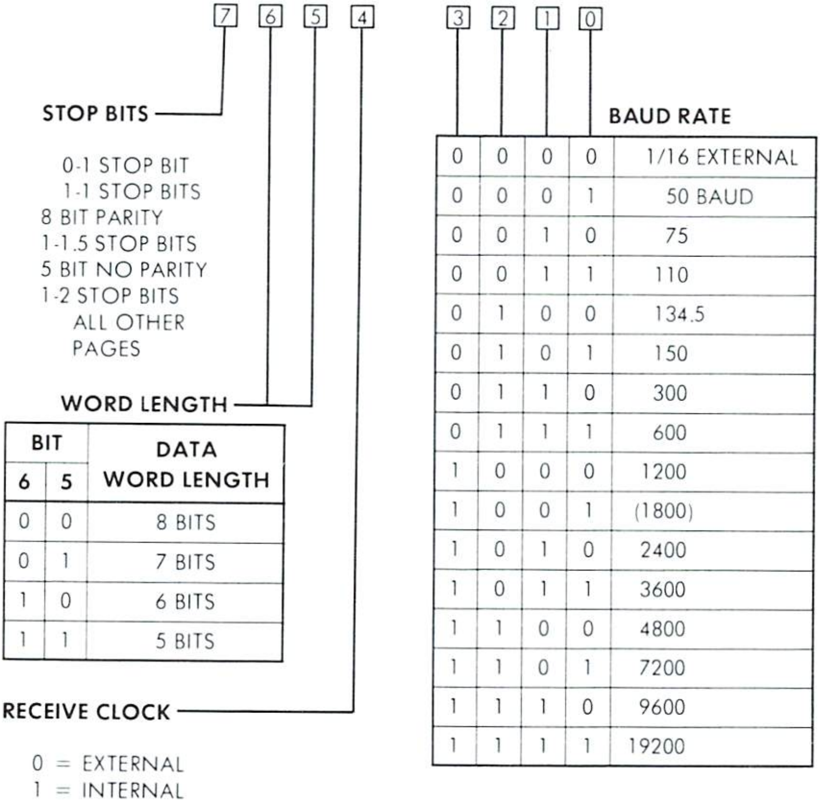
VALUE	MEANING
1	open an output channel
2	open an input channel
3	open an input /output channel

129	open an output channel and convert CBM and ASCII character codes
130	open an input channel and convert ASCII to CBM character codes
131	open an input /output channel and convert between CBM and ASCII character codes

The *secondary-address* values 1, 2, and 3 do not perform character conversions. If you're getting ASCII character codes through the RS-232C channel, they are delivered as-is to your program. If you want CBM /ASCII conversion you must select a *secondary-address* value of 129, 130, Or 131.

NOTE: If you are transmitting or receiving non-character data through your RS-232C interface, do NOT request CBM/ASCII character conversion. This will completely scramble your data.

The *openstring* for the RS-232C interface is four bytes long. The first two bytes contain detailed control information. The last two aren't used, but you must include them.



