Service

Your TRACKSTAR system has been designed, constructed, and tested to meet exacting quality standards. With reasonable handling it should give many years of trouble free service. The electronics do not require periodic maintenance under normal operating conditions.

PREVENTIVE MAINTENANCE

Equipment which is subjected to adverse operating environments should have periodic preventive maintenance by a qualified service technician. Contact your dealer or Diamond Computer Systems, Inc. for names of authorized repair and service agencies in your area.

TROUBLE SHOOTING

In the event the equipment does not operate correctly, use the following troubleshooting chart to determine the source of the problem and the solution. If you are unable to resolve the problem, contact your dealer or the Diamond Computer Systems Technical Service Department (see Below).

TROUBLESHOOTING PROBLEMS WITH THE TRACKSTAR

SYMPTOM	PROBLEM	SOLUTION
Does not boot	power	Attach power cable to computer. Plug into wall outlet.
Disk drive Does not run	power	Check connections. Turn on power to computer.
	data cable	Check connection
	interface board	**TURN OFF EQUIPMENT** **DISCONNECT POWER** Check for proper seating of the TRACKSTAR board in the slot.
Does not boot	no diskette	Insert Boot diskette in Drive A
Disk Drive runs continuously	Door Open	Insert diskette fully, turn latch.
	No DOS	Insert working copy of required disk operating system to boot system.
Wrong Drive Boots	Apple Drive not connected.	Check connection on Apple Drive.

TROUBLESHOOTING PROBLEMS WITH THE TRACKSTAR

SYMPTOM	PROBLEM	SOLUTION
No screen image	power	Check power to video monitor
	video cable	Check video cables and connections: Color or B/W board to TRACKSTAR, TRACKSTAR to Monitor.
Snow or garbage	connections	Check video connections and cables. Replace if necessary.
No color	monitor	Requires color monitor or RGB monitor.
Peripherals do not work	interface cards	Check to see printers are connected on IBM. Check to see cards are configured properly.

TECHNICAL ASSISTANCE

The Technical Service Department at Diamond Computer System, Inc. can answer questions pertinent to the installation, operation, and maintance of your TRACKSTAR. Please supply the model number and serial number of your system, equipment configuration, and the particular software or application you are using, and specific details of your requirement or problem. Diamond Computer Systems will endeavor to provide you with a prompt and complete response.

Write or telephone us at:

Diamond Computer Systems, Inc.
Technical Services
3380 Montgomery Drive
Santa Clara, CA. 95054
Phone: (408) 986-0100

WARRANTY REPAIR

Your 1-year warranty is printed at the beginning of this manual. Study it carefully to ensure that the stated conditions are met, and provide the appropriate information in writing to your dealer or Diamond Computer System Technical Services at the above address. You must supply a copy of a valid receipt, and prepay all shipping cost. Do not ship your equipment to the factory without prior authorization.

NON-WARRANTY REPAIR

In most instances, your dealer or local authorized Diamond Computer Systems agency can provide the necessary maintenance or service. For the name of your local authorized repair agency, contact your dealer or contact Diamond Computer Systems Technical Services at the above address.

Requirements that cannot be resolved by mail or by telephone will be referred to a local authorized repair agency, or to the factory. Do not ship your equipment to the factory without prior authorization.

Trackstar™ Reference Manual

© 1984 by Diamond Computer

Acknowledgements:

This manual was written by Randall Hyde and Mary Conard of Lazerware. This manual was prepared on Hayden's PIE Writer word processing system running on an Apple II and on the LISADRAW and LISAWRITE programs running on an Apple Computer Lisa. Apple™, Apple II™, Apple //e™, Apple //c™, Applesoft™, DOS 3.3™, and Lisa™ are registered trademarks of Apple Computer, Inc. IBM is a registered trademark of International Business Machines Corp. This manual is copyrighted by Diamond Computer. No parts may be reproduced without the express written permission of Diamond Computer.
©1984, Diamond Computer.
©1984, Lazerware.

Note:

The authors have attempted to ensure that the information contained herein is both factual and correct. However, neither Diamond Computer, Lazerware, nor the authors (Randall Hyde and Mary Conard) are responsible for the information in this text. The responsiblility for the use of this information is left to the reader.

Trackstar™ Technical Reference Manual

© 1984 by Diamond Computer

© 1984 by Diamond Computer

No part of this manual may be reproduced in any form without the express written consent of Diamond Computer.

TRACKSTAR™ Technical Reference Manual

Table of Contents

Chapter Zero: Installation

Installation..... 1

Chapter One: Introduction

What the TRACKSTAR™ board does for you IBM PC..... 21

Differences between the TRACKSTAR™ and an Apple][..... 21

IBM Keyboard vs. Apple][Keyboard..... 22

The TRACKSTAR™ Video Display 24

- Screen Memory..... 24
- The Text Modes 25
- The LORES Graphics Modes 25
- The HIRES Graphics Modes 26

The TRACKSTAR™ Speaker Port..... 26

The Apple Cassette Port 27

Chapter Two: Running Canned Programs

Running Canned Programs..... 29

Chapter Three: Standard I/O on the TRACKSTAR™ Board

Standard Output..... 33

The Stop-List Function..... 34

The Text Window..... 34

INVERSE, FLASHING, and NORMAL Text..... 35

Standard Input..... 36

RDKEY..... 36

GETLN..... 36

Escape Editing..... 37

Chapter Four: The TRACKSTAR™ Monitor Program

Introduction to the Apple Monitor 39
Examining Memory 39
Changing the Contents of Memory 39
Moving a Range of Memory 40
Comparing Two Ranges of Memory 40
Creating and Running Machine Language Programs 40
Examining and Changing Registers 41
Special Monitor Commands 41
Creating Your Own Monitor Commands 42
Special Monitor Memory Locations 42

Chapter 0:Installation

Introduction and Precautions

Installation Diagrams

Chapter 0: Installation

The TRACKSTAR system enables your IBM PC to run software written for an Apple II Plus computer. But before you can put the TRACKSTAR to work for you, it must be installed in your IBM PC computer system! By taking the time to carefully read these installation instructions, you can avoid considerable problems later on. So please read the entire installation guide before attempting to install your TRACKSTAR board.

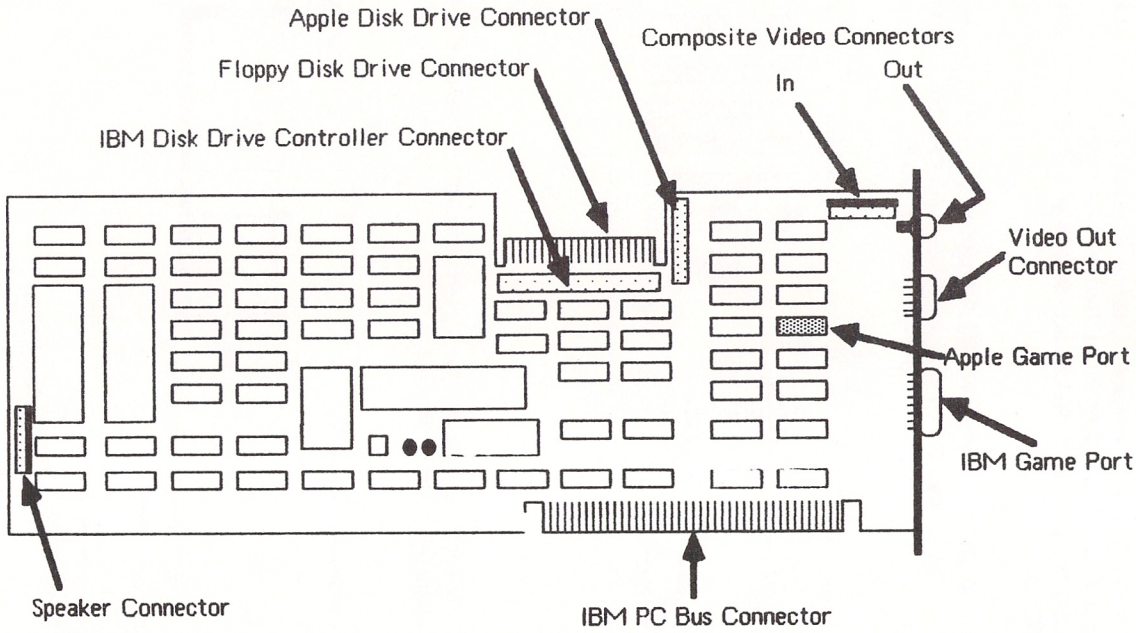
Before actually installing your TRACKSTAR system, you should inspect your package to make sure that you have everything that belongs with your TRACKSTAR system. In addition to this manual, you should also find a circuit board, three cables, and three floppy diskettes in your package. If these items are not present when you open the package, notify your dealer immediately. Most likely, everything that belongs with the TRACKSTAR package will be present so you can proceed with the installation of your TRACKSTAR hardware.

Before actually installing your TRACKSTAR board into your IBM PC computer, a few words of caution are in order. Several of the integrated circuits (ICs) on the TRACKSTAR board are susceptible to static damage. If at all possible, remove any wool clothing before installation and make sure you have been "grounded" by touching the power supply case on your IBM PC before touching the TRACKSTAR board.

The TRACKSTAR system sits electrically between the PC and the IBM PC's disk drives, speaker, and video display. Cables and connectors are provided that allow you to connect the TRACKSTAR system between the IBM disk drive controller and the IBM disk drives; that allow you to connect the TRACKSTAR system between the IBM PC motherboard and the speaker; and that allow you to connect the TRACKSTAR system between the color graphics adapter and the video display device. These connections allow the TRACKSTAR's on-board hardware devices to share physical devices (like the disk drives, speaker, and video display) yet allow the IBM PC to operate in the 8088 mode (while not running 6502 software) without having to recable or flip hardware switches. To install the TRACKSTAR board, the IBM PC must be opened, the TRACKSTAR card must be installed in one of the available slots, and the appropriate cables must be connected.

Before opening up your IBM PC it is very important that you disconnect all power from the IBM PC system. Installing your TRACKSTAR board with power applied to the IBM PC may damage the IBM PC or your TRACKSTAR board. To avoid electrical damage, always make sure that the power switch is off and that you have unplugged the power cord to the IBM PC before attempting to install your TRACKSTAR system.

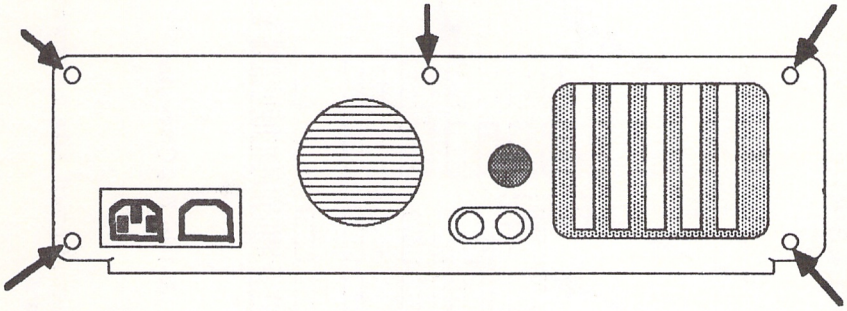
TRACKSTAR Board Layout



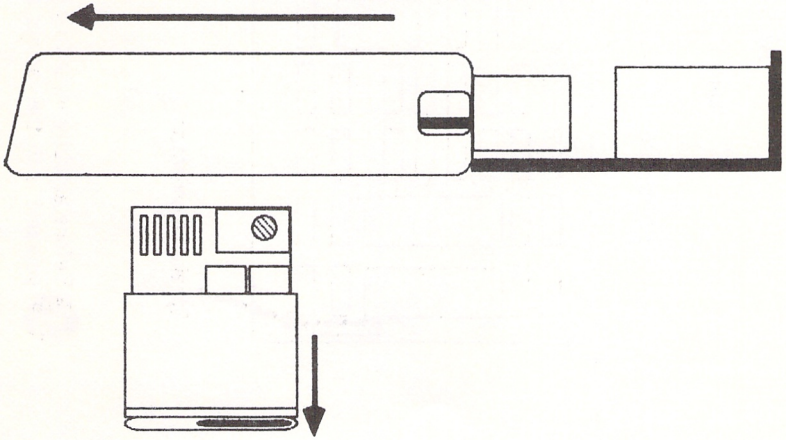
1 Before actually installing your TRACKSTAR™ circuit board, take a moment to familiarize yourself with the various connectors on the board.

2. Turn off the power and remove the line cord from your IBM PC. This step is *very* important. Failure to disconnect power may result in a damaged IBM PC or TRACKSTAR system. Also, unplug any peripheral devices attached to your IBM PC at this point.

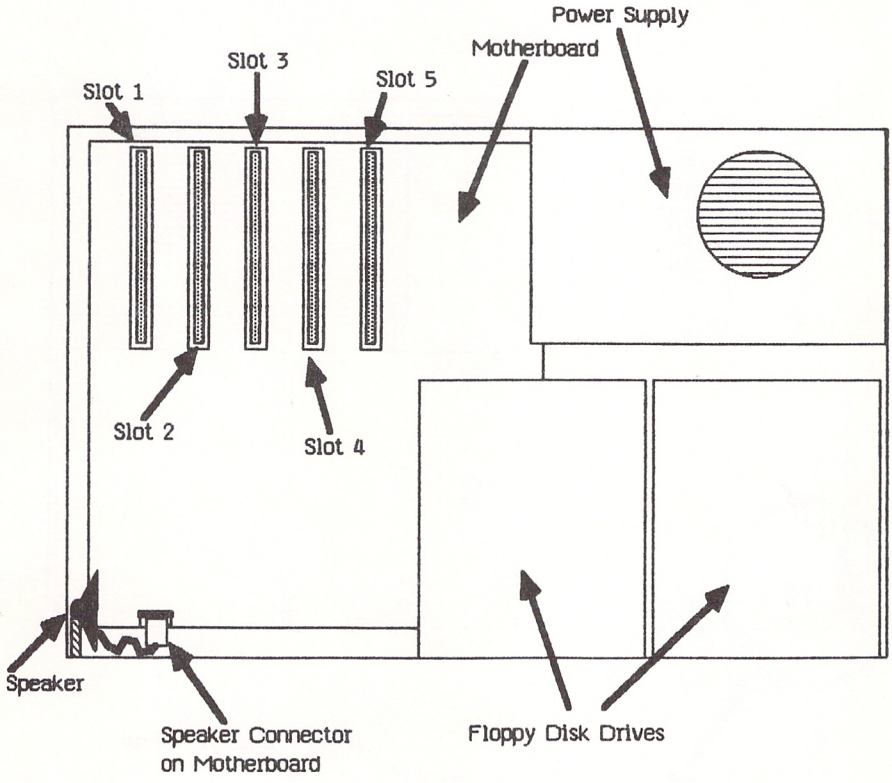
3. To install the TRACKSTAR circuit board into your IBM PC, you must first remove the cover from the IBM PC CPU box. This is easily accomplished by removing the five screws:



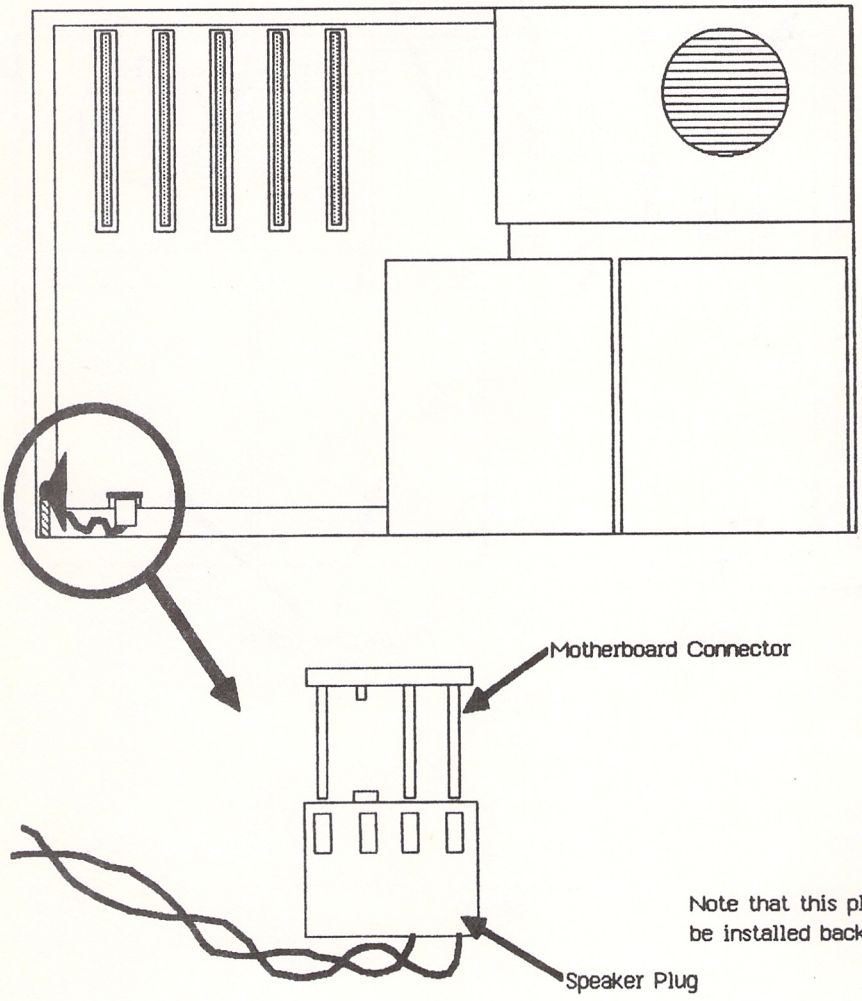
4. Carefully slide the case of your IBM PC towards the front of the computer and completely remove the case.



5 Now take a moment and familiarize yourself with the internal construction of your IBM Personal Computer.



6 The first step of actual installation is to remove the speaker plug from the speaker port connector on the IBM PC motherboard. Note that the plug is *polarized*. That is, there's only one way to attach the plug to the connector



Note that this plug cannot be installed backwards!

TRACKSTAR

User Manual

Copyright ® Leading Edge Stuff and Diamond Computer Systems, Inc

Congratulations

By selecting the Diamond TRACKSTAR™ board you have invested in a product that will bring many hours of productive use in business, education and entertainment. The people at Diamond Computer Systems, Inc. place great emphasis on quality in product design, workmanship and the support they provide to you.

With the selection of the TRACKSTAR board you will be able to harness the power of the three greatest libraries of software in the microcomputer industry. These include IBM's PC-DOS, Apple's DOS 3.3 and Digital Research's CP/M for the Apple. In addition you will be able to use software written for Apple Pascal and some progams written for Apple's new operating system ProDOS.

The TRACKSTAR board provides the convenience and power of several very unique features. These include video output that supports all Apple graphics mode and is compatable with IBM's RGB and Monochromatic Monitors as well as the standard Apple composite output. Support of 80-Column operation and Language Card features(for increased memory and use with special languages). A connector for an external Apple compatible drive to handle all the unique Apple copy-protection schemes. Support of both Apple and IBM game ports. Through software it provides support of file transfer between Apple DOS and PC-DOS, also between Apple CP/M and PC-DOS. Further, under software control you have the ability to define function keys as specific Apple commands such as Catalog, List, etc.

ABOUT THIS MANUAL

We have prepared this manual to introduce you to the features of the TRACKSTAR. It is intended as a starting point for you to learn about the Apple environment. It also provides the necessary information that you will need to use the card with Apple software.

Should you need more in-depth information about the TRACKSTAR card, we have also provided a Reference Manual which will answer any technical questions you may have.

Once again congratulations and Thank You!

TRACKSTAR and **Diamond Computer** are registered trademarks of Diamond Computer Systems, Inc.

Apple and **ProDOS** are registered trademarks of Apple Computer, Inc.

IBM and **PC** are registered trademarks of International Business Machines Corporation.

UCSD Pascal is a trademark of the Board of Regents of the University of California.

Microsoft and **MS-DOS** are registered trademarks of Microsoft Corporation.

Videoterm is a registered trademark of Videx Corporation.

Warranty

Diamond Computer Systems Inc. ~ 1-Year Limited Warranty

Diamond Computer Systems Inc. warrants this product to be free of defects in materials and workmanship for a period of 1 year from the date of purchase from an authorized Diamond dealer. This warranty is limited to the original purchaser, and to Diamond products that are sold and used within the United States. A copy of a valid dated saled receipt must be submitted with the product for warranty service. Software is specifically excluded from coverage under this warranty.

This limited warranty applies only to Diamond products which do not function properly under normal use, within the manufacturers specifications. It does not apply to products that, in the sole opinion of Diamond Computer Systems Inc. , have been damaged as a result of accident, misuse, neglect, improper packing, or shipping. This warranty is void if the Diamond label or logo, or the serial number have been removed or defaced, or the product has been modified or serviced by other that Diamond Computer System Inc. or an authorized Diamond Service Center.

During the 1- year warranty period, Diamond Computer System Inc. will repair or replace, at its option, any defective product with no charge for parts or labor. To obtain warranty service, write or phone:

Diamond Computer Systems, Inc.
Service Department
3380 Montgomery Drive
Santa Clara, CA. 95054
(408) 986-0100

The Service Department will determine whether service should be provided by an authorized service agency in your area or by Diamond Computer Systems, Inc. Do not ship equipment to Diamond Computer System, Inc. without a return authorization number. The purchaser must prepay shipping costs and insurance, and assume the risk of loss during shipping.

Software programs are provided "as is" without warranty of any kind, either expressed or implied. The entire risk as to the selection, quality, results, and performance of the programs are with the user, who must assume all liability and expenses incurred as a result thereof.

Except as expressly set forth above, no other warranties are expressed or implied, including, but not limited to, any implied warranties of merchantability and fitness for a particular purpose. Diamond Computer Systems, Inc. expressly disclaims all warranties not stated herein. All warranties, whether expressed or implied, are limited to the duration of this warrranty.

In the event that the product is not free from defects as warranted above, the purchaser's sole remedy is repair or replacement as provided above. Under no circumstances will Diamond Computer Systems, Inc. be liable to the purchaser or any user for any damage, including incidental or consequential damages, expenses, lost profits, lost savings, or other damages arising out of use of the inability to use this product.

TRACKSTAR USERS MANUAL

-- TABLE OF CONTENTS --

1. INSTALLATION ---------- install 1,20

2. OPERATION ---------- operation 1,17

3. SERVICE ---------- service 1,3

Installation

The following section is a step-by-step pictorial description of how to install the TRACKSTAR board into your IBM Computer. Please follow each step carefully and double-check your work. In order to perform this installation you will need to observe the following:

1) Turn-off ALL POWER!

2) You will need a blade screw driver to remove the IBM System Unit cover.

3) Have a clean work space for the installation.

4) Remove all diskettes from the area.

5) Follow each step of the Pictorial Installation -

CAREFULLY!!

In order to use the features of the TRACKSTAR you will need the following cards installed in your IBM:

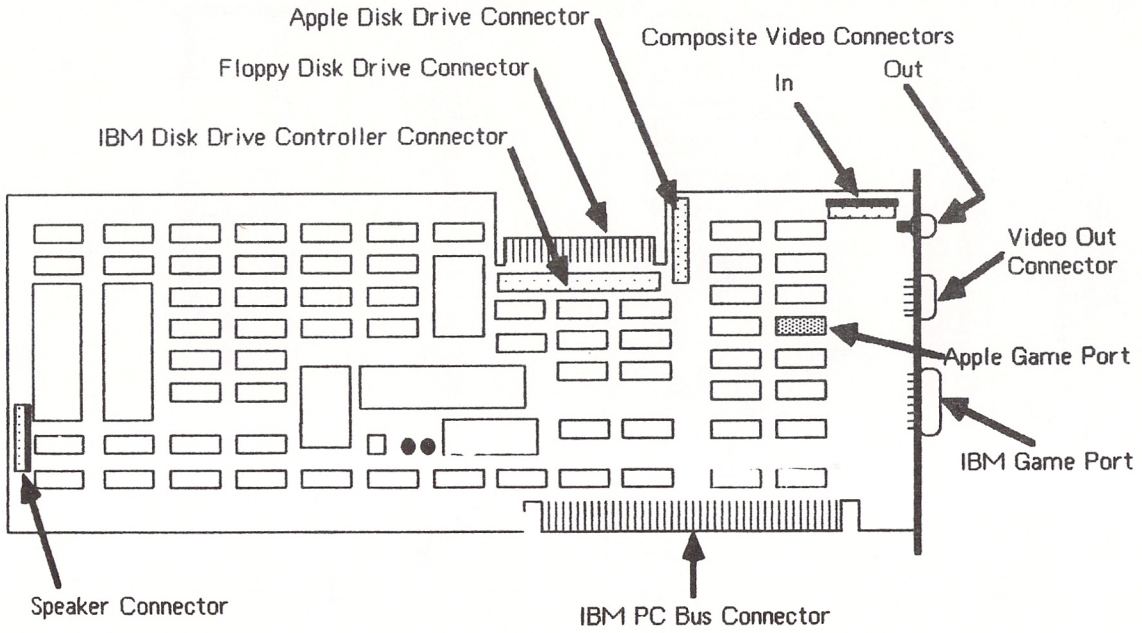
1) Monochromatic or Color Adapter Card.

2) Disk Controller Card.

3) Parallel Interface Card.
OR
4) Serial Interface Card.

With this information you should be ready to install your TRACKSTAR Card. In later chapters we will discuss how to install the software, initialize the TRACKSTAR, run Apple software, and use the many powerful commands of the TRACKSTAR menus.

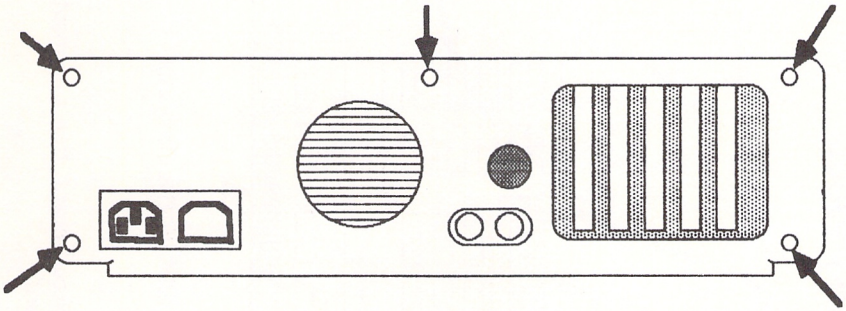
TRACKSTAR Board Layout



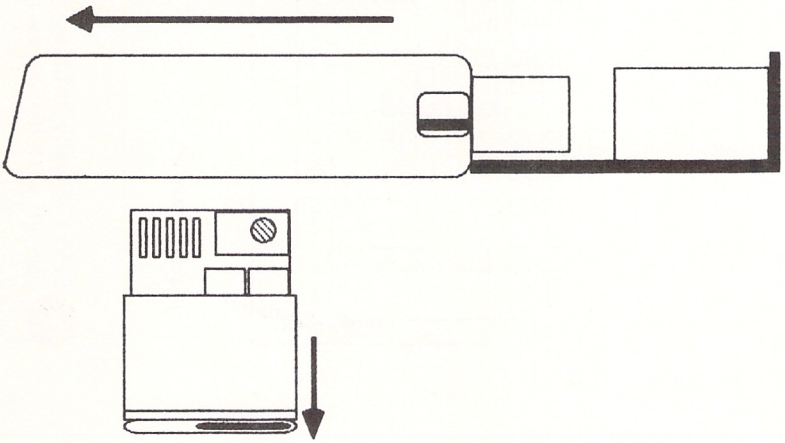
1 Before actually installing your TRACKSTAR™ circuit board, take a moment to familiarize yourself with the various connectors on the board.

2. Turn off the power and remove the line cord from your IBM PC. This step is *very* important. Failure to disconnect power may result in a damaged IBM PC or TRACKSTAR system. Also, unplug any peripheral devices attached to your IBM PC at this point.

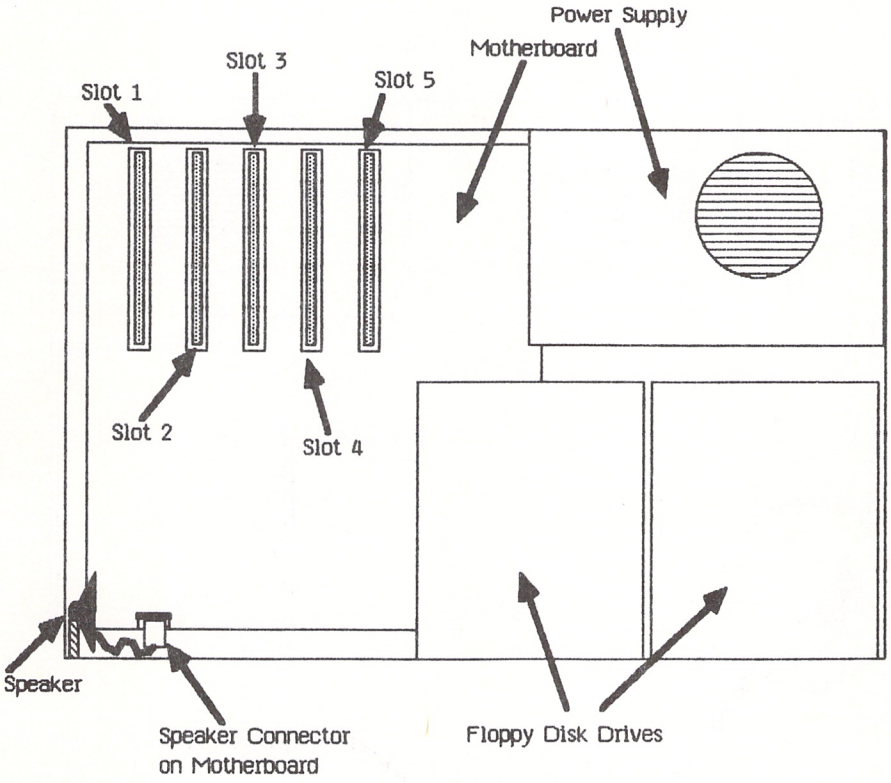
3. To install the TRACKSTAR circuit board into your IBM PC, you must first remove the cover from the IBM PC CPU box. This is easily accomplished by removing the five screws:



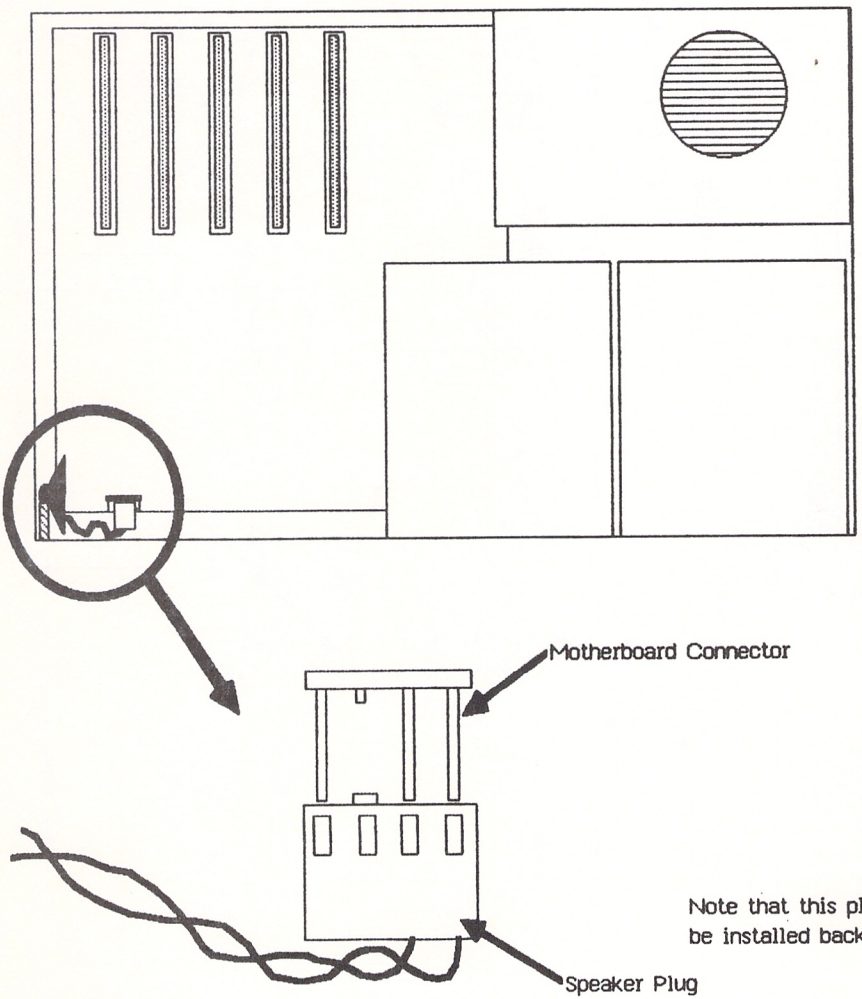
4. Carefully slide the case of your IBM PC towards the front of the computer and completely remove the case.



5 Now take a moment and familiarize yourself with the internal construction of your IBM Personal Computer.



6 The first step of actual installation is to remove the speaker plug from the speaker port connector on the IBM PC motherboard. Note that the plug is *polarized*. That is, there's only one way to attach the plug to the connector

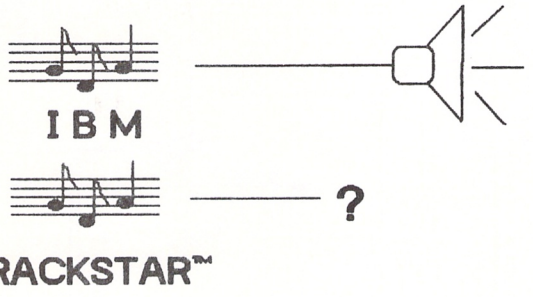


Note that this plug cannot be installed backwards!

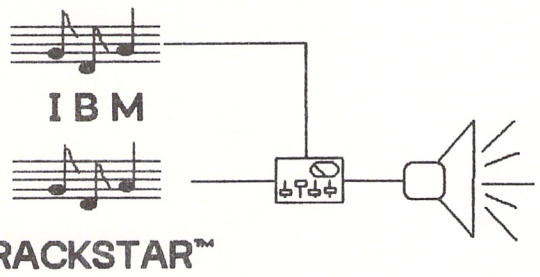
7

The TRACKSTAR™ board must be electrically inserted between the IBM PC motherboard speaker port and the speaker.
The speaker output of the IBM PC and the TRACKSTAR™ system are mixed on the TRACKSTAR™ board; this lets the TRACKSTAR™ share a single speaker with the IBM PC.

Before:

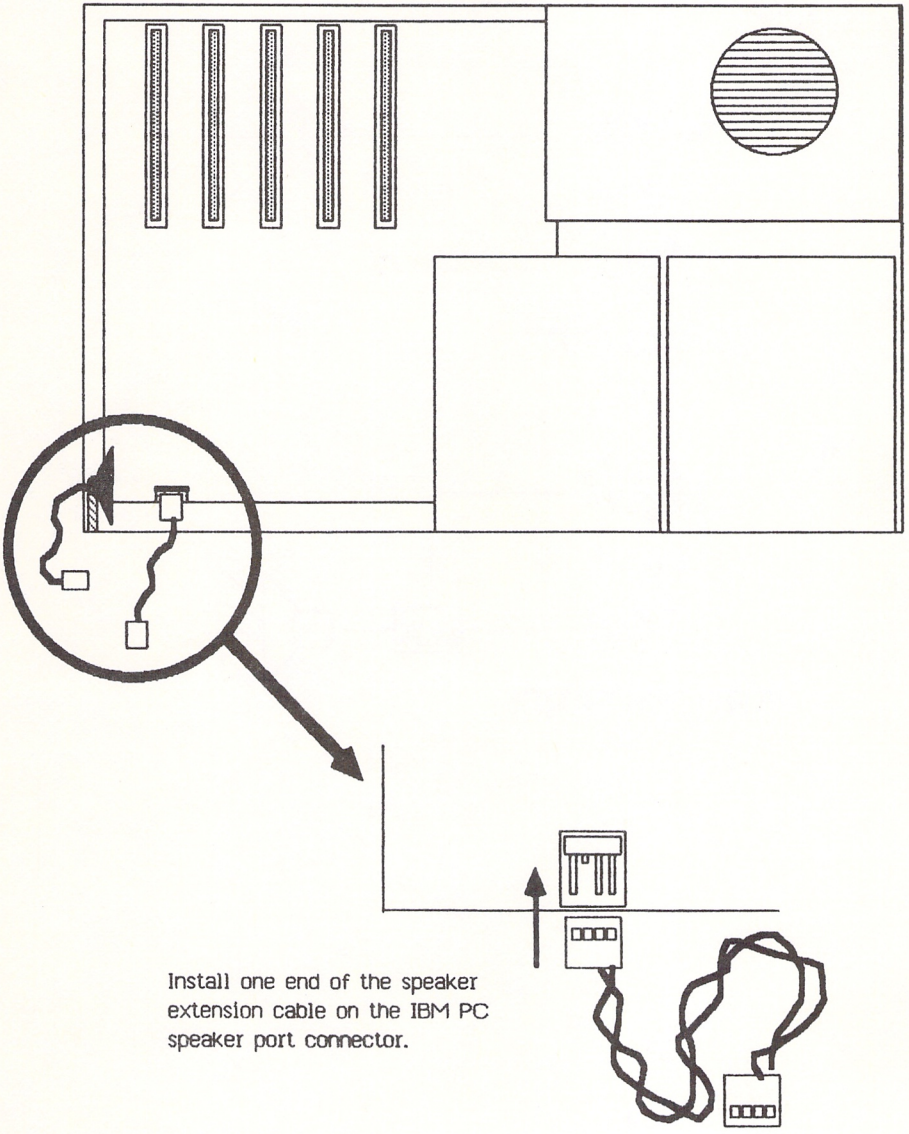


After:



8

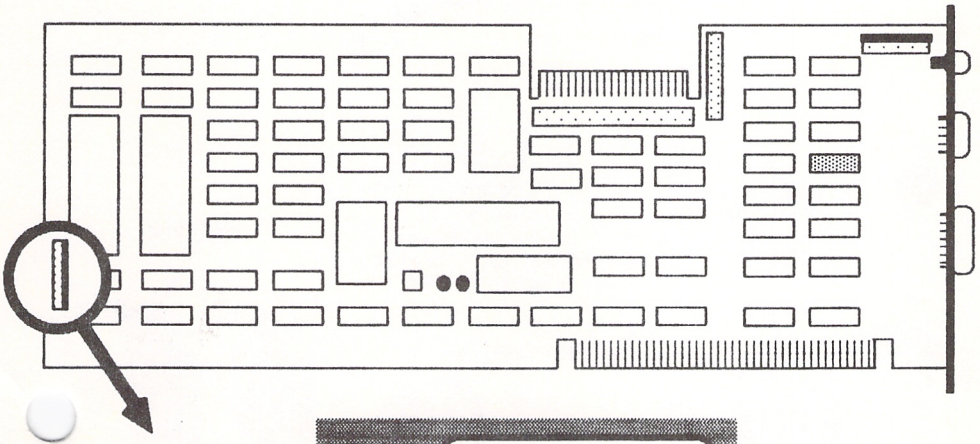
After unplugging the speaker plug from the speaker connector on the IBM PC motherboard, take the speaker port extension cable supplied with your TRACKSTAR™ board (the cable that contains a speaker port plug on each end) and plug one end of the extension cable onto the IBM PC's motherboard speaker port connector.



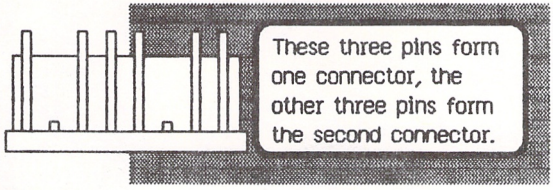
Install one end of the speaker extension cable on the IBM PC speaker port connector.

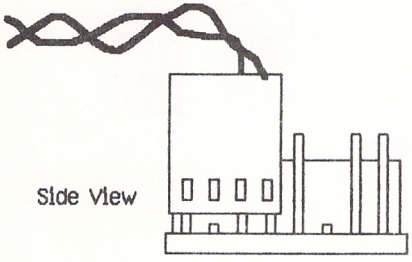
9

On the lower left hand side of the TRACKSTAR™ circuit board (when viewed from the top, component side), there is a small connector containing six pins. This connector is the speaker port connector for the TRACKSTAR™ system. Note that *two* speaker plugs can be attached to this port. Attach the free end of the speaker extension cable (whose other end was connected to the IBM PC motherboard in Step Eight) to three of these pins, connect the speaker plug to the other three.



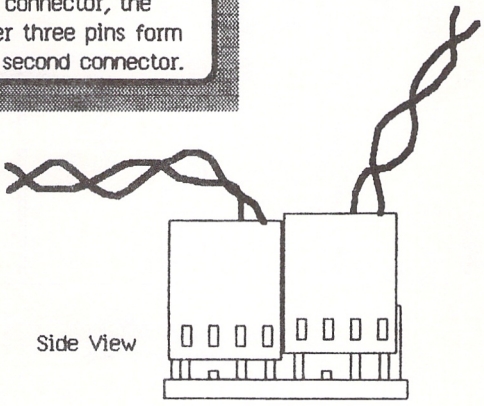
Side View





Side View

Attach IBM Speaker plug to TRACKSTAR™ speaker connector.

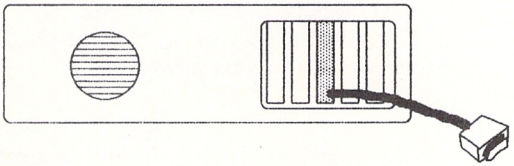


Side View

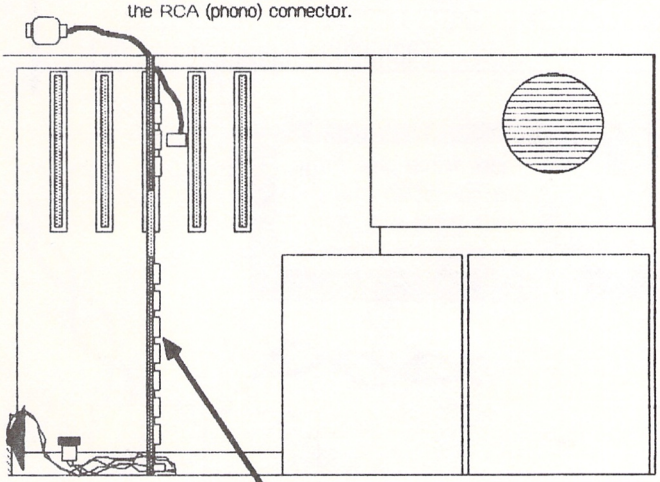
Then install the speaker extension cable to the other side of the TRACKSTAR™ speaker connector.

Note: both speaker ports on the TRACKSTAR™ board are common. You can attach either speaker plug to either side of the connector.

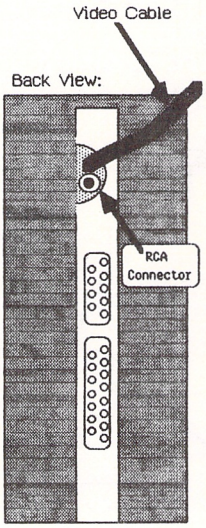
10 Get the video cable from the TRACKSTAR™ package. The video cable has a DB9 connector on one end and a molex connector on the other. Feed the molex connector through the opening for slot three in the back of the IBM PC.



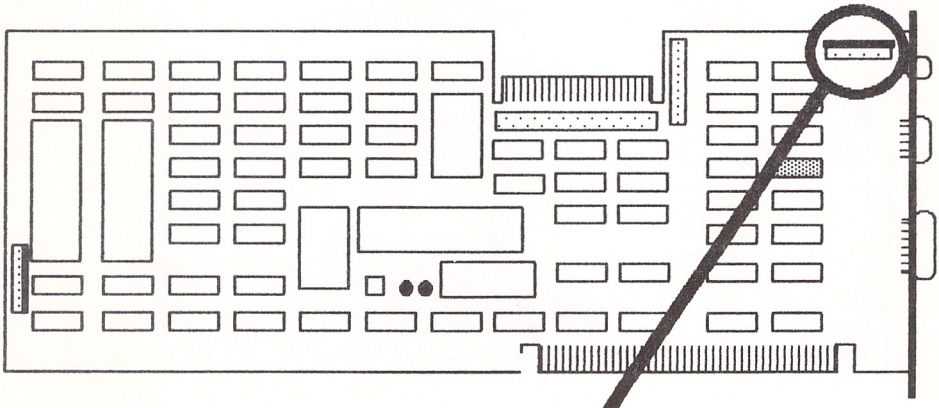
11 Now install the TRACKSTAR™ in slot 3 of your IBM PC. While there is no electrical reason why the TRACKSTAR™ must be installed in slot 3, the speaker and disk drive cables aren't of sufficient length to allow you to insert your TRACKSTAR™ board in an arbitrary slot. The video cable should fit through the hole cut in the back panel by the RCA (phono) connector.



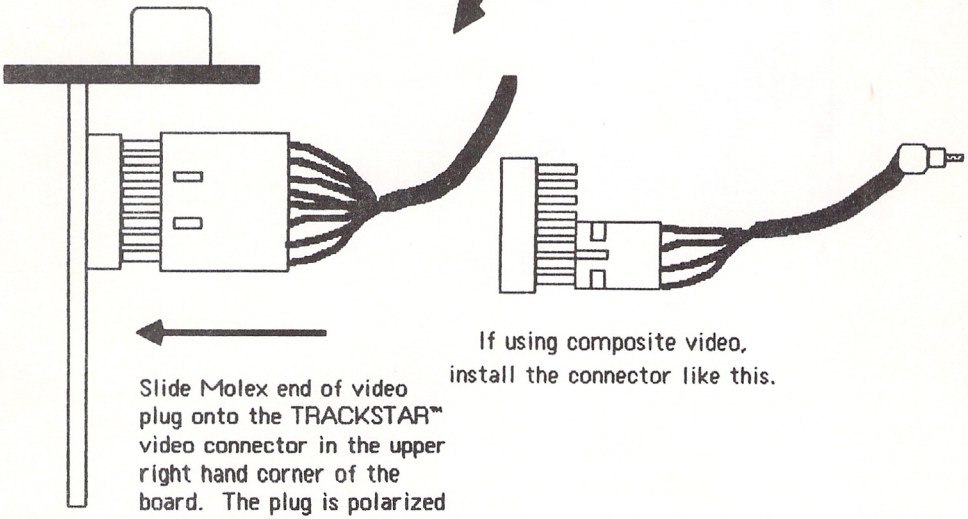
TRACKSTAR™ board installed in slot 3.