STOP BITS

0-1 STOP BIT
1-1 STOP BITS
8 BIT PARITY
1-1.5 STOP BITS
5 BIT NO PARITY
1-2 STOP BITS
ALL OTHER
PAGES

WORD LENGTH

BIT		**DATA WORD LENGTH**
6	**5**	
0	0	8 BITS
0	1	7 BITS
1	0	6 BITS
1	1	5 BITS

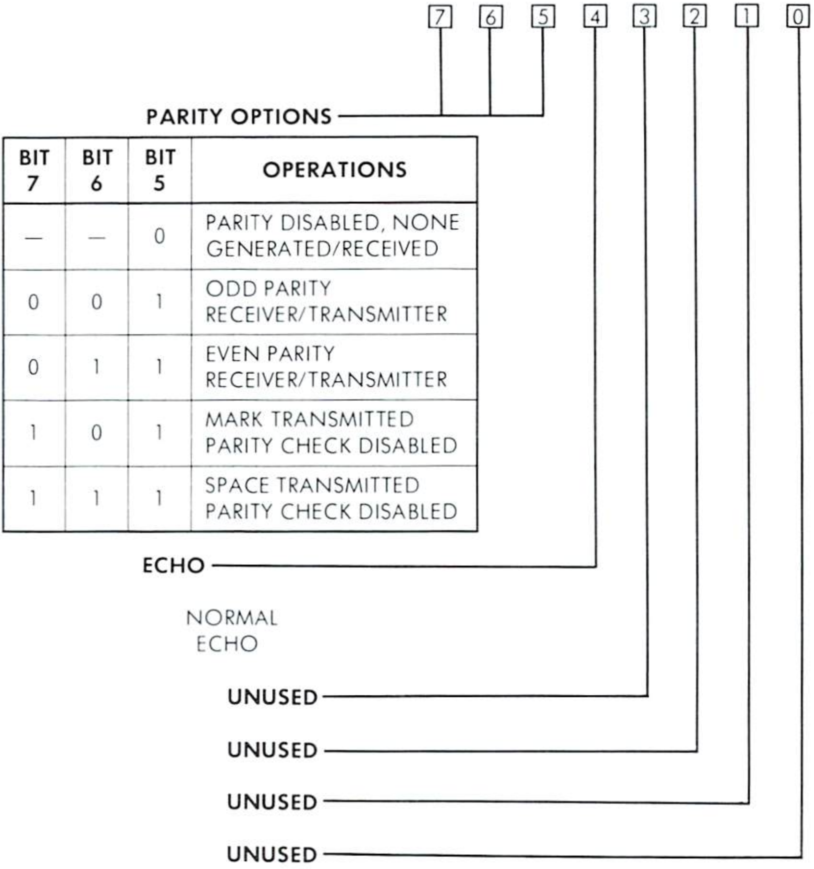
RECEIVE CLOCK

0 = EXTERNAL
1 = INTERNAL

BAUD RATE

0	0	0	0	1/16 EXTERNAL
0	0	0	1	50 BAUD
0	0	1	0	75
0	0	1	1	110
0	1	0	0	134.5
0	1	0	1	150
0	1	1	0	300
0	1	1	1	600
1	0	0	0	1200
1	0	0	1	(1800)
1	0	1	0	2400
1	0	1	1	3600
1	1	0	0	4800
1	1	0	1	7200
1	1	1	0	9600
1	1	1	1	19200

First Byte Open String
RS-232C



BIT 7	BIT 6	BIT 5	OPERATIONS
—	—	0	PARITY DISABLED, NONE GENERATED/RECEIVED
0	0	1	ODD PARITY RECEIVER/TRANSMITTER
0	1	1	EVEN PARITY RECEIVER/TRANSMITTER
1	0	1	MARK TRANSMITTED PARITY CHECK DISABLED
1	1	1	SPACE TRANSMITTED PARITY CHECK DISABLED

NORMAL
ECHO

Second Byte Open String

APPENDIX O

MACHINE LANGUAGE MONITOR

TIM is the Terminal Interface Monitor program for MOS Technology's 6500 Series microprocessors. It has been expanded and adapted to function on the B Series computers. Execution is transferred from the CBM BASIC interpreter to TIM by the SYS command. The monitor is incorporated as part of the Kernal.

Commands typed on the CBM keyboard can direct the TIM to start executing a program, display or modify registers and memory locations, load or save binary data, view other segments, send disk commands or read status, set default disk unit and load and execute programs by entering the program name (Segment 15 only). On modifying memory, TIM NO LONGER performs automatic read after write verification to insure that the addressed memory exists, and is R/W type.

TIM COMMANDS

M	Display memory
:	Alter memory
R	Display registers
:	Alter registers
G	Begin execution
L	Load
S	Save
V	View Segment
U	Set default disk unit
@	Send disk command or get disk status
X	Exit to basic
Z	Transfer to second microprocessor

<file name> load and execute

EXAMPLES

M *DISPLAY MEMORY*

```
M 0000 0010
: 0000 0f 0f 4c d9 9a 00 00 00 00 00 00 00 22 22 9e 00
: 0010 00 00 00 00 00 00 00 d4 fb 04 00 04 00 00 c4 fb
```

In a display memory command, the start and ending addresses must be completely specified as 4 digit hex numbers. To alter a memory location, move the cursor up in the display, type the correction and press **RETURN** to enter the change. When you move the cursor to a line and press **RETURN**, the colon tells the monitor that you are re-entering data.

R *DISPLAY REGISTERS*

```
R
  PC     IRQ    SR    AC    XR    YR    SP
;0007    FBF8   B0    DD    71    04    71
```

The registers are saved and restored upon each entry or exit from the TIM. They may be modified or preloaded as in the display memory example above. The semicolon tells the monitor you are modifying the registers.

G *BEGIN EXECUTION*

```
G 0200
```

The GO command may have an optional address for the target. If none is specified, the PC from the R command is taken as the target.

L *LOAD*

```
L "filename",08
```

No defaults are allowed on a load command. The device number and the file name must be completely specified. Operating system prompts for operator intervention are the same as for BASIC. Memory addresses are loaded as specified in the file header which is set up by the SAVE command. Machine language subroutines may be loaded from BASIC but care must be taken not to use BASIC variables as the variable pointer is set to the last byte loaded + 1. The machine language subroutine will be loaded into the segment that you are currently in as determined by the V command. After the load, the system will be initialized back to segment 15.

S *SAVE*
S "*filename*",08,010200,010300

As in the load command, no defaults are allowed in the SAVE command. The device number, file name and a six byte start and end address must be given. The above example will save a program to device 8 from segment #1 starting at 0200 hex and ending at 0300 hex. The first two bytes are the segment number followed by the address. Valid segment bytes may be 0 and 0F depending on your memory. After a save, the system will be initialized back to segment 15.

V *VIEW*
V 01

This will change the segment to the one that you wish to view, save, load or change memory from. The valid segments are 00 to 0F

U *UNIT ADDRESS*
U 09

This command will allow you to set the disk unit default address while you are in the monitor. When leaving, the original address is reset. Valid unit addresses are 8 to 1F. These must be entered in HEX.

@ READ ERROR CHANNEL AND PROCESS DISK COMMANDS

@	:Display error message and clear channel
@ S1:filename	:Scratch specific file from drive 1
@ I0	:Initialize disk in drive 0
@ R0:newname = oldname	:Rename file on drive 0
@ C1:filename = oldname	:Copy file from drive 0 to drive 1
@ V0	:Validate or collect disk in drive 0
@ N1:filename,id	:New or Header disk in drive 1

The above examples use the same syntax as the wedge program supplied with the disk drives.

<file name> LOAD AND EXECUTE FILE IN SEGMENT 15

This will load a machine language program from the disk and execute it. Its use is restricted to segment 15.

Z *TRANSFER TO SECOND MICROPROCESSOR*

```
Z
```

This command will allow you to utilize the 8088 when applicable.

X *EXIT TO BASIC*

```
X
```

This will cause a warm start to BASIC. In a warm start, memory is not altered in any way and BASIC resumes operation the way it was before the call to the monitor was made.

APPENDIX P

BIBLIOGRAPHY

PUBLISHER	TITLE /AUTHOR
Addison Wesley	*BASIC and the Personal Computer*, Dwyer and Critchfield
Compute	*Compute's First Book of PET/CBM*
Cowbay Computing	*Teacher's PET—Plans, Quizzes and Answers*
	Feed Me, I'm Your PET Computer, Carol Alexander
	Looking Good With Your PET, Carol Alexander
Creative Computing	*Getting Acquainted With Your VIC-20*, T. Hartnell
Dilithium Press	*BASIC Basic-English Dictionary for the Pet*, Larry Noonan
Faulk Baker Associates	*MOS Programming Manual*, MOS Technology
Hayden Book Co.	*BASIC Conversions Handbook: Apple, TRS 80, and PET*, Brain, Oviatt, Paquin, and Stone
	Library of PET Subroutines, Nick Hampshire