12 Next, connect the molex end of the video cable to the video connector on the TRACKSTAR board.



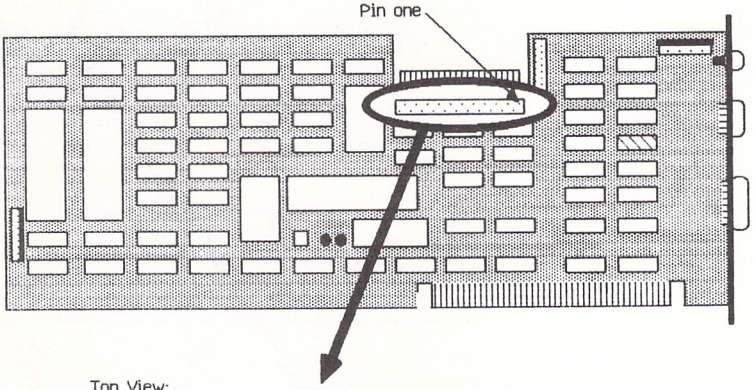
Top View:



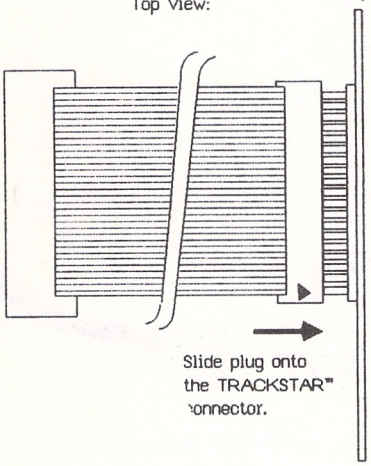
If using composite video, install the connector like this.

Slide Molex end of video plug onto the TRACKSTAR™ video connector in the upper right hand corner of the board. The plug is polarized so it is difficult to plug it in backwards.

13 Now it is time to connect the TRACKSTAR™ system between the IBM's floppy disk controller and the floppy disk drives. First, attach the disk controller connector cable supplied with the TRACKSTAR™ system to the connector diagrammed below. Pin one is marked with a small arrow on the cable's plug. Attach the plug to the connector on the circuit board with the arrow aligned with the pin in the upper right hand corner of the connector. The cable should point upwards at this point, however, trust the arrow even if the cable points downward. In either case, make sure that the cable is positioned up and away from the IBM motherboard.

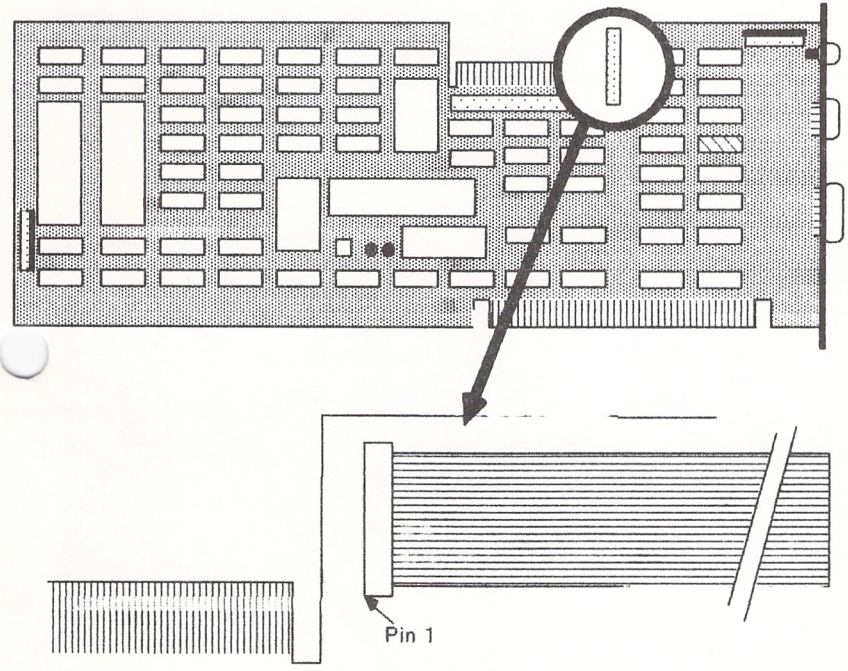


Top View:

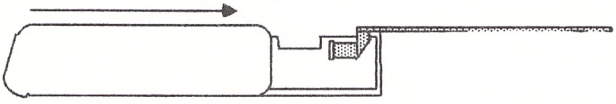


Slide plug onto the TRACKSTAR™ connector.

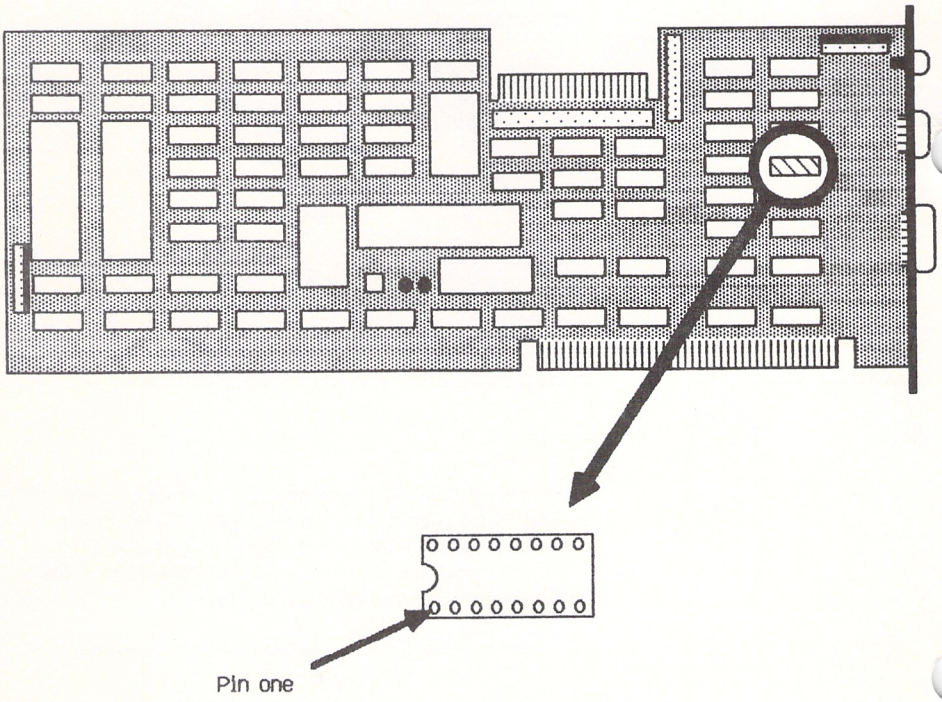
14 If you wish to install an Apple compatible floppy disk drive on your TRACKSTAR™ system, now is the best time to do so. This step is optional, if you do not have an Apple compatible floppy to install, continue on to step 15. The Apple compatible floppy disk drive cable should be plugged onto the 20-pin connector shown below. Th disk drive cable *must* always exit towards the rear of the IBM PC CPU box.



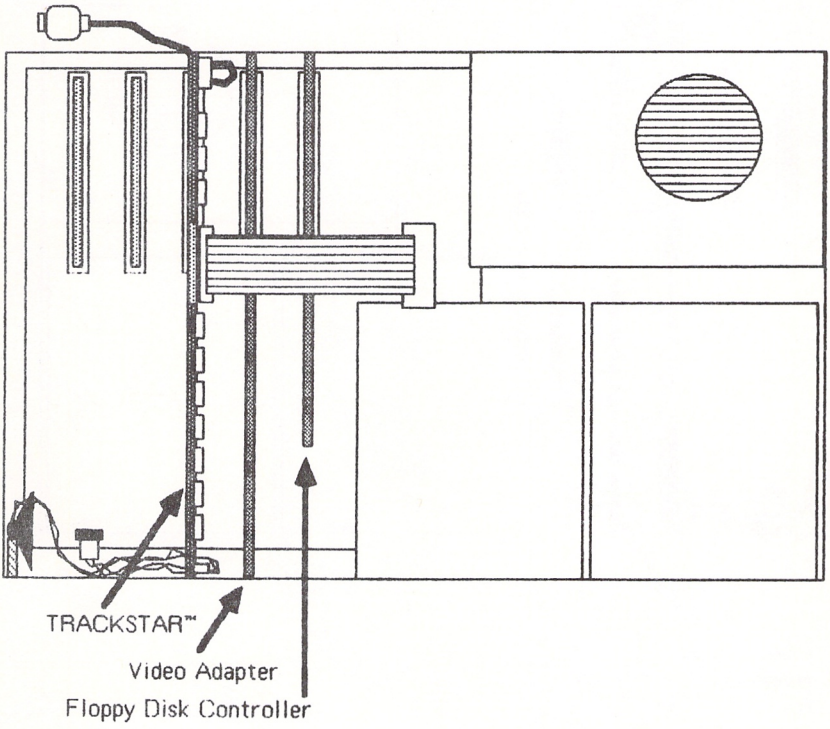
For best results, the Apple compatible floppy disk cable should be folded over the top of the TRACKSTAR™ board and it should exit the IBM via the hole for slot two. If all your IBM slots are filled, the disk cable should exit the IBM between the IBM's backplate and cover.



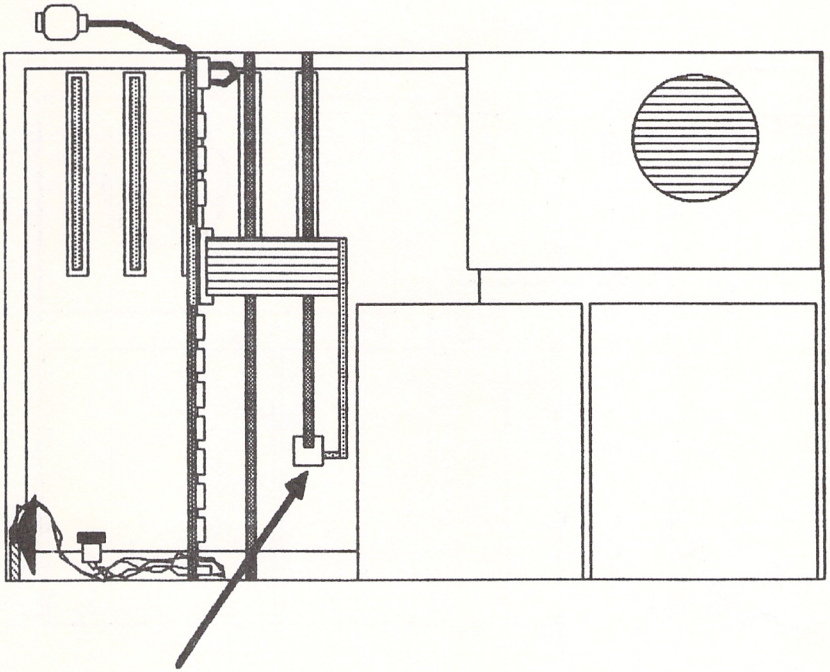
15 If you want to install an Apple compatible game controller, this is the most convenient time to do so. The Apple game port is a sixteen-pin dual in-line connector. It is the empty socket located four chips down on the right hand side of the TRACKSTAR™ board. Pin one is located at the lower left hand corner of the socket. If you aren't using slot two, the game controller cable can be fed into the IBM through the hole for slot two. The unused hole above the cassette and keyboard sockets is another likely location where you can feed the game controller cable into the IBM. Installing an Apple compatible game device is an optional step. If you do not wish to install an Apple game I/O device, skip to step 16. Note: if an Apple compatible game device is plugged into the game I/O socket, you cannot use IBM compatible devices on the DB15 connector on the back of the TRACKSTAR™ board.



6. Install the IBM video adapter in slot four and the floppy disk controller in slot five. Route the disk controller cable from the TRACKSTAR™ board over the top of the Color Graphics and disk controller cards.

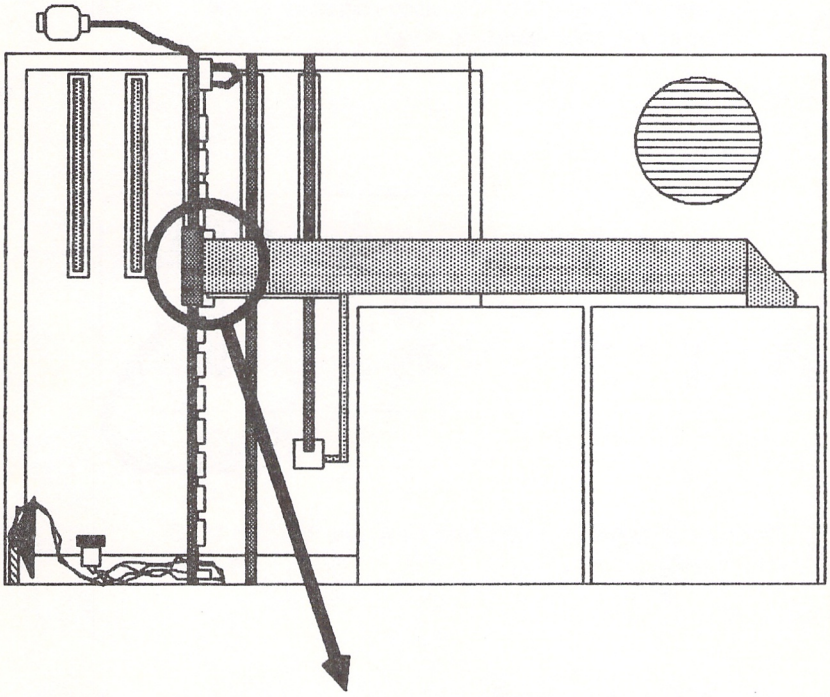


17 Route the cable around to the front of the disk controller card and plug it onto the edge-card connector on the disk controller. Note that the cable exits to the right of the disk controller card when viewed from the top.

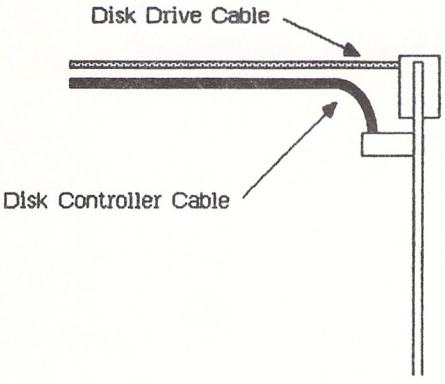


Plug TRACKSTAR™ disk connector onto the IBM floppy disk controller card.

18 Now attach the disk *drive* cable to the edge-card connector on the top of the TRACKSTAR™ board. The cable should exit to the right of the TRACKSTAR™ board when viewed from the top.

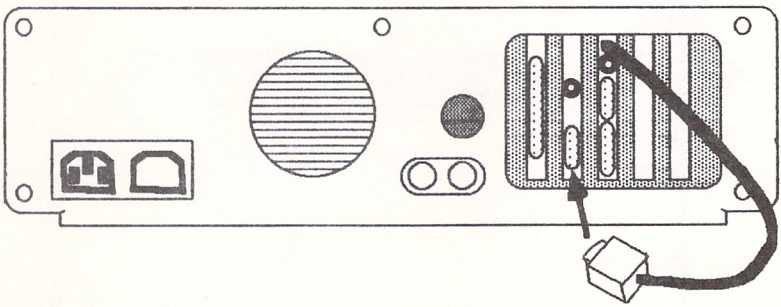


Back View:

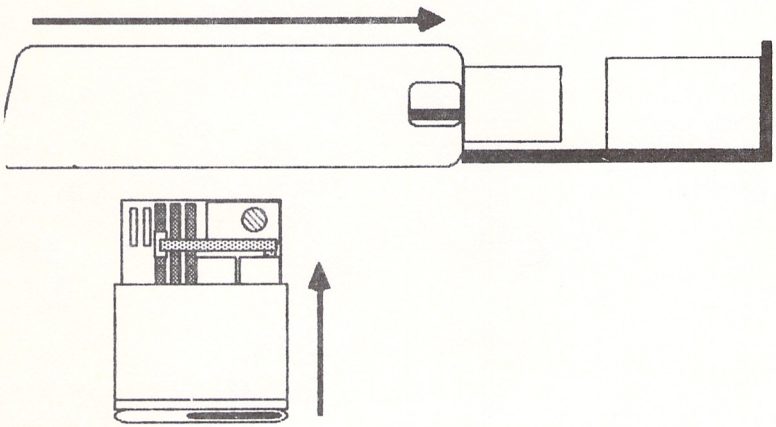


19 Install any remaining cards in your IBM PC's slots. You should use slot one before using slot two (try to leave slot two open for Apple compatible disk drives and game controller cables)

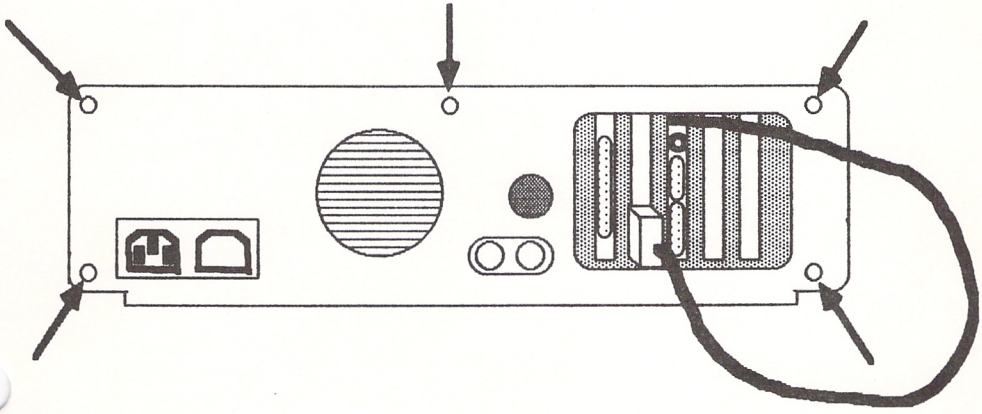
20 Attach the male DB9 end of the video connector to the female DB9 connector on the Color Graphics Adapter.



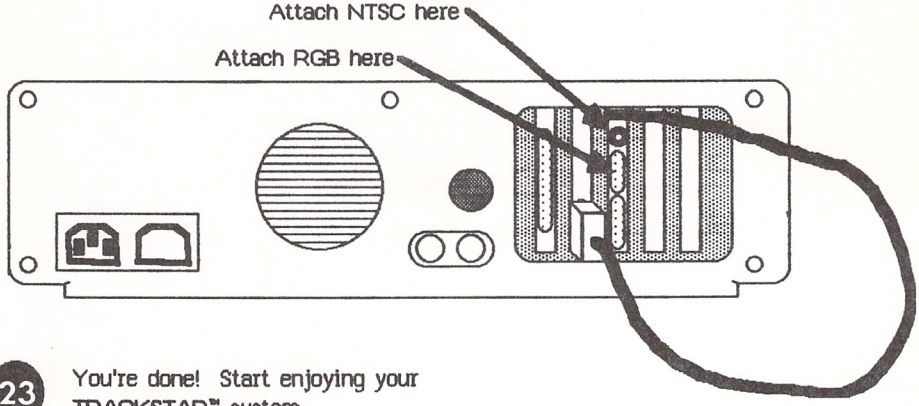
21 Carefully slide the computer case back over the IBM PC's chassis.



22 Tighten the screws on the back of the IBM PC chassis as shown below:



23 Attach your RGB display connector to the female DB9 connector on the TRACKSTAR™ board. If you are using a composite (NTSC) video monitor, connect it to the RCA (phono) jack on the TRACKSTAR™ card.



23 You're done! Start enjoying your TRACKSTAR™ system.

Operation

Start-Up

Overview - This section of the manual is designed to help you quickly begin to enjoy the power of the TRACKSTAR. In this discussion we have made the following assumptions:

1) You are familiar with the operation and use of your IBM computer.
2) That you have had little or no experience with the Apple][+.
3) You have followed the directions in the installation portion of this manual and the TRACKSTAR card is properly installed in your computer.
4) That you have either a parallel or serial printer interface to a printer in your IBM computer.

We will cover the steps needed to: a) Copy the software to run the TRACKSTAR. b) Running the TRACKSTAR. c) Explain differences between using the TRACKSTAR and the Apple][+. And d) Exiting the TRACKSTAR.

Installing Software for TRACKSTAR

Two diskettes are provided for TRACKSTAR. The first disk is the TRACKSTAR Utility Diskette. This disk is designed to run on your IBM and contain two files called **TRKSTR.EXE** and **TRKSTR.SYS** . **TRKSTR.EXE** executes **TRKSTR.SYS** which contains the actual software needed to intialize the TRACKSTAR Card (including the monitor routines, Applesoft Basic, Z-80 routines and 80 column routines). It also contains the software for all of the menu driven commands available with TRACKSTAR and the special funtion command feature of TRACKSTAR.

The second diskette is provided by S & H Software. This is a high performance version of Apple's DOS. There is a separate manual to explain the features of S&H Software and DOS is explained in detail by the TRACKSTAR Reference Manual.

In order to run the TRACKSTAR card it is necessary to run the two IBM files TRKSTR.EXE and TRKSTR.SYS. We recommend that the files be transfered to any of your boot diskettes where you think you may have a need for Apple software. The proceedure to use in copying files from one disk to another under PC-DOS is explained in the IBM DOS manual but for convenience it is repeated here:

Step 1 - Insert a working copy diskette of PC-DOS in Drive A and turn on your computer. Once the DATE and TIME prompts appear and you respond, the fimilar **A>** will appear. Now you are ready to copy the TRACKSTAR files to your PC-DOS diskette.

STep 2 - Insert your TRACKSTAR Utilities Diskette in Drive B.

Step 3 - At the **A>** type **COPY B: TRKSTR.* = A:**

*Note: the COPY utility must be on the diskette in Drive A or the BAD COMMAND error will appear. The * is used as a wild card function, as explained in your DOS manual.*

Step 4 - Both internal drives will alternately whir as the computer copies TRKSTR.EXE and TRKSTR.SYS from the TRACKSTAR Utility Diskette to the diskette in drive A:

Step 5 - You may repeat the process for any additional diskettes where you feel you will need to run TRACKSTAR (We recommend all your boot disks have it)

Now that the software is loaded lets use the TRACKSTAR !

Initializing the TRACKSTAR

To use the TRACKSTAR you will first have to run the TRACKSTAR utility programs. If you are doing this after copying the files to you PC-DOS diskette, you should still have the **A>** on the screen. If you do, type **TRKSTR**. The TRACKSTAR program will be loaded and the first TRACKSTAR menu will appear (We will discuss the other menu options in a later sections. Right now lets get the card running).

Remember: To prepare the TRACKSTAR for operation, follow these steps:

1) Insert the PC-DOS diskette with both TRKSTR files on it.
2) Turn on your system
3) When the **A>** appears type **TRKSTR**

The disk in drive A will spin and in a few seconds the following menu will appear on your monitor:

TRACKSTAR Utility Program
(c) Diamond Computer Systems, Inc. 1984

- INITIALIZATION MENU -

[A] Initalize TRACKSTAR
[F] Display/Modify TRACKSTAR Function Keys
[S] Select TRACKSTAR Control Menu
[I] Return to IBM mode

Please Select Item!