PUBLISHER	TITLE /AUTHOR
	PET Graphics. Nick Hampshire
	I Speak BASIC to my PET. Aubrey Jones, Jr.
	BASIC from the Ground Up. David E. Simon
Howard W. Sams	*Mostly BASIC Applications for Your PET*. Howard Berenbon
	PET Interfacing. J. Downey and S. Rogers
	Crash Course in Microcomputers. Louise Frenzol
Little, Brown and Co.	*Computer Games for Businesses, Schools and Homes*. J. Victor Nagigian and William S. Hodges
	The Computer Tutor: Learning Activities for Homes and Schools. Gary W. Orwig and William S. Hodges
McGraw Hill	*Home and Office Use of VisiCalc*. D. Castlewitz and L. Chisauki
	Hands-On BASIC with a PET. Herbert D. Peckman
Osborne /McGraw Hill	*Pet/CBM Personal Computer Guide*. Carroll S. Donahue
	Osborne CP/M User Guide. Thom Hogan

PUBLISHER	TITLE /AUTHOR
	PET FUN AND GAMES. R. Jeffries and G. Fisher
	PET and the IEEE. A. Osborne and C. Donahue
	Some Common Basic Programs. Lon Poole and Mary Borchers
	The 8086 Book. Russell Rector and George Alexy
P.C. Publications	*Beginning Self-Teaching Computer Lessons*
Prentice-Hall, Inc.	*The PET Personal Computer for Beginners*. S. Dunn and V. Morgan
Reston Publishing Co.	*Pet and the IEEE 488 Bus (GPIB)*. Eugene Fisher and C.W. Jensen
	PET BASIC. Richard Huskell.
	PET Games and Recreation. Ogelsvy, Lindsey, and Kunkin
	PET BASIC—Training Your PET Computer. Zamora, Carvie, and Albrecht
Total Information Services	*Understanding Your PET/CBM: Vol. 1 BASIC Programming*
	Understanding Your VIC. David Schultz

APPENDIX Q

USER'S CLUBS, MAGAZINES, AND THE COMMODORE INFORMATION NETWORK

Commodore wants you to know that our support for users is just beginning with your purchase of a Commodore computer. That's why we've created two publications with Commodore information from around the world, and a "two-way" computer information network full of valuable input by and for Commodore computer users in the U.S. and Canada from coast to coast.

In addition, we wholeheartedly encourage and support the growth of *Commodore User's Clubs* all over the globe. They are an excellent source of information for every Commodore computer user, from the beginner to the most experienced.

The magazines and network, which are described below, have the most up-to-date information on how to get involved with the User's Club in your area.

Furthermore, your local Commodore dealer is an excellent source of Commodore support and information. Your dealer can always provide literature and hardware support to fill your changing computing needs.

Power/Play: The Home Computer Magazine

When it comes to entertainment, learning at home, and practical home applications, *Power/Play* is the prime source of information for Commodore computer owners. It directs you to the User's Club nearest you and tells you about its activities. It describes software, games, programming techniques, telecommunications, and new products. *Power/Play* is your personal connection to other Commodore users, outside software and hardware developers, and to Commodore itself. Published quarterly, it's only $10.00 for a whole year of home computing excitement.

Commodore:The Microcomputer Magazine

Widely read by educators, business people, and students, as well as home computerists. *Commodore* is our main vehicle for sharing exclusive information on the more technical uses of Commodore systems. Regular departments cover the business, science, and education fields, programming tips, technical tips, and many other features of interest to anyone who uses, or is thinking about purchasing, Commodore equipment. *Commodore* is the ideal complement to *Power/Play*. It is published bi-monthly, and a subscription costs only $15.00 per year.

Commodore Information Network

The magazine of the future is here today. To supplement your subscriptions to *Power/Play* and *Commodore* magazines, the *Commodore Information Network*—our "paperless magazine"—is available now. All you need is a Commodore computer, a telecommunications device called a modem, and your home or business telephone.

Join our computer club, get help with a computing problem, "talk" to other Commodore friends, or get up-to-the-minute information on new products, software, and educational resources. Soon you will even be able to save yourself the trouble of typing in the program listings you find in *Power/Play* and *Commodore* by "downloading" directly from the *Information Network*. The best part of the network is that most of the answers to your questions are there before you even ask them. How's that for service?

To "call" our electronic magazine you only need a modem and subscription to CompuServe™, one of the nation's largest telecommunications networks.

Just dial your local number for the CompuServe™ data bank nearest you and then connect your phone to the modem. When the CompuServe™ video text appears on your screen, type "G CBM" on your keyboard. When the *Commodore Information Network's* table of contents, or "menu," appears on the screen, it's your turn to choose from one of our 16 departments. So make yourself comfortable, and enjoy the "paperless magazine" that all the other magazines are writing about.