*** Apple is the trademark of Apple Computer, Inc.**
*** CP/M is the trademark of Digital Research, Inc.**
*** IBM is the trademark of International Business Machine**
*** TRACKSTAR is the trademark of Diamond Computer Systems, Inc.**

Initialization Menu - As indicated this menu will appear whenever the TRACKSTAR system is booted on your IBM machine. A function is selected from this menu by simple typing the key indicated.

You type A

The Initialize TRACKSTAR function will be selected. This function is the first thing you must do to use the TRACKSTAR. It will cause the TRKSTR.SYS file to be loaded into the TRACKSTAR board. That will make the TRACKSTAR fully operational. When you select A the following menu will appear.

Do you require 80 Column mode (default 40 Columns) ?

This question appears because many programs for Apple were never designed for 80 column operation and will not function properly if the 80 column software routines are loaded into the TRACKSTAR. If the program you intend to run does use 80 column screens (Pascal or CP/M for an example) then type **Y**. If you do not require 80 column display then simply press any other key. (don't worry you can change this at any time).

Once you have made you choice the following statement will appear on the screen:

PLEASE WAIT !! Initializing TRACKSTAR in 80 Column mode.

This display will indicate which mode you have selected (it would say 40 Column if you had hit any other key but **Y** in response to the previous question).

After about 15 seconds of disk activity the following screen will appear :

***Pls remember the [HELP key] is [F1] then [ESC]**
- when the TRACKSTAR is in operation -

*** Insert Diskette you wish to run in Drive A**

*** If Apple-type disk drive attached**
Insert disk in Apple drive, not IBM drive.

Press Return key when ready!

This screen is telling you of many powerful features of the TRACKSTAR. (This screen appears at this point to remind you of these features).

Remember: a) When ever you are running your Apple programs and wish to leave TRACKSTAR operation, you simply press function key **F1** and then the key marked **ESC**. The TRACKSTAR's control menu will replace whatever was on the screen.

This Control Menu will not stop the program that is currently running on the TRACKSTAR, but it will give many options about what you can do. We will discuss these options in another section, but right now it is important to remember that F1 followed by **ESC** will get you to this menu. (**Menu.8**).

b) At this point the IBM internal drives will act like Apple drives. They will read and write Apple diskettes. Drive A is the equivalent of Drive 1 in Slot 6 of an Apple][+.

c) As we discussed earlier an unique feature of the TRACKSTAR is that an external Apple drive can be connected to the card. This makes the system fully Apple copy-protection compatible since it can handle all of the unique copy protection schemes used on the Apple. Some of which cannot be handled by the IBM drives. When this drive is installed, the TRACKSTAR knows this and will run Apple software from this external drive instead of Drive A. Drive B will still act as the second Apple drive in either case.

When the Apple program you want to run has been inserted in the appropriate drive, press the **RETURN** key. You will hear the familiar "ratchet noise" of the drive booting and you will be running in the Apple environment.

At this point your IBM will act like an Apple][+. You can use it to run programs that you could normally run on an Apple][+. Obviously an Apple][+ will work differently than an IBM PC, but fortunately, the programs you will use will explain how to run them on an Apple. Before you run your Apple programs lets talk about the differences between the TRACKSTAR on the IBM and an Apple][+.

The Keyboard

The IBM keyboard offers many keys not available on the Apple][+, but in reality there are only a few things to watch for:

1) Many programs were not designed to support lower case letters (because the Apple][+ does not support lower case letters), if you are running one of these programs make sure the **Caps Lock** key is pressed.

2) Some programs use the "shift wire" modification to create lower case. The TRACKSTAR cannot support this function.

3) Some of the keys on the IBM keyboard are simply not supported on the Apple][+ keyboard so they may cause unusual results. (The IBM **Alt** key for example)

4) The IBM cursor keys are not supported on Apple programs.

Disk Drive

As we discussed, the TRACKSTAR offers the special feature of supporting an external Apple type drive to ensure full Apple copy protection compatibility. When this drive is installed your Apple programs will boot from this drive. If it is not installed then they will boot from Drive A of the IBM.

Slots

The Apple (much like the IBM) provides special features through cards that plug into slots inside the computer. In the Apple environment, access to these features is obtained by calling a specific slot # (through the command PR# or IN#). Don't worry, Apple software will do this automatically. It is important to know which slots are supported when you read a software user manual to ensure you can run the particular program.

Although the TRACKSTAR does not have the physical slots, like an Apple, it does support several of the slot functions. Following are the slots supported, their features, and their limitations:

SLOT	Comments
Slot 0	Supports Language Card commands and functions. The language card function of the TRACKSTAR provides an additional 16K of memory. This is required for Pascal programming, and some programs that use the extra memory for storage (example is Apple Writer II).
Slot 1	Parallel Interface. Through tradition, Slot 1 is used for the parallel interface to the printer. TRACKSTAR emulates a standard parallel interface card to Apple programs but with an important twist. The data it receives for printing is passed along to the IBM parallel interface which is connected to the printer

If your IBM does not have a parallel interface then this slot is not supported. Also, and this is important, even though every command you send to the printer is supported, just as it would on the Apple][+, it does not support special graphics commands, such as you find on a GRAPPLER™ Card, nor does its hardware emulate printer cards. In other words, programs that expect certain hardware to be present will not function with the system (this only seems to be a problem with some graphics programs).

Slot 2

Serial Interface. Through tradition this is used for communications and for serial interface printers. Just as with Slot 1, the TRACKSTAR emulates this function through software and passes the information to the IBM for execution. In the case of serial interface the IBM will need to be intialized first. Please read the IBM manual for the serial interface card to understand how to do this.

The limitation of the serial interface is in the area of communications. Most communication programs are written for specific hardware (to improve performance). Since the TRACKSTAR does not provide that hardware it may not work with a particular communication program.

Slot 3

Supports 80 Column operation and supports all Videx Videoterm™ Commands for 80 Column mode. To engage 80 Column mode simply type **PR#3** at the Apple prompt (]). All further operations will be in 80 column mode until you type **PR#0**. Remember, don't worry the Apple software (such as Pascal and CP/M) will handle this automatically.

However just as with the other slots, this is created in software and not hardware. Some programs that use the 80 column mode do so through hardware and therefore cannot be used with the 80 Column feature. Fortunately, Pascal programs and CP/M programs do not have this limitation.

Slot 4 — Not supported (normally used for a CP/M card which the TRACKSTAR supports anyway).

Slot 5 — Not Supported (normally used for a second pair of disk drives).

Slot 6 — Supports standard Apple disk controller commands. *Remember: If an Apple compatable drive is connected to the TRACKSTAR it will act as the drive in Slot 6, Drive 1. If an Apple compatable drive is not provided then Drive A of the IBM will act as Drive 1. In either case Drive B will act as Drive 2.*

Slot 7 — Not Supported (normally used for video functions which the TRACKSTAR already supports this through its RGB, Composite and B/W output).

Cntrl-RESET

In the Apple environment this combination is the "last resort" in case of problems with a program before turning off the system. Since a RESET key does not exist on the IBM keyboard, you would use the following proceedure to perform the same function.

1) Press the **F1** and the **ESC** keys

2) Press the **R** key from the TRACKSTAR Control Menu.

This combination will create the same result as pressing CNTRL-RESET on the Apple][+. (CNTRL-RESET on the Apple][+ is like CNTRL-ALT-DEL on the IBM).

Other Options

From the same Control Menu you can select: **B** to reboot or boot a new program. **X** to return to TRACKSTAR without a CNTRL-RESET. **I** to go to the IBM operation mode.

OPERATION

- INDEX -

1) START-UP ---------- 1

2) INSTALLING SOFTWARE FOR TRACKSTAR ---------- 1

3) RUNNING THE TRACKSTAR ---------- 2

4) CONTROL MENU ---------- 5

5) EXIT TO MS DOS ---------- 7

6) PROGRAMMING THE FUNCTION KEYS ---------- 8

7) THE FILE TRANSFER UTILITY ---------- 10

8) THE DIFFERENCES BETWEEN TRACKSTAR AND AN APPLE II+ ---------- 13

1. START-UP

Overview - This section of the manual is designed to help you quickly begin to enjoy the power of the TRACKSTAR. In this discussion we have made the following assumptions:

1) You are familiar with the operation and use of your IBM computer.
2) That you have had little or no experience with the Apple II+.
3) You have followed the directions in the installation portion of this manual and the TRACKSTAR card is properly installed in your computer.
4) That you have either a parallel or serial printer interface to a printer in your IBM computer.

2. INSTALLING SOFTWARE FOR TRACKSTAR

Two diskettes are provided for TRACKSTAR. The one diskette is the **TRACKSTAR Utility diskette.** It is MS DOS formatted and has two files named **TRAKSTAR.EXE** and **TRAKSTAR.SYS.** TRAKSTAR.EXE is a program file and TRAKSTAR.SYS is a data file which will be used by the TRAKSTAR.EXE during initialization.

The other diskette is the **File Transfer Utility Diskette** which is Apple formatted and will run in TRACKSTAR mode and not in MS DOS. It enables you to transfer files between **APPLE DOS and MS DOS : APPLE CP/M and MS DOS.** There is a separate section of the manual to explain the features of File Transfer Utility in detail.

We recommend that the TRACKSTAR Utility files be transferred to any of your boot diskettes where you think you may have a need for Apple software. The procedure to use in copying files from one disk to another under MS DOS is explained in the MS DOS manual but for convenience it is repeated here:

Step 1 - Insert a working copy diskette of MS DOS in Drive A and turn on your computer. Once the DATE and TIME prompts appear and you respond, the familiar A> will appear. Now you are ready to copy the TRACKSTAR files to your MS-DOS diskette.

Step 2 - Insert your TRACKSTAR Utility Diskette in Drive B.

Step 3 - At the A> type **COPY B: TRAKSTAR.* = A:**

Note: the COPY utility must be on the diskette in Drive A or the BAD COMMAND error will appear. The * is used as a wild card function, as explained in your DOS manual.

Step 4 - Both internal drives will alternately spin as the computer copies TRAKSTAR.EXE and TRAKSTAR.SYS from the TRACKSTAR Utility diskette to the diskette in drive A.

Step 5 - You may repeat the process for any additional diskettes where you feel you will need to run TRACKSTAR (we recommend that all your boot disks have it).

Now that the software is ready. Lets use the TRACKSTAR!

3. RUNNING THE TRACKSTAR

To run the TRACKSTAR follow these simple steps:

1) Insert the MS DOS diskette with both TRAKSTAR files on it.
2) Turn on your system.
3) When the A> appears type **TRAKSTAR** then the <Return> key.

The disk in drive A will spin and in a few seconds the following menu will appear on your monitor:

TRACKSTAR Utility Program
(c) Diamond Computer Systems, Inc. 1985

- INITIALIZATION MENU -

[A] Normal Initialization

[B] High Compatibility Initialization

For option [B],
Apple DOS 3.3 Master diskette is required.

Please select item!

***Apple is the trademark of Apple Computer, Inc.**
***Trackstar is the trademark of Diamond Computer Systems, Inc.**

Initialization Menu - As indicated, this menu will appear whenever the TRACKSTAR system is booted on your IBM machine. A function is selected from this menu by simply typing the key indicated and <Return>.

3-1. Selecting Option [A]

This will cause the FPBASIC and I/O drivers in the TRACKSTAR Utility diskette to be loaded into TRACKSTAR memory. The following message will appear.

```
------------------------------------------------------------
Do you require 80 column screen (Y/N)?
------------------------------------------------------------
```

This question appears because many programs for Apple were never designed for 80 column operation and will not function properly if the 80 column software routines are loaded into the TRACKSTAR. If the program you intend to run does use 80 column screens(Pascal or CP/M for an example) then type Y. If you do not require 80 column display then simply press any other key(don't worry you can change this at any time).

Once you have made your choice the following statement will appear on the screen for just a moment.

```
------------------------------------------------------------
Please Wait!
------------------------------------------------------------
```

Then, the following screen will appear:

```
------------------------------------------------------------
*Please remember the [HELP key] is [F1] then [ESC]
  -when the TRACKSTAR is in operation-

*Insert Diskette you wish to run in Drive A

*If Apple-type disk drive attached
 Insert disk in Apple drive, not IBM drive.

Press Return key when ready!
------------------------------------------------------------
```

This screen is telling you of the many powerful features of the TRACKSTAR(this screen appears at this point to remind you of these features).

Remember:

1) Whenever you are running your Apple programs and wish to leave TRACKSTAR operation, you simply press function key F1 and then the key marked ESC. The TRACKSTAR's control menu will replace whatever was on the screen.

This Control Menu will not stop the program that is currently running on the TRACKSTAR, but it will give many options for what you can do. We will discuss these options in another section, but for right now it is important to remember that F1 followed by ESC will get you to this menu.

2) At this point the IBM internal drives will act like Apple drives. They will read and write Apple diskettes. Drive A is the equivalent of Drive 1 in Slot 6 of an Apple II+.

3) As we discussed earlier a unique feature of the TRACKSTAR is that an external Apple drive can be connected to the card. This makes the system fully Apple copy-protection compatible since it can handle all of the unique copy protection schemes used on the Apple, some of which cannot be handled by the IBM drives. When this drive is installed, the TRACKSTAR knows this and will run Apple software from this external drive instead of Drive A. Drive B will still act as the second Apple drive in either case.

When the Apple program you want to run has been inserted in the appropriate drive, press the RETURN key. You will hear the familiar "ratchet noise" of the drive booting and you will be running in the Apple environment.

At this point your IBM will act like an Apple II+. You can use it to run programs that you could normally run on an Apple II+. Obviously an Apple II+ will work differently than an IBM PC, but fortunately, the programs you will use will explain how to run them on an Apple.

3-2. Selecting Option [B]

This will make the TRACKSTAR more compatible with the Apple machine. We recommend that you use this option if you have an Apple DOS 3.3 Master diskette. If a certain Apple program does not work with Option [A] it may work with Option [B].

Selecting Option [B] will load the FPBASIC file in the Apple Master diskette into the TRACKSTAR. There are two different versions of the Apple Master diskette. One is the old version which has FPBASIC with Monitor Routines(Autostart ROM) and the newer version which does not have the Monitor Routines. Using the older version of the Apple Master diskette will insure higher compatibility with the TRACKSTAR than the newer version. But, the newer version is still better than selecting Option [A]. When you use the old version of the Apple Master diskette you will see the "APPLE" sign-on message instead of the "Language Arts" sign-on message when booting diskettes.

To select Option [B] type B then <RETURN> when Initialization Menu appears. Then, the following message will appear:

```
-----------------------------------------------------------------
Insert Apple DOS 3.3 Master diskette in Drive 1(D1).
Press <RETURN-Key> when ready or <ESC-Key> to abort.
-----------------------------------------------------------------
```

At this point, insert the Apple DOS 3.3 Master diskette in PC Drive A or external Apple Drive if it is attached. Press <RETURN> Key. Then, the following message will appear:

```
-----------------------------------------------------------------
Do you require 80 column screen (Y/N)?
-----------------------------------------------------------------
```

If you are going to run an 80 column program, type Y, and for all others type N. The Drive will spin for a few seconds and the following Menu will appear:

```
-----------------------------------------------------------------
*Please remember the [HELP key] is [F1] then [ESC]
  -when the TRACKSTAR is in operation-

*Insert Diskette you wish to run in Drive A

*If Apple-type disk drive attached
 Insert disk in Apple drive, not IBM drive.

Press Return key when ready!
-----------------------------------------------------------------
```

At this point you are ready to run Apple programs.

Remember: Whenever you are running your Apple programs and wish to leave TRACKSTAR operation or other control actions of TRACKSTAR, you simply press function key F1 and then the <ESC> key.

4. CONTROL MENU

This Control Menu will be recognized when F1 function key then <ESC-Key> is pressed while an Apple program is running.

```
------------------------------------------------------------------
    - TRACKSTAR CONTROL MENU -

[R]  Reset Trackstar

[B]  Boot the Disk(PR#6)

[F]  Display/Modify Function Keys

[X]  Exit to MS DOS

[Return-key]  Return to Trackstar

------------------------------------------------------------------
```

Remember, while this menu is displayed your Apple program is still running!

The Control Menu provides the following options.

[R] : Reset the TRACKSTAR.

Equivalent to CNTRL-RESET on an Apple //e. Use it if the program indicates or if the program is frozen.

[B] : Boot the Disk.

Equivalent to PR#6 or <Control open-apple reset> on apple //e. Use it if you want to: a) Boot a new program in the Apple mode, or b) if the current program is frozen and not responding to [R].

[F] : Display/Modify Function Keys.

This will be explained in a later section. It allows you to define the IBM function keys to act like specific commands such as DOS commands or particular program commands(F2 = Catalog, F3 = Cntrl-P(Print in AppleWriter II), etc.).

[X] : Exit to MS DOS.

Refer to Exit to MS DOS section of the manual.

[Return] : Return to TRACKSTAR.

You have changed your mind or made changes to the function keys and wish to return to your Apple program.

5. EXIT TO MS DOS

There are two different ways to exit to MS DOS from the TRACKSTAR

5-1. From Control Menu

1) Press <F1>-<ESC> keys in TRACKSTAR mode.
   You will have "Control Menu" on the screen

2) Type "X", then you will have the following message

You have 2 options to go to MS DOS

[1] Terminate TRACKSTAR operation
    You will need to run TRACKSTAR Utility software to go to TRACKSTAR

[2] Let TRACKSTAR run in background
    -40K of P/C RAM will be reserved for TRACKSTAR Utility software.
    -If you do not have enough RAM, you may have difficulty running MS DOS software. In this case, reset the system <CTRL-ALT-DEL>.
    -To go to TRACKSTAR mode again from MS DOS mode, type <ESC-Key> with <ALT-Key> pressed.

---

3) If you want the TRACKSTAR to run in the background, select Option 2. In this case, 40K of P/C RAM will be reserved for the TRACKSTAR Utility program. And you can use <ALT><ESC> keys to return to TRACKSTAR while the MS DOS program is running. This option will be convenient to work with MS DOS and TRACKSTAR simultaneously. But if you do not have enough P/C memory, reserving 40K of RAM may cause some problems when you try to run the programs which require more memory.

4) When you have selected Option 2, then return to TRACKSTAR again, and try to exit to MS DOS from Control Menu. It will assume that you have selected Option 2 without giving any message.

5) If you select Option 1, no memory will be reserved for the TRACKSTAR, and you can use the whole P/C memory. In this case, you cannot use the toggle key "<ALT><ESC>" to return to TRACKSTAR again. You should run the TRACKSTAR Utility program to go to the TRACKSTAR.

5-2. Using Toggle Key "<ALT><ESC>"

1) If you press "<ALT><ESC>" keys in the TRACKSTAR mode, you will get out immediately to MS DOS without any message.

2) This action has the same effect as Option 2 followed by "X" at the Control Menu.

Once you reserve the TRACKSTAR Utility program in P/C memory, the only way to eliminate it is by resetting the P/C with "Reset Button" or with <CTRL><ALT><DEL> Keys.

6. PROGRAMMING THE FUNCTION KEYS

TRACKSTAR takes advantage of the Function Keys that are present on the P/C keyboard. They are easy to program and wonderful to use. Instead of typing in CATALOG each and every time it is needed, all you need to do is to press <F2>. To program the function keys, get into the Control Menu by pressing "<F1>-<ESC>" keys, and then type "F". The following menu will appear:

```
------------------------------------------------------------------
****DISPLAY/MODIFY FUNCTION KEY SETS****

There are 8 sets of Function Key commands

[C] Current Selected Function Key Set: 1

[1] Function Key Set 1
[2] Function Key Set 2
[3] Function Key Set 3
[4] Function Key Set 4
[5] Function Key Set 5
[6] Function Key Set 6
[7] Function Key Set 7
[8] Function Key Set 8

[S] Save Modified Function Key Sets on Diskette
[Return] Exit

Please Select Function!
------------------------------------------------------------------
```

If, for example, you select 1, the following screen will appear:

```
-----------------------------------------------------------------
CURRENTLY DEFINED FUNCTION KEYS FOR SET 1

-F2 :  CATALOG^M
-F3 :  LIST^M
-F4 :  CALL-151^M
-F5 :  SAVE
-F6 :  DELETE
-F7 :  THANK YOU FOR BUYING TRACKSTAR
-F8 :  BLOAD
-F9 :  BRUN FID^M
-F10 : 1234567890123456789012345678901234

Pls Enter Command!

[M] Modify Function Key

[Return] Exit to Function Key sets
-----------------------------------------------------------------
```

This menu displays what is currently in the selected Function Key set. There are eight Function Key sets with Function Keys available in each one(remember, F1 is dedicated to the Control function in all sets). Each key can accept up to 32 characters.

If you press the <Return> key you return to the DISPLAY/MODIFY FUNCTION KEY SETS MENU.
If you press M, the EDIT FUNCTION KEY MENU will appear:

```
-----------------------------------------------------------------
Editing Commands

   [DEL key]  Deletes Character

   [END key]  Ends Input to Function Key

   [FUNCTION key]  To be Modified

   [Return] Exit to Function Key Sets
-----------------------------------------------------------------
```

This menu allows you to select the function key you want to modify by pressing the appropriate function key. You may then type in any characters you wish(including Return key) as your desired command. The only keys that will not be entered are the DEL and END keys. These keys allow you to edit your commands. The DEL key will delete characters in case you make a mistake and the END key will tell the system when the command string is complete.

When END is selected you will return to this menu and you may choose another function key in this group to modify or you may press the [Return] key to return to the CURRENT DEFINED FUNCTION KEY MENU. Remember, in all cases, selecting a function will move you to the next menu and [Return] key will return you to the previous menu.

Once you have completed modifying the Function Key Set and have pressed [Return] key twice to return you to the DISPLAY/MODIFY FUNCTION KEY SETS menu you will have the option to save the commands onto your TRACKSTAR Utility diskette. Press S at this menu to save. Then, the following message will appear:

```
-----------------------------------------------------------------
Please be sure that TRACKSTAR Utility files are in Drive-X of PC

[Return] To Save
[ESC]    To Abort

Please Enter Command!
-----------------------------------------------------------------
```

If you only need to use these commands this time and do not wish to save them, then you can use Return key to EXIT the menu and the commands will be available until the system is shut-off.

7. THE FILE TRANSFER UTILITY

The File Transfer Utility diskette has two different file transfer programs. One is for Apple DOS and the other is for Apple CP/M. The front side of the diskette is Apple DOS file transfer and the flip side is Apple CP/M file transfer. Both programs do not have operating systems. For each of the file transfer programs you have to boot its own DOS(Apple DOS 3.3 or Apple CP/M).

In this section we will discuss only Apple DOS file transfer utility program. The operation of the CP/M file transfer utility is similar to the Apple DOS file transfer. With the exception that no data conversions are done.

Only text or binary files of Apple DOS 3.3 can be transferred to MS DOS. Some form of conversion to binary or text files will be needed to be done to any other files. Any type of MS DOS files can be transferred to Apple DOS 3.3 but they will become either a binary or text file on the Apple side.