For more information about the *Commodore Information Network* or about CompuServe™, visit your local Commodore dealer or contact CompuServe™ customer service at 1-800-848-8990 (in Ohio, 614-457-8600).

COMMODORE INFORMATION NETWORK

Main Menu Description
Direct Access Codes
Special Commands
User Questions
Public Bulletin Board
Magazines and Newsletters
New Product Announcements
Commodore New Direct
Commodore Dealers
Educational Resources
User Groups
Descriptions
Questions and Answers
Software Tips
Technical Tips
Directory Descriptions

INDEX

Index

"B" SERIES QUICK REFERENCE CARD

SIMPLE VARIABLES

Type	*Name*	*Range*
Real	XY	± 1.70141183E + 38
		± 2.93873588E − 39
Integer	XY%	± 32767
String	XY$	0 to 255 characters

X is a letter (A-Z), Y is a letter or number (0-9). Variable names can be more than 2 characters, but only the first two are recognized.

ARRAY VARIABLES

Type	*Name*
Single Dimension	XY(5)
Two-Dimension	XY(5,5)
Three-Dimension	XY(5,5,5)

Arrays of up to eleven elements (subscripts 0-10) can be used where needed. Arrays with more than eleven elements need to be DIMensioned.

ALGEBRAIC OPERATORS

= Assigns value to variable
− Negation
↑ Exponentiation
* Multiplication
/ Division
+ Addition
− Subtraction

RELATIONAL AND LOGICAL OPERATORS

=	Equal
<>	Not Equal To
<	Less Than
>	Greater Than
<=	Less Than or Equal To
>=	Greater Than or Equal To
NOT	Logical "Not"
AND	Logical "And"
OR	Logical "Or"

Expression equals 1 if true, 0 if false.

SYSTEM COMMANDS

DLOAD "NAME"	Loads a program from disk
DSAVE "NAME"	Saves a program on disk
LOAD "NAME",8	Loads a program from disk
SAVE "NAME",8	Saves a program to disk
VERIFY "NAME"	Verifies that program was SAVEd without errors
RUN	Executes a program
RUN xxx	Executes program starting at line xxx
STOP	Halts execution
END	Ends execution
CONT	Continues program execution from line where program was halted
PEEK(X)	Returns contents of memory location X
POKE X,Y	Changes contents of location X to value Y
SYS xxxxx	Jumps to execute a machine language program, starting at xxxxx
WAIT X, Y, Z	Program waits until contents of location X, when EORed with Z and ANDed with Y, is nonzero
USR(X)	Passes value of X to a machine language subroutine

EDITING AND FORMATTING COMMANDS

LIST	Lists entire program
LIST A-B	Lists from line A to line B
REM Message	Comment message can be listed but is ignored during program execution
TAB(X)	Used in PRINT statements. Spaces X positions on screen
SPC(X)	PRINTs X blanks on line
POS(X)	Returns current cursor position
CLR/HOME	Positions cursor to left corner of screen
SHIFT CLR/HOME	Clears screen and places cursor in "Home" position
SHIFT INS/DEL	Inserts space at current cursor position
INS/DEL	Deletes character at current cursor position
CTRL	Prints graphics on non-alphabetic keys and accesses control functions
CRSR Keys	Moves cursor up, down, left, right on screen
Commodore Key	Stops the program from scrolling. Press any key to restart.

ARRAYS AND STRINGS

DIM A (X, Y, Z)	Sets maximum subscripts for A; reserves space for (X + 1)*(Y + 1)*(Z + 1) elements starting at A (0,0,0)
LEN (X$)	Returns number of characters in X$
STR$(X)	Returns numeric value of X, converted to a string
VAL(X$)	Returns numeric value of A$, up to first nonnumeric character
CHR$(X)	Returns ASCII character whose code is X
ASC(X$)	Returns ASCII code for first character of X$
LEFT$(A$,X)	Returns leftmost X characters of A$
RIGHT$(A$,X)	Returns rightmost X characters of A$
MID$(A$,X,Y)	Returns Y characters of A$ starting at character X

INPUT/OUTPUT COMMANDS

INPUT A$ OR A	PRINTs "?" on screen and waits for user to enter a string or value
INPUT "ABC";A	PRINTs message and waits for user to enter value. Can also INPUT A$
GET A$ or A	Waits for user to type one-character value; no RETURN needed
DATA A, "B", C	Initializes a set of values that can be used by READ statement
READ A$ or A	Assigns next DATA value A$ or A
RESTORE	Resets data pointer to start READing the DATA list again
PRINT "A = ";A	PRINTs string "A = " and value of A ";" suppresses spaces "," tabs data to next field.

PROGRAM FLOW

GOTO X	Branches to line X
IF A = 3 THEN 10	IF assertion is true THEN execute following part of statement. IF false, execute next line number
FOR A = 1 TO 10	Executes all statements between FOR
STEP 2 : NEXT	and corresponding NEXT, with A going from 1 to 10 by 2. Step size is 1 unless specified.
NEXT A	Defines end of loop. A is optional
GOSUB 2000	Branches to subroutine starting at line 2000
RETURN	Marks end of subroutine. Returns to statement following most recent GOSUB
ON X GOTO A,B	Branches to Xth line number on list. If X = 1 branches to A, etc.
ON X GOSUB A,B	Branches to subroutine at Xth line number in list

OWNER'S REGISTRATION CARD

Please mail this card to Commodore to register your computer with us.

Name: ____________________

Address: ____________________

City: ____________________

Zip Code: ____________________

____

1. What is your family's present income bracket?
   - ☐ less than $14,999
   - ☐ $15,000 - $24,999
   - ☐ $25,000 - $39,999
   - ☐ $40,000 - $59,999
   - ☐ $60,000 and above

2. Purchaser's age?
   - ☐ Under 18
   - ☐ 18 - 24
   - ☐ 25 - 34
   - ☐ 35 - 49
   - ☐ over 50

3. Are you male or female?

   ☐ Male ☐ Female

4. Are you married?

   ☐ Yes ☐ No

5. Number of Children? ____________________

6. What's your educational background?
   - ☐ Did not finish high school
   - ☐ High School graduate
   - ☐ Some College
   - ☐ College Graduate
   - ☐ Some Graduate School
   - ☐ Graduate Degree

7. What is your primary area of computing interest?
   - ☐ Self teaching
   - ☐ Education
   - ☐ Recreation and Hobby
   - ☐ Small business
   - ☐ Telecommunications/Timesharing
   - ☐ Engineering
   - ☐ Productivity
   - ☐ Other ____________________

Get the most out of your Commodore computer with a subscription to Commodore's user magazines

POWER/PLAY

Fun, Games and Beyond with Commodore Home Computers

Published quarterly in March, June, September and December, POWER/PLAY is devoted solely to the exciting and rapidly expanding world of Commodore home computing. It provides valuable information on new products, applications, games, programming techniques, learning-at-home, telecommunications and just about anything else Commodore home computer users need to know to get maximum enjoyment out of their home computing experience. Subscription price: $10.00/year.

FILL OUT AND MAIL TODAY

Name ______________________ Phone __________

Address ________________________________

City ______________ State _________ Zip _________

Computer model: ___________________________

☐ Address Change. Enter new address above & enclose present mailing label
☐ Renewal subscription
☐ New subscription

commodore

The Microcomputer Magazine

Widely read by educators, businessmen, students and home computerists, this bi-monthly publication provides a vehicle for sharing exclusive product information on Commodore systems, programming techniques, hardware interfacing, and applications for the wide range of Commodore's products. Each issue contains features of interest to anyone that uses, or is thinking about purchasing Commodore equipment. Get the most out of your microcomputer with Commodore Magazine. Subscription price: $15.00/year.