To transfer files, boot the APPLE DOS 3.3 MASTER in TRACKSTAR mode. When you see the APPLE prompt, insert APPLE DOS File Transfer Utility diskette in Drive D1. Type "BRUN FILE TRANSFER <Return>." The following menu will appear:

```
-------------------------------------------------------------------
-FILE TRANSFER UTILITY V. 2.5-
   for Apple DOS 3.3 File

(C) Diamond Computer Systems, Inc., 1984

ENTER SELECTION

(A) Directory of Apple Disk

(I) Directory of MS DOS Disk

(T) File Transfer

(Q) Quit

-Apple is the trademark of Apple Computer, Inc.
-------------------------------------------------------------------
```

The prompts will lead you easily through the transfer.

WILD CARD FUNCTION

When the program asks for the source file name: Entering *.* will transfer the entire disk. All files which are binary or text will be transferred.

When transferring from Apple DOS to MS DOS the Apple filename will be converted into MS DOS format. Since MS DOS filenames are limited to FILENAME.EXT format while Apple filenames are limited to 30 characters some changes have to occur. The changes include:

1) All characters past the eighth are ignored. If a period(.) is after the eighth character a three character extension is added(SUPERDUPERFILE becomes SUPERDUP. APPLE TEST.TXT1 becomes APPLETES.TXT).

2) Spaces are ignored (FILE TRANSFER UTILITY becomes FILETRAN).

3) Illegal characters cause difficulties. Remove control characters and /'s from the Apple names. If present they will cause "IBM disk read error" message to appear and the transfer to stop.

THE TRANSFER

1) Text File Transfer

From APPLE DOS to MS DOS

- Bit 7 of ASCII data of text file are reset
- Carriage Return code(HEX 8D) is converted to Carriage Return(HEX 0D) and Line Feed(HEX 0A)
- End of file mark of Apple DOS(Code 0) is converted to Control-Z(HEX 1A)

From MS DOS to APPLE DOS

- Bit 7 of ASCII data of MS DOS text file are set
- Line Feed code(HEX 0A) is neglected
- End of file mark of MS DOS(Code 1A) is converted to end of file mark of Apple(Code 0)

2) Binary File Transfer

From APPLE DOS to MS DOS

- The first two bytes of the APPLE binary file, which is the start address of the binary file, are cut out
- The second two bytes of the APPLE binary file, which is the file length of the binary file, are cut out
- The rest of the file is transferred without any conversion

From MS DOS to APPLE DOS

- Two bytes(HEX 00, HEX 10) are added at the beginning of the file, which indicates the start address(HEX 1000) of binary file. The binary file will be loaded at the address HEX 1000 when you "BLOAD FILENAME"
- The next two bytes are added according to the file size. They indicate the size of the binary file
- The rest of the file is transferred without any conversion

SPECIAL TIPS

Converting an APPLE DOS binary data file into an MS DOS text file

Often Apple application programs will store their text and data files in binary format to save storage space and quicken access time. To convert a APPLE DOS binary data file into an MS DOS text file do the following:

1) Transfer the APPLE binary file to MS DOS mode using the File Transfer Utility

2) Transfer the newly transferred MS DOS file back to APPLE DOS as a text file. This will dissemble the file

3) Transfer the file a third time, this time as a text file to the MS DOS world. The file can now be inspected using the TYPE command

8. THE DIFFERENCES BETWEEN TRACKSTAR AND AN APPLE II+

8-1. The Keyboard

The IBM keyboard offers many keys not available on the Apple II+, but in reality there are only a few things to watch for:

1) Many programs were not designed to support lower case letters(because the Apple II+ does not support lower case letters). If you are running one of these programs make sure the CAPS LOCK key is released. If CAPS LOCK is on, then the lower case letter will appear.

2) Some programs use the "shift wire" modification to create lower case. The TRACKSTAR cannot support this function.

3) Some of the keys on the IBM keyboard are simply not supported on the Apple II+ keyboard so they may cause unusual results. (The IBM ALT key for example).

4) The IBM cursor keys are not supported on Apple programs.

8-2. Disk Drive

As we discussed, the TRACKSTAR offers the special feature of supporting an external Apple drive to ensure full Apple copy protection compatibility. When this drive is installed your Apple programs will boot from this drive. If it is not installed then they will boot from Drive A of the IBM.

8-3. Slots

The Apple (much like the IBM) provides special features through cards that plug into slots inside the computer. In the Apple environment, access to these features is obtained by calling a specific slot #(through the command PR# or IN#).

Don't worry, Apple software will do this automatically. It is important to know which slots are supported when you read a software users manual to ensure you can run the particular program.

Although the TRACKSTAR does not have the physical slots, like an Apple, it does support several of the slot functions. Following are the slots supported, their features, and their limitations.

SLOT	Comments
Slot 0	Supports Language Card commands and functions. The language card function of the TRACKSTAR provides an additional 16K of memory. This is required for Pascal programming, and some programs that use the extra memory for storage(example is Apple Writer II).
Slot 1	Parallel Interface. Through tradition, Slot 1 is used for the parallel interface to the printer. TACKSTAR emulates a standard parallel interface card to Apple programs but with an important twist. The data it receives for printing is passed along to the IBM parallel interface which is connected to the printer. If your IBM does not have a parallel interface then this slot is not supported. Also, and this is important, even though every command you send to the printer is supported, just as it would be on the Apple II+, it does not support special graphic commands, such as you find on a Grapple Card, nor does its hardware emulate printer cards. In other words, programs that expect certain hardware to be present will not function with the system(this only seems to be a program with some graphics programs).
Slot 2	Serial Interface. Through tradition this is used for communications and for serial interface printers. Just as with Slot 1, the TRACKSTAR emulates this function through software and passes the information to the IBM for execution. In the case of serial interface the IBM will need to be initialized first. Please read the IBM manual for the serial interface card to understand how to do this. The limitation of the serial interface is in the area of communications. Most communication programs are written for specific hardware(to improve performance). Since

the TRACKSTAR does not provide that hardware it may not work with a particular communication program.

Slot 3 — Supports 80 column operation and supports all Videx Videoterm Commands for 80 column mode. To engage 80 column mode simply type PR#3 at the Apple prompt(]). All further operations will be in 80 column mode until you reset TRACKSTAR from the control menu. Remember, don't worry, the Apple software(such as Pascal and CP/M) will handle this automatically.

However just as with the other slots, this is created in software and not hardware. Some programs that use the 80 column mode do so through hardware and therefore cannot be used with the 80 columnfeature. Fortunately, Pascal programs and CP/M programs do not have this limitation.

Slot 4 — Not supported

Slot 5 — Not supported

Slot 6 — Supports standard Apple disk controller commands.

Remember: If an Apple compatible drive is connected to the TRACKSTAR it will act as the drive in Slot 6, Drive 1. If an Apple compatible drive is not provided then Drive A of the IBM will act as Drive 1. In either case Drive B will act as Drive 2.

Slot 7 — Z80 card

8-4. Operating Systems

Now that you have the TRACKSTAR running and have learned how to access it, we will discuss how to use it with the various operating systems it can support. First, here is a list of the Operating Systems that are supported by the TRACKSTAR with comments on each:

O/S	Comments
DOS 3.2	Early Apple Operating System. Supported by the TRACKSTAR with the limitations outlined under Keyboard, Drive and Slots.

DOS 3.3 — Most prolific Apple Operating System. Supported by the TRACKSTAR with the limitations outlined under Keyboard, Drive and Slots.

DOS 3.3 is the most popular of the Apple Operating Systems. Literally tens of thousands of programs are available for this operating environment. Because of this we have made every effort to provide the most compatibility between this system and the TRACKSTAR. Unlike MS DOS or CP/M which require that an Operating System disk be booted first or copied to the program disk, almost every DOS 3.3 program you can buy already includes the Operating System as part of the program. This makes DOS 3.3 programs very easy to run and use.

ProDOS — Apple's latest Operating System. The TRACKSTAR will support this system to the extent of an Apple II+. It will not support all programs written for ProDOS and the Apple //e. It also will not support the Hard Disk features of the System.

Apple Pascal — UCSD Pascal version for the Apple. 80 column is automatically displayed. Supported by the TRACKSTAR with the limitations outlined under Keyboard, Drive and Slots.

CP/M 2.2(56K and 60K) — These are the two versions of Digital Research's popular business Operating System. 80 Column mode is automatically displayed. Supported by the TRACKSTAR with the limitations outlined under Keyboard, Drive, and Slots.

data files in binary format to save storage space and quicken access time. To convert a APPLE DOS binary data file into an MS DOS text file do the following:

1. Transfer the APPLE binary file to MS DOS mode using the File Transfer Utility

2. Transfer the newly transferred MS DOS file back to APPLE DOS as a text file. This will dissemble the file

3. Transfer the file a third time, this time as a text file to the MS DOS world. The file can now be inspected using the TYPE command

Transferring BASIC Files back and forth

APPLE DOS to MS DOS

1. Using the following capture program, convert your Apple BASIC program into a text file. The file can be transferred as a text

file. If transferred with BASIC extension, the file can be used immediately as an MS DOS BASIC program(as long as it does not violate any MS DOS BASIC commands or syntax).

MS DOS to APPLE DOS

1. Save the BASIC file with an A extension. This will convert the file to ASCII format(COPY FILENAME.BAS FILENAME.A)

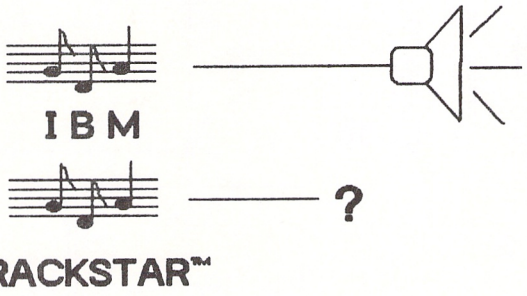
2. Transfer it to APPLE DOS using the APPLE DOS File Transfer Utility as a text file

3. EXEC(EXEC FILENAME.A) the program to convert it to a BASIC program(EXEC is an Apple command that enters a program into memory as if it was typed in from the keyboard). Many Apple BASIC programmers will create the BASIC program on a word processing program and EXEC the file into the computer when they are finished.

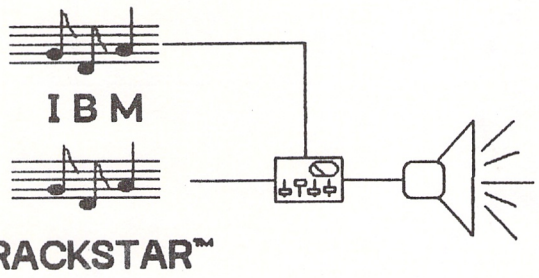
7

The TRACKSTAR™ board must be electrically inserted between the IBM PC motherboard speaker port and the speaker.
The speaker output of the IBM PC and the TRACKSTAR™ system are mixed on the TRACKSTAR™ board; this lets the TRACKSTAR™ share a single speaker with the IBM PC.

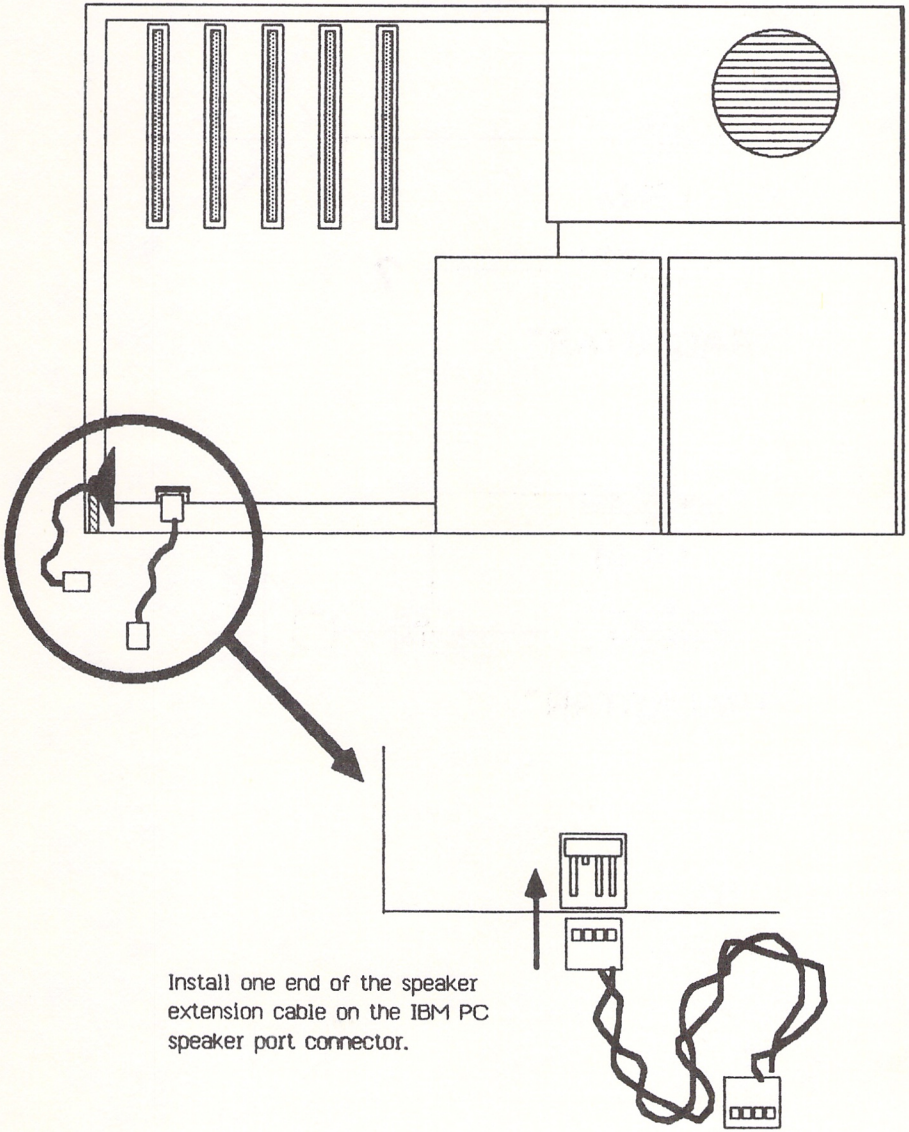
Before:



After:

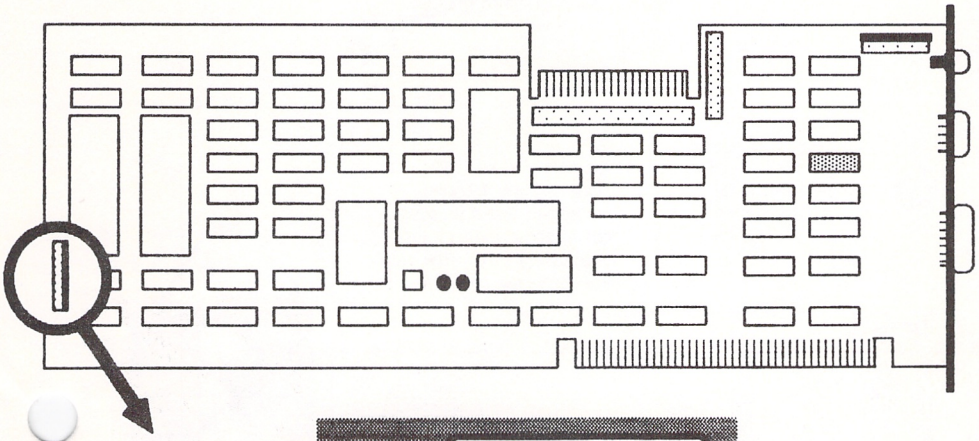


8 After unplugging the speaker plug from the speaker connector on the IBM PC motherboard, take the speaker port extension cable supplied with your TRACKSTAR™ board (the cable that contains a speaker port plug on each end) and plug one end of the extension cable onto the IBM PC's motherboard speaker port connector.

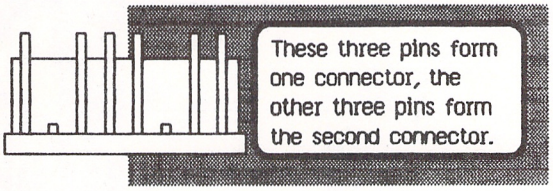


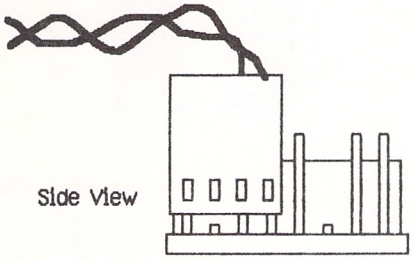
Install one end of the speaker extension cable on the IBM PC speaker port connector.

9 On the lower left hand side of the TRACKSTAR™ circuit board (when viewed from the top, component side), there is a small connector containing six pins. This connector is the speaker port connector for the TRACKSTAR™ system. Note that *two* speaker plugs can be attached to this port. Attach the free end of the speaker extension cable (whose other end was connected to the IBM PC motherboard in Step Eight) to three of these pins, connect the speaker plug to the other three.



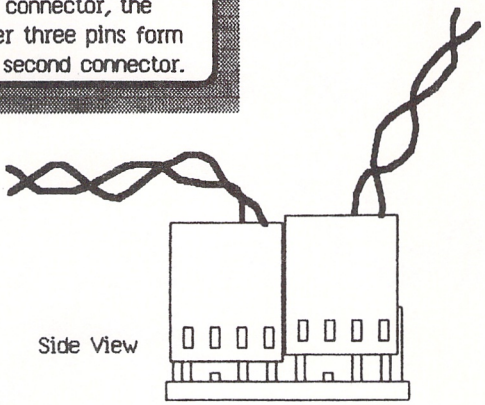
Side View





Side View

Attach IBM Speaker plug to TRACKSTAR™ speaker connector.

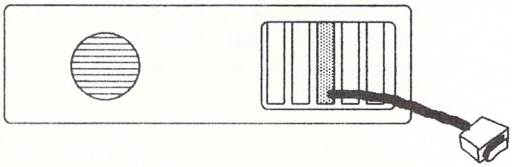


Side View

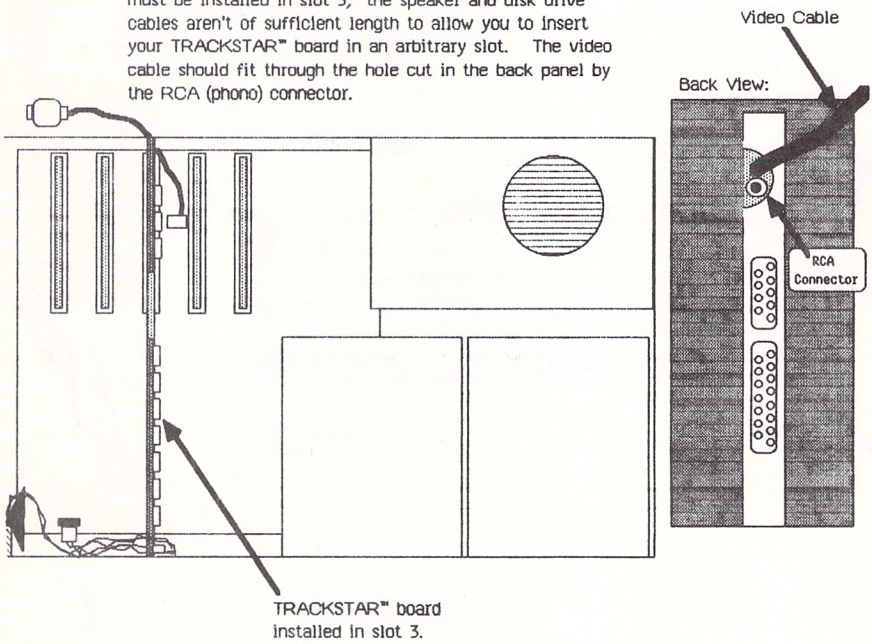
Then install the speaker extension cable to the other side of the TRACKSTAR™ speaker connector.

Note: both speaker ports on the TRACKSTAR™ board are common. You can attach either speaker plug to either side of the connector.

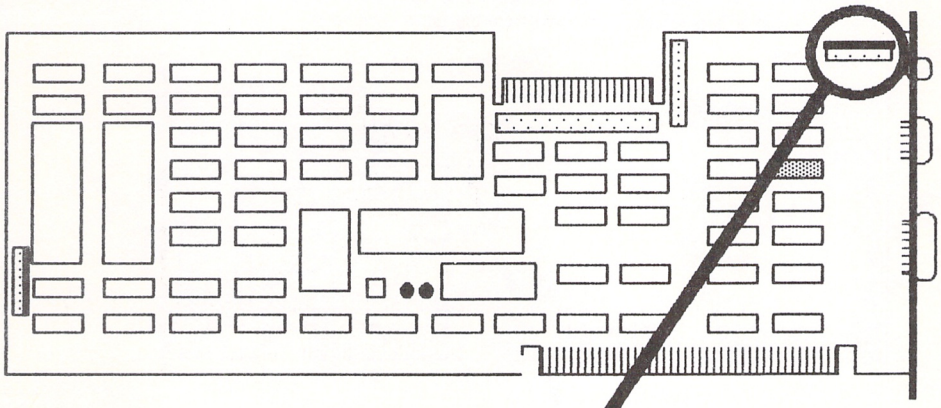
10 Get the video cable from the TRACKSTAR™ package. The video cable has a DB9 connector on one end and a molex connector on the other. Feed the molex connector through the opening for slot three in the back of the IBM PC.



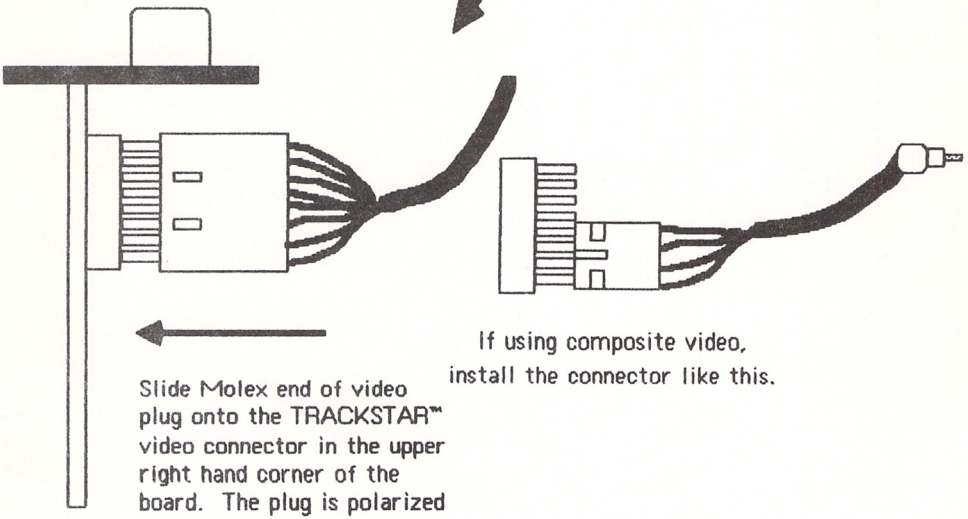
11 Now install the TRACKSTAR™ in slot 3 of your IBM PC. While there is no electrical reason why the TRACKSTAR™ must be installed in slot 3, the speaker and disk drive cables aren't of sufficient length to allow you to insert your TRACKSTAR™ board in an arbitrary slot. The video cable should fit through the hole cut in the back panel by the RCA (phono) connector.



12 Next, connect the molex end of the video cable to the video connector on the TRACKSTAR board.



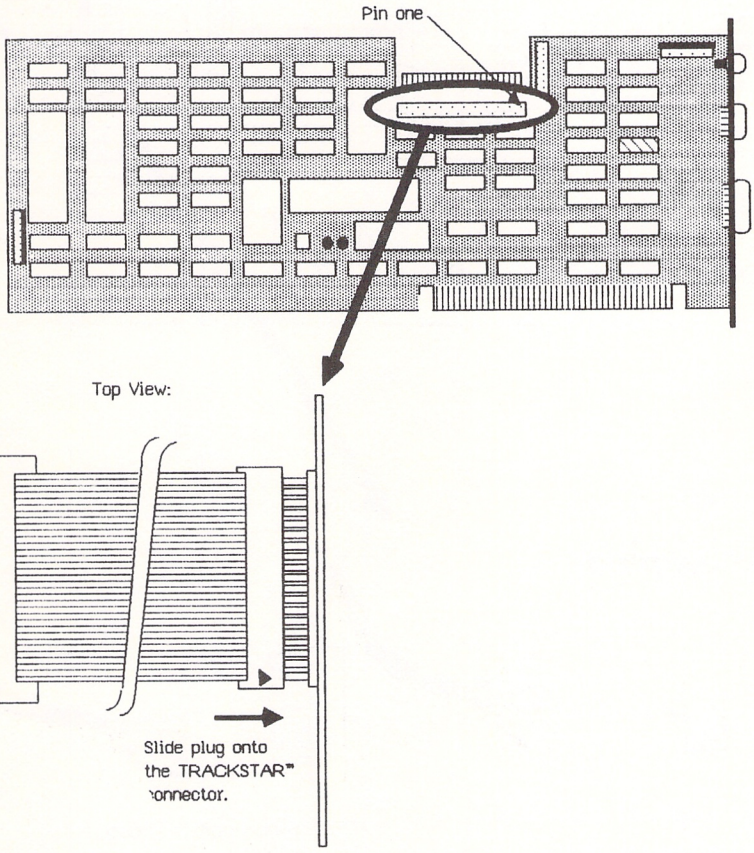
Top View:



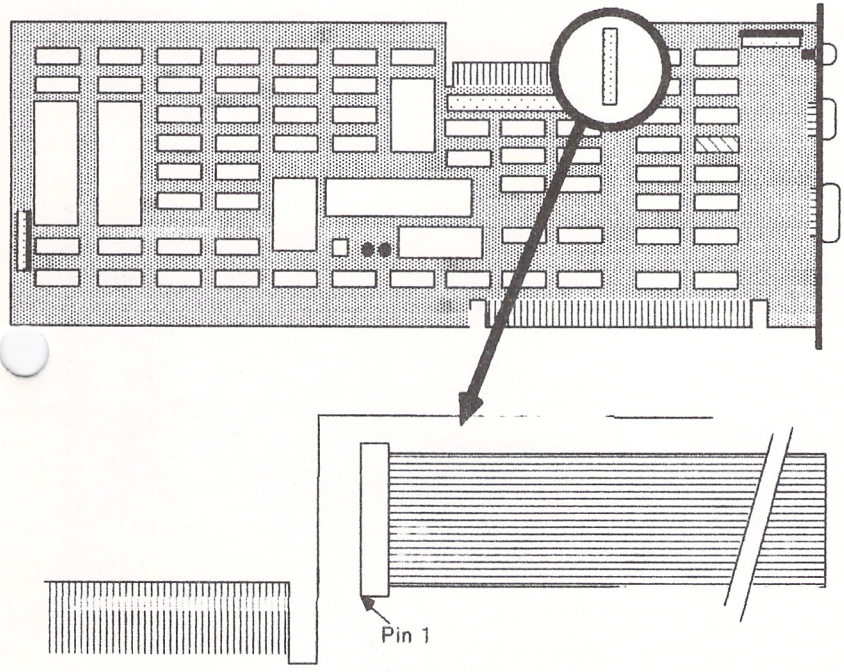
Slide Molex end of video plug onto the TRACKSTAR™ video connector in the upper right hand corner of the board. The plug is polarized so it is difficult to plug it in backwards.

If using composite video, install the connector like this.

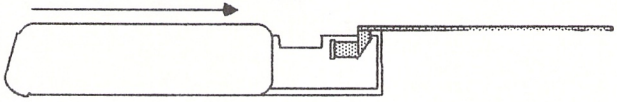
13 Now it is time to connect the TRACKSTAR™ system between the IBM's floppy disk controller and the floppy disk drives. First, attach the disk controller connector cable supplied with the TRACKSTAR™ system to the connector diagrammed below. Pin one is marked with a small arrow on the cable's plug. Attach the plug to the connector on the circuit board with the arrow aligned with the pin in the upper right hand corner of the connector. The cable should point upwards at this point, however, trust the arrow even if the cable points downward. In either case, make sure that the cable is positioned up and away from the IBM motherboard.



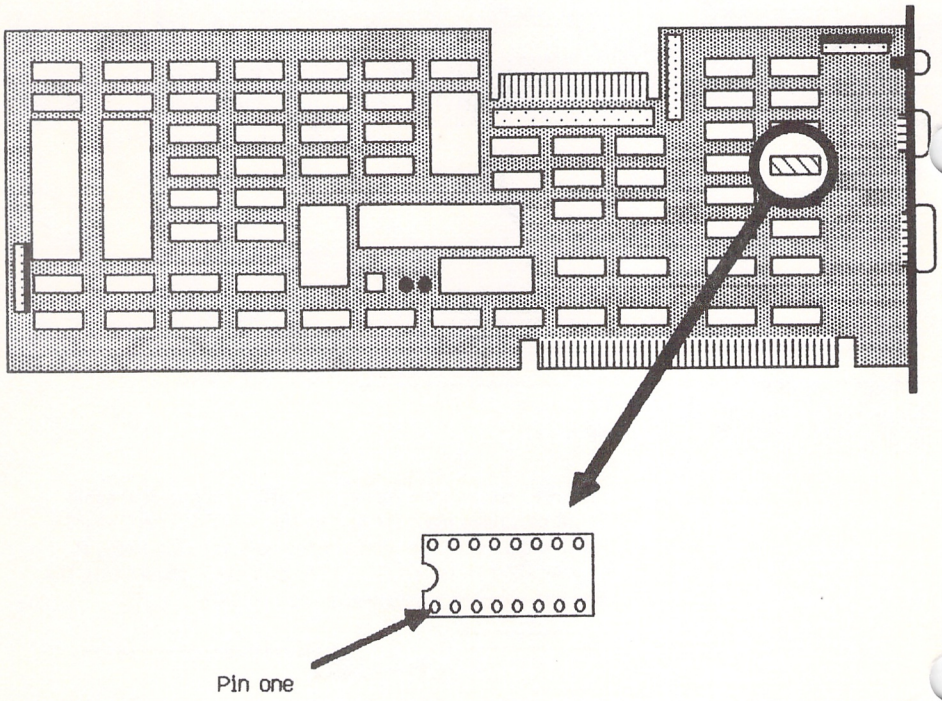
14 If you wish to install an Apple compatible floppy disk drive on your TRACKSTAR™ system, now is the best time to do so. This step is optional, if you do not have an Apple compatible floppy to install, continue on to step 15. The Apple compatible floppy disk drive cable should be plugged onto the 20-pin connector shown below. Th disk drive cable *must* always exit towards the rear of the IBM PC CPU box.



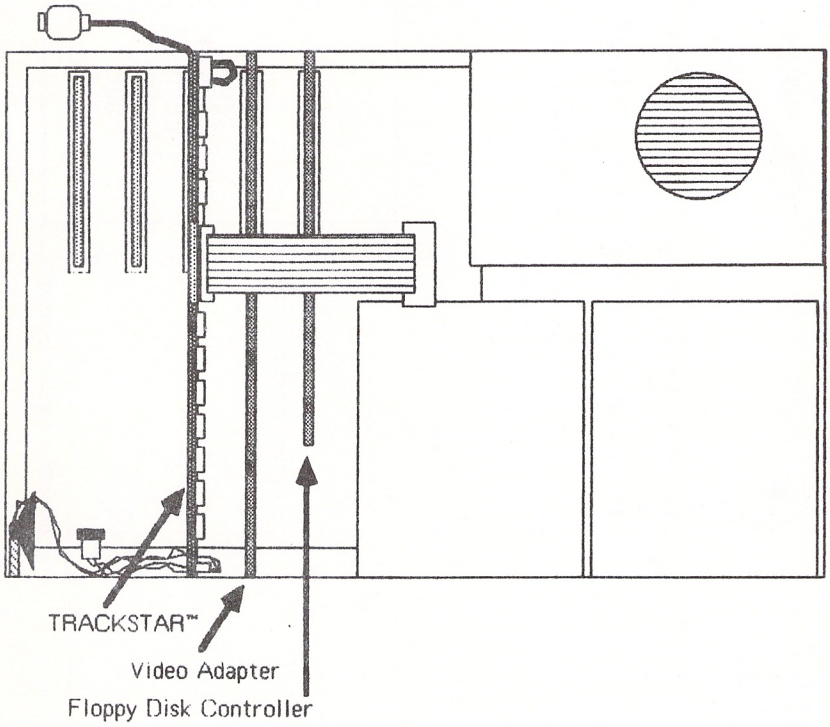
For best results, the Apple compatible floppy disk cable should be folded over the top of the TRACKSTAR™ board and it should exit the IBM via the hole for slot two. If all your IBM slots are filled, the disk cable should exit the IBM between the IBM's backplate and cover.



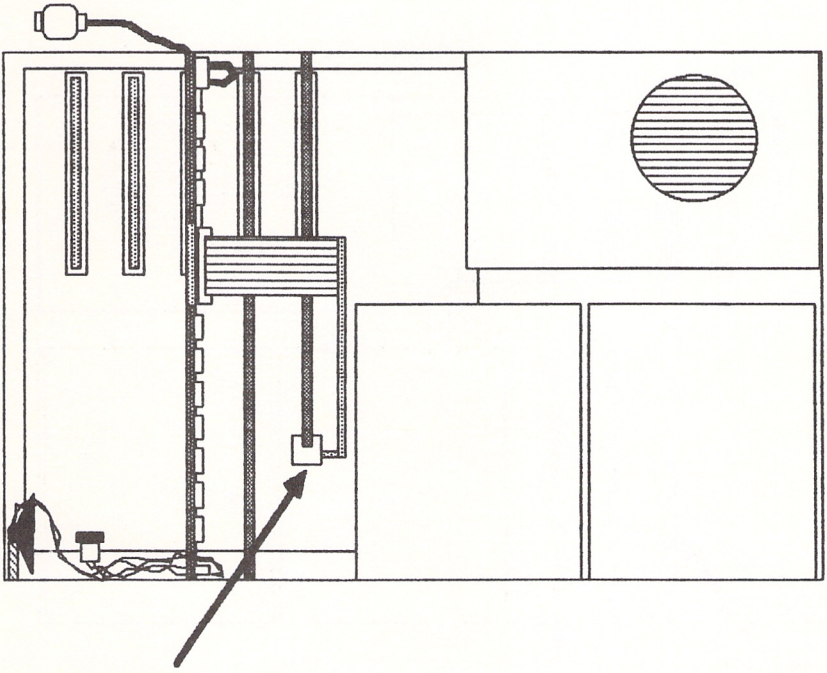
15 If you want to install an Apple compatible game controller, this is the most convenient time to do so. The Apple game port is a sixteen-pin dual in-line connector. It is the empty socket located four chips down on the right hand side of the TRACKSTAR™ board. Pin one is located at the lower left hand corner of the socket. If you aren't using slot two, the game controller cable can be fed into the IBM through the hole for slot two. The unused hole above the cassette and keyboard sockets is another likely location where you can feed the game controller cable into the IBM. Installing an Apple compatible game device is an optional step. If you do not wish to install an Apple game I/O device, skip to step 16. Note: if an Apple compatible game device is plugged into the game I/O socket, you cannot use IBM compatible devices on the DB15 connector on the back of the TRACKSTAR™ board.



6 Install the IBM video adapter in slot four and the floppy disk controller in slot five. Route the disk controller cable from the TRACKSTAR™ board over the top of the Color Graphics and disk controller cards.

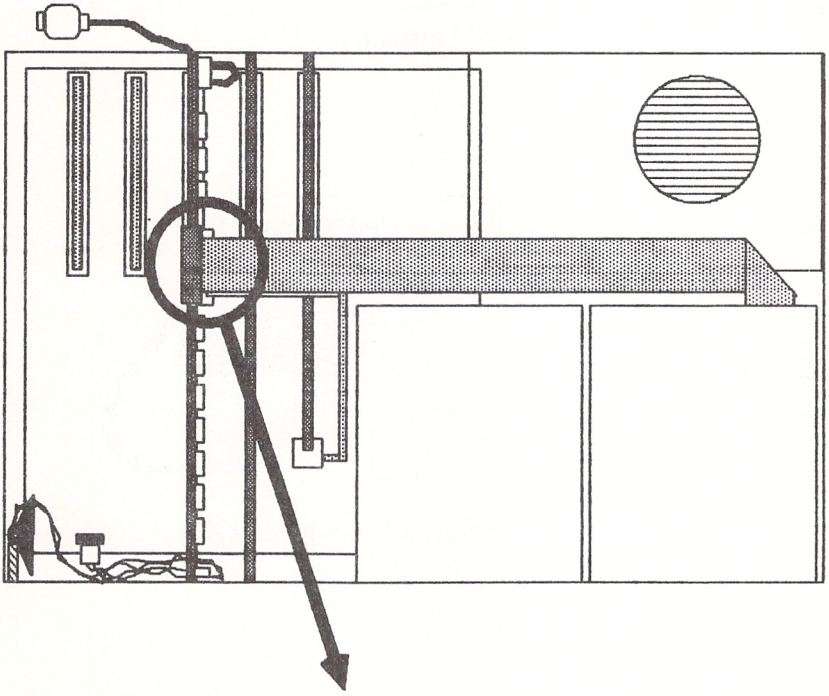


17 Route the cable around to the front of the disk controller card and plug it onto the edge-card connector on the disk controller. Note that the cable exits to the right of the disk controller card when viewed from the top.

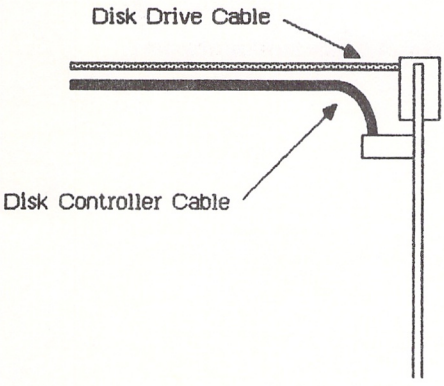


Plug TRACKSTAR™ disk connector onto the IBM floppy disk controller card.

18 Now attach the disk *drive* cable to the edge-card connector on the top of the TRACKSTAR™ board. The cable should exit to the right of the TRACKSTAR™ board when viewed from the top.

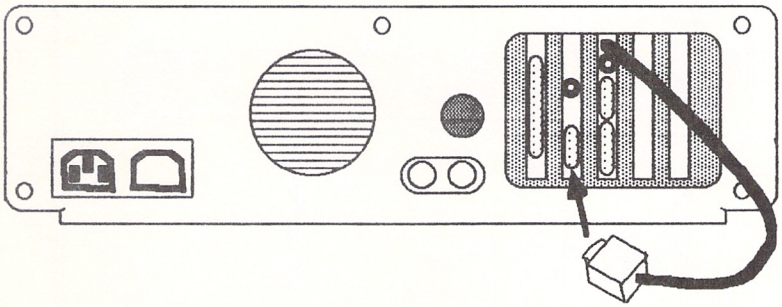


Back View:

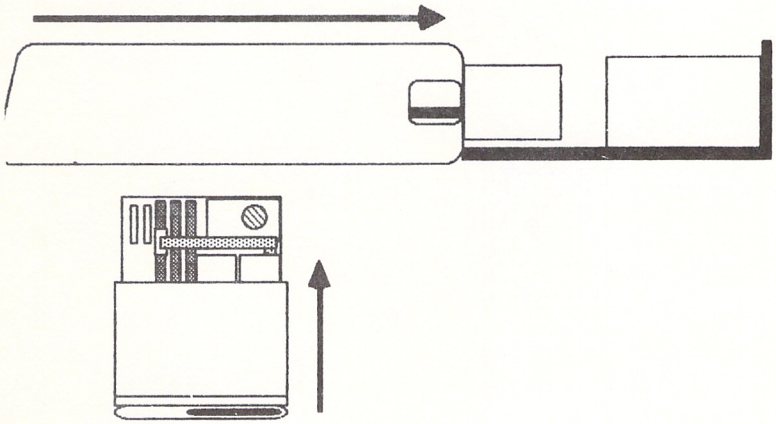


19 Install any remaining cards in your IBM PC's slots. You should use slot one before using slot two (try to leave slot two open for Apple compatible disk drives and game controller cables)

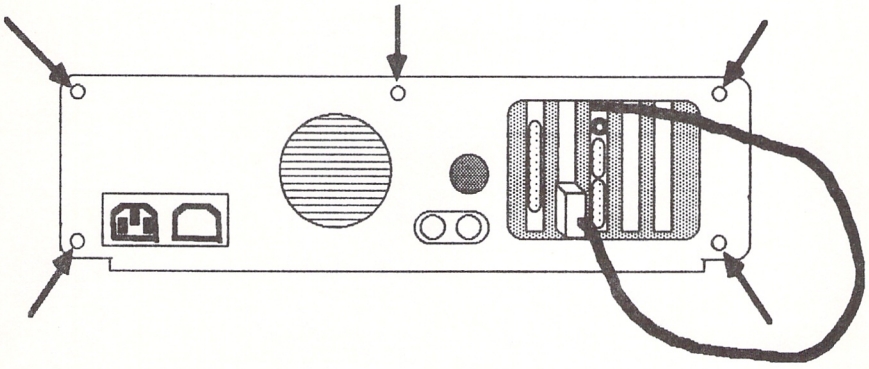
20 Attach the male DB9 end of the video connector to the female DB9 connector on the Color Graphics Adapter.



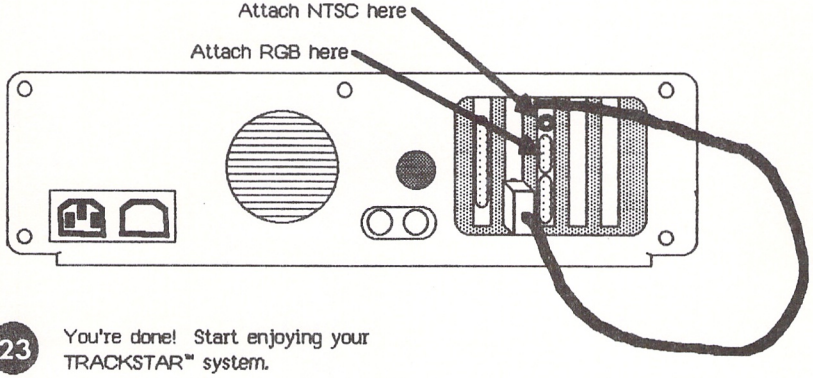
21 Carefully slide the computer case back over the IBM PC's chassis.



22 Tighten the screws on the back of the IBM PC chassis as shown below:



23 Attach your RGB display connector to the female DB9 connector on the TRACKSTAR™ board. If you are using a composite (NTSC) video monitor, connect it to the RCA (phono) jack on the TRACKSTAR™ card.



23 You're done! Start enjoying your TRACKSTAR™ system.

Chapter One: Introduction

What the TRACKSTAR Does for Your IBM PC

Differences Between the TRACKSTAR and an Apple II

IBM Keyboard vs. Apple II Keyboard

The TRACKSTAR Video Display

- Screen Memory
- The Text Modes
- The LORES Graphics Modes
- The HIRES Graphics Modes

The TRACKSTAR Speaker Port

The Apple Cassette Port

What the TRACKSTAR Board Does for Your IBM PC

The TRACKSTAR board plugs into a slot of your IBM PC and allows you to run software written for a 64K Apple II or Apple II Plus. Providing complete Apple compatibility was not an easy task. The IBM PC is not hardware compatible with an Apple II. The TRACKSTAR board represents a complete "Apple on a board" for IBM PC users.

The first step towards complete Apple compatibility required the inclusion of special 80-column display hardware and an Apple compatible floppy disk controller. With its built-in 40 and 80-column display the TRACKSTAR is capable of running most of the software available for the Apple II Plus. The Apple disk drive port lets you (optionally) add an Apple compatible disk drive which is capable of reading any Apple formatted disk, even those with special copy protection techniques that normally cannot be read with an IBM disk drive.

Finally, TRACKSTAR's software system provides compatibility with Apple's DOS 3.3 environment. This opens your IBM PC up to the widest selection of software written for any microcomputer system.

In addition to DOS 3.3 compatibility, the TRACKSTAR system also opens up Apple CP/M capability on your IBM PC. Apple CP/M is the most widely accepted CP/M format (there are more installations of Apple CP/M than any other format for CP/M), and the TRACKSTAR system is the only product available for the IBM PC which supports this important CP/M format.

So the TRACKSTAR system lets your IBM PC run the largest base of software available--the combined software base of Apple, IBM, and CP/M applications. No other system available for the IBM PC matches the versatility of the TRACKSTAR system.

Differences Between the TRACKSTAR Board and the Apple II.

Although the TRACKSTAR board enables your IBM PC to run most Apple application programs, there are still a few differences between the TRACKSTAR system and an Apple II of which you should be aware.

First of all, the TRACKSTAR board is a single plug in board which you install in an IBM PC. References to Apple's physical hardware layout (such as slots and various chips on the motherboard) you may find in various pieces of literature simply do not apply to the TRACKSTAR board. In particular, the TRACKSTAR board does not support Apple peripheral slots. So you cannot attach Apple peripheral devices to your TRACKSTAR system.

The TRACKSTAR system only supports two disk drives, referenced as Apple drives one and two in slot six. When using the built-in IBM disk drives, many copy protected programs will not operate properly because the IBM disk drives do not support a special feature known as half-tracks. To overcome this limitation, the TRACKSTAR system contains a special

connector that lets you install an Apple II disk drive. With the optional Apple disk drive installed, your TRACKSTAR system can easily read any Apple formatted floppy disk.

The Keyboard

The first and most obvious difference between your TRACKSTAR system and a standard Apple II is the IBM keyboard. The TRACKSTAR system uses the IBM PC's keyboard for user input.

Although similar to the Apple's keyboard, the IBM PC's keyboard contains several additional keys not found on the standard Apple keyboard. Furthermore, there are a couple of keys found on the Apple //e and Apple //c keyboards which are not present on the IBM PC keyboard.

The newer Apple //e and Apple //c keyboards contain two special keys labeled with an open Apple and a solid Apple. These two keys are connected to the push button zero and push button one ports on the GAME I/O connector. You can easily simulate these two keys by purchasing a pair of game paddles for your TRACKSTAR board and using the buttons on those paddles in place of the Apple keys.