GET MORE INFORMATION FOR YOUR MONEY

Please sign me up for:

____year(s) of POWER/PLAY at $10.00/year

____year(s) of COMMODORE at $15.00/year

Canadian and Foreign: POWER/PLAY $15.00/year; COMMODORE $25.00/year

Enclosed is my check or money order for $ ____________

Make check or money order payable to:

COMMODORE BUSINESS MACHINES, INC.
The Meadows, 487 Devon Park Drive, Wayne, PA 19087

LIMITED 90-DAY WARRANTY
COMMODORE PERSONAL COMPUTER SYSTEMS

Commodore Business Machines, Inc. ("Commodore") warrants to the original consumer purchaser that its Personal computer products ("UNIT") (*) (Not including computer programs on cassettes or disks) shall be free from any defect in material and workmanship for a period of 90 days from the date of purchase. If a defect covered by this warranty occurs during this 90 day warranty period, you should return the UNIT within such 90 days to:

> Your original dealer or any Full Service Commodore dealer together with a copy of your sales slip or similar proof-of-purchase. The dealer will repair the defective UNIT under this warranty.

In the unlikely event that your dealer is unable to repair UNIT or you need assistance in locating a Full Service Dealer you may, if necessary, contact the Commodore Customer Support Group at (215) 436-4200.

This warranty does not cover damage or malfunctions resulting from improper handling, accident, misuse, abuse, failure of electrical power, use with other products not manufactured or approved by Commodore, damage while in transit for repairs, repairs attempted by any unauthorized person or agency, or any other reason not due to defects in materials or workmanship. This warranty is also void if the serial number has been altered, defaced, or removed.

ANY IMPLIED WARRANTIES ARISING OUT OF THE SALE OF THIS UNIT INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO THE ABOVE NINETY (90) DAY PERIOD. COMMODORE'S LIABILITY IS LIMITED SOLELY TO THE REPAIR OR REPLACEMENT OF THE DEFECTIVE UNIT IN ITS SOLE DISCRETION, AND IN NO EVENT SHALL INCLUDE DAMAGES FOR LOSS OF USE OR OTHER INCIDENTAL OR CONSEQUENTIAL COSTS, EXPENSES, OR DAMAGES INCURRED BY THE PURCHASER, INCLUDING WITHOUT LIMITATION ANY DATA OR INFORMATION WHICH MAY BE LOST OR RENDERED INACCURATE, EVEN IF COMMODORE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

All computer programs, whether sold by Commodore or others, are distributed on an "AS IS" basis without warranty of any kind. The entire risk as to the performance and suitability of such programs is with the purchaser.

Should the programs (on cassettes or disks) prove defective following their purchase, the purchaser and not the manufacturer, distributor, or retailer assumes the full responsibility for service or replacement.

Commodore shall have no liability or responsibility to a purchaser, customer, or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by any computer programs (on any media) sold by Commodore or others. This includes but is not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer programs.

Commodore shall have no obligation to enhance or update any UNIT once manufactured.

Some states do not allow limitations on how long any implied warranty lasts or exclustion of consequential damages, so the above limitation or exclustion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

(*) UNITS COVERED UNDER THIS WARRANTY ARE:
ALL SERIES- 2000, 4000, 8000, 9000 UNITS Peripherals and their Accessories.
ALL SERIES -'C', 'P', 'B', 'BX' UNITS Peripherals and their Accessories.

commodore
COMPUTER

COMMODORE 'B' SERIES ADVANCED BUSINESS MACHINES

THE PRACTICAL, VERSATILE BUSINESS SYSTEM

Commodore's versatile 'B' Series business microcomputers provide powerful computing systems for your most important business needs: word processing, record keeping, accounting, database management, and a variety of other applications. The microcomputers in this series offer state-of-the-art technology and superior features:

- 128K or 256K RAM
- 8-bit or 16-bit microprocessor
- Optional tilt and swivel monitor
- 94-key keyboard
- 20 programmable function keys
- Separate 19-key calculator keypad
- Expandable memory
- 80 column by 25 line screen display
- Extended BASIC version 4.0 + (66 commands)
- Compatible with Commodore business peripherals

This manual describes 'B' Series system features, software applications, technical and BASIC programming information. Your Commodore dealer can provide additional up-to-date information on 'B' Series compatible peripherals and software.

commodore
COMPUTERS

Commodore Business Machines, Inc.
1200 Wilson Drive • West Chester, PA 19380

Commodore Business Machines, Limited
3370 Pharmacy Avenue • Agincourt, Ontario, M1W 2K4