The IBM PC keyboard, on the other hand, contains several keys which are not available on the Apple II keyboard. First of all, unlike the older Apple II keyboards (but like the newer Apple //e and Apple //c keyboards), the IBM PC keyboard supports upper and lower case characters. Since the original Apple II keyboards only supported upper case, many programs available for the Apple II require upper case input. For example, Applesoft BASIC and DOS 3.3 both require all reserved words and keywords (words which are designated command names and are not to be used as variable names) in uppercase. Lower case versions will not be recognized. This problem is easily rectified when operating a program which requires upper case input: simply press the caps lock key on the IBM keyboard.

The IBM PC keyboard provides several function keys which are not present on a standard Apple keyboard. Except for F1, which initiates the TRACKSTAR reset sequence, these keys are programmable by the user. To reset your TRACKSTAR system, press F1 followed by ESC. This action will display the reset selection menu. When the menu appears, you can reset the TRACKSTAR system by pressing "R" (for reset) or "B" (for BOOT). Note that F1 ESC B is similar to pressing CTRL-Open Apple-RESET on an Apple //e.

The remaining function keys are user programmable and will return any string you program them to return (see Chapter Two).

In the NUMLOCK mode, the IBM's numeric keypad behaves like, well, a numeric keypad. In the cursor keypad mode, the keycodes transmitted to the TRACKSTAR board are modified for use by Apple software. The LEFT arrow returns $88 (backspace), the RIGHT arrow returns $95 (CTRL-U), the UP arrow returns $8B (CTRL-K), and the DOWN arrow returns $8A (CTRL-J). Finally, the HOME key returns the pair ESC ($9B) followed by "@" ($C0).

This Key	Returns
LEFT ARROW	$88
RIGHT ARROW	$95
UP ARROW	$8B
DOWN ARROW	$8A
HOME	$9B, $C0

Most high level languages that operate on the TRACKSTAR board provide statements which allow you to input data from the keyboard (e.g., INPUT and GET in BASIC, READ and READLN in Pascal). However, your programs can easily read data directly from the Apple's keyboard should the need arise.

The TRACKSTAR's keyboard port is an eight-bit input port which provides the information obtained from the IBM PC's keyboard whenever a key is pressed. The keyboard supports the standard seven-bit ASCII character code. The ASCII code for the key pressed is returned in the low-order seven bits of the keyboard port. The high order bit is set when a key is available at the keyboard port; the high order bit is cleared when a character is not available at the keyboard port.

Your programs can test the keyboard port to see if a character is available by testing bit seven. If it is set, then the ASCII character can be read from the port.

Once your program has read the character from the keyboard port, your program should tell the TRACKSTAR hardware that the key has been taken by accessing the keyboard strobe port. Accessing this I/O port clears the high order bit (the highest numbered bit in a byte) of the keyboard port so that further accesses to the keyboard port will not read the character already read from the keyboard port. After accessing the keyboard strobe port, the high order bit of the keyboard port will remain cleared until another key is pressed on the keyboard.

Note that when reading the keyboard port, the value you obtain is the ASCII code of the character with the high order bit set. Most computers use ASCII codes with the high order bit clear. The Apple II (and the TRACKSTAR system) usually likes to see its characters with the high order bit set. Many high level languages (like floating point BASIC and Pascal) invert the high order bit to make the ASCII code lie in the range 0..127. However, if you read the keyboard by directly accessing the hardware, keep in mind that the ASCII code returned will lie in the range 128..255.

The TRACKSTAR's keyboard port and keyboard strobe port are located in two memory locations in the TRACKSTAR's memory space. The keyboard port is located at address $C000 (49152/-16384) in memory. The keyboard strobe port is located at address $C010 (49168/-16368). You can read the keyboard port directly from BASIC with a PEEK command; you can clear the keyboard strobe with a PEEK or a POKE instruction. From 6502 assembly language, a load accumulator instruction (LDA) can be used to read the keyboard data from the keyboard port, and a store accumulator (STA) or an LDA instruction can be used to access the keyboard strobe port to clear the current character at the keyboard port.

The TRACKSTAR Video Display

While operating under the TRACKSTAR system, all output to the video display is controlled by the TRACKSTAR board. By cabling your IBM color graphics adapter through the TRACKSTAR board, the TRACKSTAR system can intercept the video output from your IBM PC and feed the video output from the TRACKSTAR board to the display instead.

The TRACKSTAR display supports nine different display modes. The TRACKSTAR system supports two 40x24 text modes, two 40x48 LORES graphics modes, two 280x192 HIRES graphics modes, and an 80x24 80-column text mode. In addition to these text and graphics modes, the TRACKSTAR system supports several combinations of mixed text and graphics modes, such as 40x40 LORES graphics with four lines of text (two pages), and 280x160 HIRES graphics with four lines of text (two pages).

1. 40x24 (page 1)	40x24 (page 2)
2. 40x48 (page 1)	40x48 (page 2)
3. 280x192 (page 1)	280x192 (page 2)
4. 280x160 (page 1)	280x160 (page 2)
5. 80x24	

Note that the TRACKSTAR creates the graphics displays on its own board, the IBM color graphics adapter is not used to generate the color graphics for the TRACKSTAR system.

Screen Memory

The information used to create a text or graphics display on the TRACKSTAR board is stored in the on-board RAM of the TRACKSTAR system. Two blocks of 1K bytes are reserved for the two TEXT and LORES graphics screens. The HIRES graphics screens are stored in two separate areas of memory. The two HIRES screens each require an 8K block of memory.

In the text and LORES graphics modes, the same memory locations are used to display the video information on the display screen. The page one text and LORES graphics screens occupy most of the memory locations between $400 and $7F7. The page two text and LORES graphics screens occupy memory locations $800 through $BF7.

In the HIRES graphics modes, memory locations $2000 through $3FF7 are utilized by the page one, or primary HIRES graphics screen. The page two, or secondary, HIRES screen occupies memory locations $4000 through $5FF7.

The bytes in a display page are not arranged in a straight-forward fashion. Sequential characters on a line are stored in sequential memory locations, but sequential lines on the screen are not mapped into sequential bytes of memory (i.e., the first byte of the second line on the screen does not follow the last byte of the first line on the screen). There are also several "holes" in the screen memory where the information stored is not displayed on the video screen. Those locations are referred to as "I/O scratchpad locations" and they are used by various peripheral devices in the TRACKSTAR system. Care should be exercised when accessing the screen memory since disturbing these I/O locations may adversely

affect the operation of TRACKSTAR peripheral devices.

There are ten memory locations in the TRACKSTAR's memory space which are used to switch between the various video modes. Any access (read or write) will cause the specified action to be taken. These locations are arranged in pairs. One of the addresses in the pair turns a specific mode on, the other location turns it off. The screen pairs are

LOCATION			
HEX	DECIMAL		DESCRIPTION
$C020	49184	-16352	Turns on the 40-column display mode.
$C028	49192	-16344	Turns on the 80-column display mode.
$C050	49232	-16304	Display a graphics mode.
$C051	49233	-16303	Display a text mode.
$C052	49234	-16302	Display all text or graphics.
$C053	49235	-16301	Display mixed text and graphics.
$C054	49236	-16300	Display page one text or graphics.
$C055	49237	-16299	Display page two text or graphics.
$C056	49238	-16298	Display LORES graphics mode.
$C057	49239	-16297	Display HIRES graphics mode.

Note that the "all text/graphics", "mixed text/graphics", "display LORES" and "display HIRES" switches are only functional if a graphics mode is selected with the $C050/$C051 graphics/text mode switch.

The TEXT Modes

The TRACKSTAR system supports three text modes. In the forty column mode the TRACKSTAR hardware supports two 40x24 text pages. When displaying either of these 40 column text pages, TRACKSTAR displays 24 lines of text, each line containing 40 characters. In the eighty column mode, the TRACKSTAR system displays 24 lines of text. Each line of text in the eighty column mode holds eighty characters.

The LORES (low resolution) Graphics Modes

In the LORES graphics mode, the TRACKSTAR system displays the contents of the 40-column text buffer in a graphics format. Each byte in the display screen is formatted as two rectangular colored blocks. In the LORES graphics mode, the TRACKSTAR hardware displays the contents of the text/LORES graphics pages as an array of 40x48 blocks.

Each byte normally used to display a single character on the text screen is used to display two blocks in the LORES graphics mode. The low order four bits of each byte in screen memory are used to store the color of the top block; the high order four bits of each screen memory byte are used to hold the color information of the lower block on the screen. The colors, numbered from zero to fifteen are

0	0	Black
1	1	Magenta
2	2	Dark Blue
3	3	Purple
4	4	Dark Green
5	5	Gray I
6	6	Medium Blue
7	7	Light Blue
8	8	Brown
9	9	Orange
10	$A	Gray II
11	$B	Pink
12	$C	Light Green
13	$D	Yellow
14	$E	Aquamarine
15	$F	White

The HIRES Graphics Modes

The TRACKSTAR hardware supports two high resolution graphic pages. In the HIRES mode, the TRACKSTAR can display 53,760 dots arranged in a 280x192 matrix.

The dots on the screen can take on one of six different colors depending on the position of the dots on a horizontal line. Dots in an even column must be black, violet, or blue. Dots in an odd column must be black, green or red. If both an odd and an even column contain set bits, the color will be white. Each byte on the screen must be either a violet/green byte or a blue/red byte. This combination is selected by the high order bit of the byte. It is not possible to mix green and blue, green and red, violet and blue, or violet and red in the same byte.

Each byte in screen memory is used to display seven dots on the screen. The low order bit is plotted towards the left side of the screen, and the seventh bit is plotted towards the right side of the screen. The eighth bit is used to select the green/violet and blue/red pairs.

The TRACKSTAR Speaker Port

The TRACKSTAR system supports a fully Apple compatible speaker port. Any access to memory location $C030 (49200/-16336) causes the IBM's built in speaker to move in or out, opposite its current position. By reading this address, and causing the speaker cone to move in or out, you can create a small click on the speaker. By repeatedly accessing the speaker port, you can create tones and other sounds at the speaker port.

Note that certain 6502 write operations actually perform a read before write operation. Therefore, when writing to the speaker port using certain 6502 addressing modes the speaker's position will be toggled twice in the space of one microsecond. Since the frequency response of the speaker built into the IBM PC does not extend up to 1Mhz (nor could you hear it if it did), these write operations tend to leave the speaker in the same position it was in before the write operation. Therefore, you should only use read operations (like PEEK and LDA) to access the speaker port.

The Apple Cassette Port

Some minor differences may exist between the operation of the TRACKSTAR board and a standard Apple II. For example, the Apple II supports a cassette tape storage device. This hardware support is not provided on the TRACKSTAR system. In fact, the I/O addresses normally dedicated to the cassette port are used by the TRACKSTAR hardware to activate and deactivate the 80-column display. So if you encounter a program which makes use of the Apple II cassette port (an extremely rare situation these days), the program will not work properly on the TRACKSTAR board.

Chapter 2 : Running Canned Program

(Intentionally missing chapter, refer to users manual)

Chapter Three: Standard I/O on the TRACKSTAR

Standard Output

The Stop-list Feature

The TEXT Window

INVERSE, FLASHING, and NORMAL Text

Standard Input

RDKEY

GETLN

Escape Editing

Standard Output

Most programs written for the Apple II and TRACKSTAR systems require some sort of input from the keyboard and they need some mechanism for sending output to the video display. These chores are handled by special subroutines inside the TRACKSTAR monitor ROM.

The standard output subroutine in the TRACKSTAR monitor is often referred to as the "COUT" (Character OUTput) subroutine. COUT takes a character and passes it on to the currently activated peripheral device. The default peripheral device is the TRACKSTAR video screen. When information is sent to the video screen, the TRACKSTAR software ignores all control characters except return, backspace, linefeed, and the bell character.

The COUT subroutine maintains an invisible output cursor on the screen. Whenever a character is output via the COUT subroutine, the COUT routine places the character on the screen at the current cursor position and then advances the cursor one position to the right. When the cursor approaches the right hand side of the current text window (see the description below), the cursor is moved to the beginning (left hand side) of the line immediately below the one the cursor is currently on. If the cursor is on the bottom line when it reaches the right hand side of the text window, the contents of the entire screen are moved up one line and the cursor is moved to the beginning of the newly blanked line at the bottom of the screen. This operation is called scrolling.

If a line feed is sent to the COUT subroutine, then the COUT subroutine moves the cursor down one line without affecting the cursor's horizontal position. If the cursor is located on the last line of the text window when a line feed character is encountered, the screen is scrolled up one line.

The bell character, when received by the COUT subroutine, produces a 1Khz tone for approximately 0.1 seconds. It does not affect the cursor's position.

The backspace character will move the cursor one position to the left without otherwise affecting the data on the display screen. If the cursor is at the left hand side of the text window when a backspace character is printed, the cursor is moved to the end of the previous line on the screen. If the cursor is at the beginning of the first line on the screen, it is moved to the end of the same line.

The Stop-List Feature

Every time a carriage return is sent to the video display via the COUT subroutine, the TRACKSTAR monitor checks the keyboard to see if a key was pressed since the last return was sensed. If a key was pressed and it was a CTRL-S, then the monitor program stops until another key is pressed at the keyboard. If the key is any character except CTRL-C, the stop-list code eats the character and continues processing. If the key pressed was a CTRL-C, then the stop-list code passes the CTRL-C back to the code which called the COUT subroutine. This allows high level languages like Applesoft to abort processing whenever a CTRL-C is encountered.

The TEXT Window

Under normal circumstances, the text sent to the TRACKSTAR's display screen is printed on the 40x24 physical screen. However, by changing certain zero page locations, you can define almost any logical screen which is a subset of the physical screen. Once a logical screen is defined, all output via COUT is confined to the boundaries of the logical screen.

Four locations at addresses 32, 33, 34, and 35 (hex $20, $21, $22, and $23) are used to define the screen left side starting column, the screen width, the top of screen row number, and the screen height (respectively). Whenever the TRACKSTAR system is reset, the values 0, 40, 0, and 24 are loaded into these respective locations. This sets up the logical screen whose upper left hand corner is located at character coordinate (0,0), the left and top edge define this value. The screen's width is 40, and its height is 24. With these default values, the logical screen and the physical screen are one and the same.

By changing these values, you can protect a portion of the screen. Whenever COUT writes a character to the screen, all characters are output to the logical screen. Whenever the screen is cleared or scrolled, only the data within the logical screen is affected. This provides your programs with the power to control the placement of text on the screen, and it gives your programs the ability to protect portions of the screen from erasure.

Location 32 ($20) contains the position of the left hand side of the screen. This location should contain a value in the range zero through thirty-nine. If you store a value larger than thirty-nine into this location it may stop your program. All columns on the screen whose column position is less than the value stored in location 32 are protected from the COUT subroutine.

Location 33 ($21) holds the screen width. This value must never be greater than forty minus the value currently in location 32. For example, if location 32 contains eight, then location 33 should never contain a value greater than thirty-two since 8+32=40. If location 33 contains a width which, when added to the contents of location 32, produces a sum greater than forty, your program may stop.

Location 34 ($22) contains the row number of the top line of the logical screen. Normally this byte contains a zero. Under no circumstances should this location contain a value greater than twenty-three, nor should it contain a value greater than the value in location 35. Locations 32 and 34 define the upper left hand corner of the logical screen on the physical screen.

Location 35 ($23) contains the coordinate of the bottom edge of the screen plus one. This byte should contain a value in the range 1..24 (1..$18). These locations are summarized as follows:

Function	Location DEC	Location HEX	Minimum/Normal/Maximum Value DEC	Minimum/Normal/Maximum Value HEX
Left edge	32	$20	0 / 0 / 39	0/ 0/ $17
Width	33	$21	0 /40 / 40	0/ $28/ $28
Top edge	34	$22	0 / 0 / 23	0/ 0/ $17
Bottom edge	35	$23	1 /24 / 24	0/ $18/ $18

INVERSE, FLASHING, and NORMAL Text

The COUT routine has the ability to print text sent to the screen in normal or inverse. Location 50 ($32) is used to determine whether normal or inverse characters are printed.

In the 40-column mode, if memory location 50 contains $FF, all text will be printed using normal characters. If location 50 contains $3F, then all characters sent to the screen will be printed in inverse video. Any other value stored into location 50 will cause the COUT subroutine to produce strange results. In particular, storing $7F into location 50 will cause certain characters to appear in inverse while other characters appear flashing.

In the 80-column mode, storing $FF into location 50 produces normal text, storing $7F into location 50 produces inverse text. Other values will modify the text sent to the screen and produce gibberish.

The Standard Input

Many programs require user input from the TRACKSTAR (IBM) keyboard. This is accomplished by two subroutines in the TRACKSTAR monitor program. The RDKEY subroutine reads a single character from the keyboard, the GETLN subroutine reads up to 255 characters, terminated by a carriage return, from the keyboard.

RDKEY

The RDKEY performs three different tasks: it waits until a key is pressed at the IBM keyboard and then reads the key pressed; it prompts the user to press a key by making the cursor visible on the screen; and it increments a pair of memory locations while waiting for a keypress from the keyboard. Since these two memory locations (78 and 79 or $4E and $4F) are incremented several thousand times per second, and the average user hits a key on the keyboard at fairly random intervals, the value which ends up in these two memory locations is truly randomized by the RDKEY subroutine. The random value stored in these two locations is used by several application programs to randomize a random number generator.

GETLN

Most Apple programs read text from the keyboard a line at a time. To facilitate this form of keyboard input, the TRACKSTAR monitor provides a special subroutine call GETLN. The GETLN subroutine repeatedly calls the RDKEY routine to read individual characters from the keyboard. As the keys are entered into the system, the GETLN subroutine copies the characters into an input buffer and prints the character to the video display. Upon encountering a carriage return character the GETLN subroutine terminates and returns to the code which called GETLN. The code that called GETLN can read the characters typed from the input buffer which begins at address 512 ($200) in the TRACKSTAR memory space.

The GETLN subroutine provides a few features which aren't supported by the simple RDKEY subroutine. First of all, at any time you can erase the entire contents of the input line buffer by pressing CTRL-X. When you press CTRL-X, the GETLN subroutine prints a slash ("/") and a carriage return to notify you that the input line buffer's contents have been purged. Once you press CTRL-X, GETLN is expecting you to enter a brand new line of text from the keyboard.

In addition to the delete line function, the GETLN subroutine provides several other editing capabilities as well. Pressing the back arrow or backspace key deletes the previous key typed into the input buffer. The forward arrow key (on the keypad, this key generates the same code as CTRL-U) copies the character currently under the cursor into the input line buffer. This key is useful, in conjunction with the ESC key editing commands described in the next section, for copying data which appears on the display screen into the input buffer.

Escape Editing

Another useful feature of the GETLN subroutine is that it supports ESCape editing. Whenever you press the ESC key on the IBM PC keyboard, the GETLN subroutine enters the escape mode. In this mode, eleven of the keys on the IBM PC keyboard take on a special meaning. These keys are @, A, B, C, D, E, F, I, J, K, and M. All of these keycodes perform pure screen editing commands. They do not, in any way, modify the contents of the Apple's input buffer.

The two character sequence ESC-A moves the cursor one position to the right. Neither the ESC nor the A code is entered into the GETLN input buffer. This command is useful when you wish to skip over some input before copying additional characters on the screen with the right arrow key. Simply press ESC-A for each character you wish to skip over on the screen.

The ESC-B editing function is used to move the cursor back one character without affecting the contents of the input line buffer. This command can be used to back the cursor up so you can recopy some information already typed on the current line.

The ESC-C command moves the cursor down one line. Like the other ESC commands, it does not affect the contents of the input buffer. ESC-C is useful for moving the cursor down to the next line before copying data on the line with the right arrow key.

The ESC-D command moves the cursor up one line. This command is extremely useful for positioning the cursor on the previous line on the screen so you can recopy the line or copy a portion of the previous line into the current line.

The ESC-E command clears the screen from the current cursor position to the end of the current line. The ESC-E command is useful for clearing the remainder of the line so that you can continue typing without a lot of confusing text on the same line as you're typing on.

The ESC-F command clears the screen from the current cursor position to the end of the screen. It can be used like the ESC-E command—to clear up the garbage on the screen and make it less confusing to read.

The ESC-@ function is called "home and clear". It clears the entire screen and positions the cursor in the upper left hand corner of the screen window.

The ESC-I, J, K, and M functions are similar to the ESC-D, B, A, and C functions (respectively). ESC-I moves the cursor up, ESC-J moves the cursor left, ESC-K moves the cursor right, and ESC-M moves the cursor down. Unlike the ESC-D, B, A, and C functions, the ESC-I, J, K, and M functions move the cursor and leave the computer in the ESC mode. This allows you to repeatedly press I, J, K, and M without having to retype the ESC key for each movement. To terminate the multiple-key ESC mode, simply press the space bar.

Chapter Four: The TRACKSTAR Monitor Program

Introduction to the Apple Monitor

Examining Memory

Changing the Contents of Memory

Moving a Range of Memory

Comparing Two Ranges of Memory

Creating and Running Machine Language Programs

Examining and Changing Registers

Special Monitor Commands

Creating Your Own Monitor Commands

Introduction to the TRACKSTAR Monitor

Buried within the TRACKSTAR monitor is a 6502 debugger program affectionately called the "monitor". Although when someone refers to the monitor he is usually referring to the entire 2K of code in the range $F800..$FFFF, the monitor program is a term applied to the 6502 debugger program which also lies within the confines of the TRACKSTAR monitor. The monitor program is an extremely useful utility for examining and modifying the contents of memory on the TRACKSTAR board.

The TRACKSTAR monitor program beings at address $FF69. You can enter the monitor program with a CALL -155 or a CALL -151 command from BASIC. Upon entering the monitor program you will be greeted with an asterisk prompt. The monitor program accepts three types of input: addresses, values, and commands. The monitor program expects all numeric input (addresses and values) in a hexadecimal format. Decimal input is not allowed.

Examining Memory

If you simply type the address of a memory location after the asterisk monitor prompt and press return, the monitor program will display (in hex) the contents of that memory location. The monitor program saves the address entered as the last opened location. This address is also used (as you will see in a moment) as the next changeable location.

If you type a period (".") on an input line followed by a hexadecimal address, the monitor program will display every location between the last opened location and the address specified after the period. Assuming, of course, that the address specified after the period is greater than the last opened address. If you specify an extremely large ending address and you wish to abort, use the TRACKSTAR reset sequence.

You can specify a complete memory range to display by simply typing a starting address followed by a period followed by an ending address. All of the memory locations between the two addresses will be displayed on the screen.

Once you've opened an address, you can display up to eight consecutive memory locations by pressing the return key. Each time return is pressed another eight consecutive locations will be displayed.

Changing the Contents of Memory

The colon (":") command is used to modify a memory location. To modify the content of some memory location simply specify the address of the memory location followed by a colon and the data you wish to store into the specified memory location.

To change additional sequential locations, simply type addtional data values after the colon. For example,

F2:00 05 FF F0 AA

stores $00 into location $F2, $05 into location $F3, $FF into location $F4, $F0 into location $F5, and $AA into location $F6. This operation would leave location $F7 opened. At any time you can store data into the last location opened by simply typing a colon followed by the data you wish to store after the asterisk prompt.

Moving a Range of Memory

The TRACKSTAR monitor program includes a built-in memory move command. The syntax for the move command is

{dest} < {start} . {end} M

where {dest} represents the destination address, {start} represents the starting address of the block of memory you wish to move, and {end} is the last address in the memory block you wish moved. The monitor program uses a moveleft logarithm, so the move will only be performed sucessfully if the source and destination blocks do not overlap or if the destination address is less than the starting address.

Comparing Two Ranges of Memory

The Monitor "V" (for verify) command is used to compare two blocks of memory to see if they are identical. The syntax for the verify command is

{dest} < {start} . {end} V

The monitor program compares the source range to the destination range and prints the address of any location in the destination range which is not identical to the corresponding byte in the source range. The bytes at both locations are also displayed.

Creating and Running Machine Language Programs

6502 machine language programs are created with the help of a program known as a 6502 assembler. While there are several popular 6502 assemblers available for the Apple II which run on the TRACKSTAR system, the LISA v2.6 interactive assembler from Lazerware is probably your best choice. Once you create a machine language program, the TRACKSTAR monitor can help you debug it. Two commands, in addition to those already discussed, are extremely valuable when attempting to run programs under the monitor's control: the monitor "G" command and the monitor "L" command.

The "G" (for Go) command is used to run a machine language program from the TRACKSTAR monitor. The syntax for the GO command is

{address}G

This monitor command will execute a 6502 JSR instruction to the specified address.

The "L" (for List) command is used to disassemble the 6502 instructions at some specified address. The syntax for the list command is

{address}L

The monitor, when it encounters this command, will disassemble 20 consecutive 6502 instructions beginning at the specified address.

Examining and Changing Registers

By pressing CTRL-E followed by a return you can list the contents of the 6502 registers. Actually, you do not list the contents of the registers themselves, but the contents of five memory locations which will be loaded into the accumulator, X-register, Y-register, program status register, and the stack pointer the next time the "G" command is executed.

After displaying the contents of the 6502 registers, the monitor CTRL-E command leaves the address of the accumulator save location open so you can issue a colon command to modify the contents of the register save locations. For example, if you issued the command sequence:

```
CTRL-E
:0 1 2 3 4
```

you would load the 6502 accumulator with zero, the X-register with one, the Y-register with two, the program status register with three, and the stack pointer save location with four. Please note that the stack pointer will not be loaded from the register save area when the "G" command is executed.

Special Monitor Commands

The monitor "I" command can be used to select inverse character output, the monitor "N" command can be used to return monitor output to normal. Note that while in the inverse mode, only output from the computer is converted to inverse. The GETLN subroutine always displays user input using normal text output.

The PRINTER command, CTRL-P, can be used to turn on a peripheral device in a specified slot. The syntax for the CTRL-P command is

{n}CTRL-P

where "n" is a slot number in the range 0..7. A slot number of zero selects the video display page, any other value selects the specified slot. Consult Appendices One and Two for slot assignments on the TRACKSTAR board. A virtual peripheral must be present and it must be capable of supporting an output device or the system may hang. Note that selecting slot six will cause the disk drive to boot an Apple-compatible

floppy disk.

The KEYBOARD command, CTRL-K, is used to select an input peripheral device. Its syntax and usage is almost identical to the PRINTER command. The only difference is that the peripheral device selected must be capable of acting as an input device. A slot number of zero will select the IBM keyboard as the current input device.

The TRACKSTAR monitor will also support simple hexadecimal arithmetic. Simply enter a line using the format

{value} + {value}
{value} - {value}

and the monitor program will print the result of the addition or subtraction. Note that the values are limited to eight bits.

The TRACKSTAR monitor contains built-in cassette I/O commands. The "R" command attempts to read data from the cassette port, the "W" command attempts to write data to the cassette port. Since the TRACKSTAR hardware does not support an Apple compatible cassette port, these functions are not available and if you attempt to use them you may hang the system.

Creating Your Own Commands

The CTRL-Y command, when encountered in the input buffer while the monitor is parsing the line, transfers control to location $3F8. By placing a jump instructon at this address, you can create your own monitor command. Only three bytes are set aside for this command, so you should place a 6502 JMP instruction at this address and locate the code for your user-defined monitor command somewhere else (locations $300..$3BF are usually available).

Special Monitor Memory Locations

Sixteen bytes in page three are reserved for use by the monitor program and Applesoft BASIC. These locations are

$3F0 & $3F1	Holds the address of the BRK instruction handler subroutine (normally $FA59).
$3F2 & $3F3	Soft entry vector. Holds the address of the reset subroutine if not a power up situation. (Normally contains $E003 or $3D3.)
$3F4	Power up byte, used to determine if this is a power up reset.
$3F5...$3F7	Holds a JMP instruction to the code which handles the Applesoft "&" command.
$3F8...$3FA	Holds a JMP instruction to the code which handles the monitor CTRL-Y command.
$3FB...$3FD	Holds a JMP to the subroutine which handles non-maskable interrupts.
$3FE...$3FF	Holds the address of the subroutine which handles interrupt requests (IRQs).

Appendix One: The TRACKSTAR I/O Structure

Appendix One: The TRACKSTAR I/O Structure

The I/O structure of the TRACKSTAR system is where the greatest difference lies between the Apple II and TRACKSTAR systems. An Apple II system provides several peripheral connector slots, similar to those in your IBM PC, which allow expansion of the Apple II system. Due to physical and cost constraints, the TRACKSTAR board does not provide this level of expandability. Although Apple protocol states that a peripheral card can be installed in any slot, 99% of all Apple programs which access peripheral devices plugged into one of the Apple's slots expect a printer interface in slot one, a serial communications interface in slot two, an eighty-column adapter in slot three, and a disk driver controller in slot six. Therefore, the TRACKSTAR system has assigned these important functions to their respective slots. The PRINTER and COMMUNICATIONS ports are assigned to the IBM PRINTER and COM devices; hardware is provided on board to support the eighty-column and disk drive controllers.

SLOT 1: Parallel Printer Interface

SLOT 2: Serial Communications Interface

SLOT 3: 80-column Adapter

SLOT 6: Disk Drives

SLOT 7: Z-80 Card

A memory map of the TRACKSTAR system may be helpful in the following discussion. The memory layout for the TRACKSTAR system is

```
$0000..$BFFF: 48K user RAM space

$C000..$C00F: Keyboard data port
$C010..$C01F: Keyboard strobe clear port
$C020..$C027: Activates 40-column screen
$C028..$C02F: Activates 80-column screen
$C030..$C03F: Clicks the speaker
$C040..$C047: Eight-bit communication port to IBM
$C048..$C04F: Four-bit communication port to IBM
$C050       : Selects graphics mode
$C051       : Selects text mode
$C052       : Selects full graphics mode
$C053       : Selects mixed graphics and text
$C054       : Selects primary (page one) screen
$C055       : Selects secondary (page two) screen
$C056       : Selects LORES mode
$C057       : Selects HIRES mode
$C058..$C05F: Repeats $C050..$C057 functions
$C060       : Not assigned (Cassette in an Apple II)
$C061       : Push button one input
$C062       : Push button two input
$C063       : Not assigned (Push button three on Apple II)
$C064       : Game control input #0
```

```
$C065       : Game control input #1
$C066       : Game control input #2
$C067       : Game control input #3
$C068..$C06F: Repeats $C060..$C067
$C070..$C07F: Clears game controller port

$C080/ $C084: Select upper 16K of RAM with bank0 (in the $D000
              area) selected.
$C081/ $C085: Select upper 12K of ROM.  Two accesses write
              enable the upper 12K of RAM (bank0 is selected)
$C082/ $C086: Select upper 12K of ROM, write protect RAM.
$C083/ $C087: Read upper 12K of RAM (bank 0 selected).  Two
              accesses write enable the RAM.
$C088/ $C08C: Read RAM with bank 1 selected. Write protects RAM
$C089/ $C08D: Read ROM.  Two consecutive accesses write enables
              bank2 of the RAM.
$C08A/ $C08E: Read ROM and write protect RAM.
$C08B/ $C08F: Read RAM (bank1), two consecutive accesses write
              enables the RAM.

$C090..$C0BF: Unassigned.

$C0C0..$C0FF: Floppy disk and 80-column bank selection ports.
              Note that the functions are repeated every 16
              bytes as follows:

$C0x0..$C0x7: Floppy disk head control.
$C0x8       : Turn drive motor on.
$C0x9       : Turn drive motor off.
$C0xA       : Selects drive #1 and 80 column bank #1
$C0xB       : Selects drive #2 and 80 column bank #2
$C0xC       : Strobe data latch for disk I/O.
$C0xD       : Load data latch.
$C0xE       : Prepare latch for input.
$C0xF       : Prepare latch for output.

$C100..$C7FF: Virtual slot peripheral driver programs.
              Slot 1: $C100.
              Slot 2: $C200.
              Slot 3: $C300.
              Slot 4: $C400.
              Slot 5: $C500.
              Slot 6: $C600.
              Slot 7: $C700.
              Note: not all slots are assigned functions by the
              TRACKSTAR system.

$C800..$CBFF: Eighty-column firmware code.
$CC00..$CFFF: Eighty-column display pages.  Two 1K banks
              are bank selected into the memory area.

$D000..$DFFF: Read-only.  Selected by accessing memory
              locations $C081, $C082, $C085, $C086, $C089,
              $C08A, $C08D, or $C08E.

$D000..$DFFF: Bank0 RAM. Selected by accessing location
              $C080, $C083, $C084, or $C087.  Accessing
```

locations $C083 or $C081 twice in a row write enables this bank of RAM.

$D000..$DFFF: Bank1 RAM. Selected by accessing location $C088, $C08B, $C08C, or $C08F. Accessing locations $C089 or $C08B two consecutive times write enables this bank of RAM.

$E000..$FFFF: ROM. Selected by the same locations which select the $D000 bank of ROM.

$E000..$FFFF: RAM. Selected by any location which selects bank0 or bank1 of the $D000 RAM.

Built-in I/O

Most of the built-in I/O devices were described in the introduction to the TRACKSTAR system in Chapter Two.

The TRACKSTAR on-board I/O devices are controlled via a memory-mapped I/O arrangement using 128 locations in the TRACKSTAR memory map between locations $C000 and $C07F, and 64 memory locations between $C0C0 through $C0FF. Thirty-six I/O functions share these 192 locations. Obviously, some functions use more than one location (in fact, up to sixteen locations are reserved for the same function). These locations are outlined in the table above.

DATA INPUTS: The only data input is the port at address $C000 (mirrored in locations $C001..$C00F) which represents the data currently at the Apple keyboard port. The high order bit of this port is similar to the flag inputs described below: the low order seven bits contain the data, in ASCII format, of the character at the keyboard. If the high order bit is set, then data is available at the keyboard; if it is clear, then a key has not been pressed since the last time the keyboard strobe was cleared.

FLAG INPUTS: Several input locations on the TRACKSTAR board are simply flag inputs. You read the status of the input port by reading the specified byte for that port. The single bit result is returned in the high order bit of the input port. Examples of ports which operate in this fashion include the push button inputs and the game controller inputs.

STROBE OUTPUTS: These types of I/O devices are activated simply by reading the memory location associated with the port. To avoid problems, you should never write to one of these locations. Examples of strobe output ports include the KEYBOARD STROBE port, and the GAME CONTROLLER STROBE port.

TOGGLE SWITCHES: One of the on-board devices, the speaker port, is a toggled device. Every time you access the speaker port you toggle the voltage of the speaker output. This action will cause the speaker to move in or out, opposite of its last position.

SOFT SWITCHES: Most of the built-in peripheral ports are accessed via "soft switches". An I/O device which uses soft switches has two memory locations associated with it. One location activates the device, the other deactivates it. The 40/80 column ports, graphics/text mode selection, mix/nomix selection, and HIRES/LORES ports are all good examples of ports which are activated via soft switches.

The CSW/KSW Switches

Locations $36, $37, $38, and $39 in page zero on the TRACKSTAR board are the four locations which control I/O redirection for software running under the TRACKSTAR DOS operating system. Locations $36 and $37 are called the CSW location (CSW stands for Console output SWitch). Locations $38 and $39 are called KSW (for Keyboard SWitch).

The CSW locations contain a pointer to the current output device. Normally, these two locations contain the address of the video display output subroutine at address $FDF0. However, the CTRL-P monitor command and the PR# DOS and Applesoft command can be used to modify the contents of these two locations to redirect output to one of the virtual slots on the TRACKSTAR system. PR#1 (or 1CTRL-P) stores $C100 into CSW, PR#2 stores $C200 into CSW, PR#3 stores $C300 into CSW, etc. PR#0 stores $FDF0 back into CSW.

The KSW switch normally contains $FD1B-- the address of the keyboard read routine in the Apple monitor. These two locations are modified by the IN# and n-CTRL-K commands. IN#n loads $Cn00 into KSW (except for IN#0 which loads $FD1B into KSW).

To turn a peripheral device on, the PR# or IN# (or CTRL-P/CTRL-K) comands are executed with the slot number of the desired peripheral you wish to activate. To turn a peripheral device off, the IN#0 and PR#0 commands are used.

When running TRACKSTAR DOS, the KSW and CSW switches point at subroutines internal to DOS. Whenever you execute a PR#n or IN#n command, DOS maintains its own pointers internal to DOS. Therefore, if you examine the CSW and KSW locations while DOS is active, they will not contain the true peripheral card address.

Appendix Two: An Introduction to the Z80 Subsystem

Appendix Two: An Introduction to the Z80 Subsystem

The TRACKSTAR system, in addition to emulating the native 6502 mode of the Apple II Plus system, supports a Microsoft compatible CP/M system. With its own on-board Z80 chip, the TRACKSTAR system can execute 6502 (Apple) programs or Z80 (CP/M) programs. Note that the CP/M operating system is not included with the TRACKSTAR system, it must be obtained from an independent supplier like Digital Research, Microsoft, or Lifeboat Associates.

When operating under the Apple CP/M environment, the TRACKSTAR system reads from and writes to diskettes in an Apple CP/M software format. This means that the myriad of CP/M programs available in the Apple CP/M format can be directly read by your IBM PC system. Almost every major CP/M application program has been converted to the Apple format, so software should be readily available.

Note that the Z80 chip on the TRACKSTAR board uses the same memory map as that available for a 64K Apple II. When running Apple CP/M on the TRACKSTAR system, you have the equivalent of a 56K or 60K CP/M system (depending on whose software you're using). The TRACKSTAR system supports 16-sector Apple format disks. If you obtain older 13-sector Apple CP/M formatted diskettes, you will need to obtain a copy of the RW13 program from Microsoft in order to read them. Almost all Apple CP/M software is supplied on 16-sector disks so this limitation shouldn't prove to be much of a problem.

While operating in the Z80 (CP/M) mode, the TRACKSTAR board emulates a CP/M system with the following peripherals:

Slot 1: Printer interface (LST: device)
Slot 2: Serial interface (RDR: and PUN: devices)
Slot 3: Eighty column display.
Slot 6: Disk drives A: and B:
Slot 7: Emulated Microsoft Z80 card.

Note that the TRACKSTAR system normally supports a 56K CP/M system. With special CP/M software, available from certain vendors, your TRACKSTAR hardware can be set up to operate like a 60K CP/M system. Software to accomplish this task is available from certain Apple vendors such as Advanced Logic Systems and Microsoft. For more information about the internal operation of an Apple CP/M system, consult the documentation which comes with the CP/M package you obtain.

The Z80 chip on the TRACKSTAR board uses an address translation technique to help make the non-contiguous memory map of the Apple II appear to be contiguous to the Z80. The memory addresses are laid out as follows:

Z80 Address	6502 Address
$0000	$1000
$1000	$2000
$2000	$3000
$3000	$4000
$4000	$5000
$5000	$6000
$6000	$7000
$7000	$8000
$8000	$9000
$9000	$A000
$A000	$B000
$B000	$C000
$C000	$D000
$D000	$E000
$E000	$C000
$F000	$0000

The Z80 on the TRACKSTAR board is turned on by writing to a location in the range $C700..$C7FF in the 6502 memory space. Writing to this location turns off ("freezes") the 6502 and the Z80 begins a reset operation by fetching the instruction at location $0 ($1000 in the 6502 address space). Before writing to this location, your 6502 program should set up a Z80 program at location $0 ($1000) so that the system doesn't hang when the Z80 is activated. To deactivate the Z80 and return to the 6502 mode, the Z80 must write location $E700 ($C700 in the 6502 address space). This returns control to the 6502 which begins executing the instruction immediately following the store instruction which activated the Z80 in the first place. The next time the 6502 writes to the $C700 page of memory, the Z80 will continue where it left off; it will not be reset and jump to location zero again.

Appendix Three:Glossary

Glossary

6502	Numeric designation for the microprocessor chip used in the Apple II and the TRACKSTAR computer systems.
Allocate	To reserve or create space for some variable. Typically, this term is used to describe the processes of setting aside a group of memory locations for a string or array variable.
Array	A collection of variables assigned to sequential memory locations. The array data type allows you to access different variables via a numeric expression. Each variable within an array is called an "element" of that particular array.
ASCII	Acronym for American Standard Code for Information Interchange. The term ASCII almost always refers to the 128 standard data codes assigned to the characters used by most computer systems like the IBM PC and TRACKSTAR.
Assembly Language	A symbolic, human readable, form of machine code (see machine code).
Auto-dimension	A process used by TRACKSTAR BASIC to automatically dimension an array if the array hasn't been explicitly declared before it was used. This dimensioning process is performed automatically by the compiler.
Autostart ROM	This term refers to the executive program which controls the TRACKSTAR (and Apple II) computer system. The autostart ROM (also called the monitor program) occupies the memory in the $F800..$FFFF range of the 6502's address space.
Bank switching	On most eight-bit microprocessors like the 6502, the address range is limited to 64K. To address more than 64K of memory, certain blocks of memory must share the same physical location. However, only one "bank" of this memory can be active at any one time. Bank switching is the process of switching the active memory bank from one bank to the other.
BASIC	Acronym for Beginner's All-purpose Symbolic Instruction Code. BASIC is the most popular computer language running on microcomputers today.

Bit	Each of the 65,536 memory locations in the 6502's 64K address space is further subdivided into eight bits. A bit can hold a single binary piece of information: on or off, true or false, zero or one, etc.
Bit seven	Bit seven is the eighth, or most significant bit of a byte (bits are numbered starting at bit number zero). For several I/O ports on the Apple II and TRACKSTAR systems, bit seven has special significance.
Boot	The process of powering-up or restarting a computer system from scratch.
Bottom up	A technique for analyzing (parsing) a program command structure.
Breakpoint	A user-injected instruction, or section of code which halts a program and prints certain information. Breakpoints are used mainly for debugging a program.
Byte	The universal atomic unit of program memory on most microcomputer systems. (Also see BIT.)
Carriage return	Same as RETURN (see Return).
Clearing a bit	The process of setting a bit value to zero, regardless of its previous value.
COUT	Character OUTput subroutine in the TRACKSTAR monitor program.
CP/M	Control Program for Microcomputers. A popular operating system for 8080 and Z80 systems. Note:CP/M is a trademark of Digital Research, Inc.
CTRL	Abbreviation for control. Used mainly in conjunction with the control key on the IBM keyboard.
Cursor	A location on the screen (visible or invisible) which marks where the next character will be output or where the next input character will be printed after input.
Debugger	A special program or utility which aids in debugging a TRACKSTAR program.
Deferred execution	A term used to describe commands which aren't executed until a program is actually run.
Dimension	When used as a noun, dimension refers to an index of an array. Uses as a verb, this term describes the process of allocating storage for an array.

Directory	A file on a floppy diskette which contains a list of the files stored on the diskette.
Diskette	A floppy disk.
DOS	Acronym for Disk Operating System.
DOS 3.3	The name for Apple's standard disk operating system.
Drive	This term usally refers to the actual floppy disk drive. Sometimes drive is used to describe the disk drive number assigned by the operating system.
Eighty-column adapter	Since the Apple II normally supports a forty-column screen, a special hardware device is required to produce eighty-column video output. This device is called an eighty-column adapter. The TRACKSTAR board contains its own built-in eighty-column adapter.
ESC	An abbreviation for escape.
expr	An abbreviation meaning you should substitute an arithmetic expression.
File	A collection of related items stored on a floppy diskette.
Filename	The name of a file. Used to access the file through the directory.
Flag	A boolean value representing go/no-go, true/false, zero/one, data available/data not available, etc.
FLASHING	A video mode on the Apple and TRACKSTAR systems in which the affected characters switch between the inverse and normal display modes several times a second.
Function	A subroutine which returns a value to the caller.
Game controller	An analog input device (usually a joystick or a simple game paddle) which returns a value between 0 and 255 depending on the setting of a potentiometer.
Garbage collection	The process of freeing up unused memory when allocating storage for a string variable.
Half-tracks	Every floppy disk stores data on concentric rings called "tracks". An Apple compatible disk drive is capable of moving the disk controller head one-half track at a time. Certain copy

	protection schemes employ this ability of an Apple disk drive.
Hang	Whenever a computer begins execution of an infinite loop, or disappears without any trace of what it is doing, most computer programmers say that the system has "hung up". If your TRACKSTAR system refuses to respond to input from the keyboard and it doesn't appear to be accessing any other devices, you may want to exit the possible infinite loop by pressing the F1, ESC, R reset sequence.
Hexadecimal	Refers to the base 16 numbering system.
High-order	Whenever items in a list (for example, bits in a byte) are numbered, the term high-order usually refers to the highest-numbered element in the list. For example, bit seven in a byte is the high-order bit.
HIMEM	An abbreviation for HIgh MEMory. Usually refers a value in BASIC which points at the highest memory location available to BASIC.
HIRES	Short for HIgh RESolution.
Immediate execution	Commands that can be executed immediately after they are typed into the language system.
Initialization	The act of preparing a floppy disk so that it can be read or written by an floppy disk drive.
Input hook	A pointer in the TRACKSTAR address space which contains the address of the current input device handler subroutine.
Input port	A memory location from which data passed from the outside world can be read by the TRACKSTAR's 6502 microprocessor.
INVERSE	A mode of video output in which black letters are output on a white background.
Inverting a bit	If a bit contains zero, the inverted value is one; if a bit contains one, the inverted value is zero.
Keyboard strobe port	A memory location which, when accessed, clears the data available bit in the keyboard port.
lineno	A term which means "substitute a valid line number here".
LOMEM	Stands for LOw MEMory. A location containing the lowest memory location available for BASIC variable storage.

Loop	A sequence of commands which are repeatedly executed.
Loop control variable	A variable which controls the repeated execution of a loop. Usually a loop control variable is incremented or decremented once each time through the loop.
LORES	Abbreviation for LOw RESolution.
Low-order	Whenever items in a list are assigned sequential values, the lowest numbered item is called the low-order item. For example, bit number zero in a byte is called the low-order bit.
Machine language	A set of binary instructions that a microcomputer CPU chip understands.
Monitor	A program in the $F800..$FFFF address space of the TRACKSTAR system which performs various I/O and debugging functions.
Output hook	A two-byte pointer in the TRACKSTAR address space (at address $36) which contains a pointer to the currently active output peripheral driver code.
PC-DOS	The disk operating system commonly employed on the IBM PC.
Prompt	A string which reminds you to input data into a program.
Random access	When applied to a disk file structure, the term random access means that the program can randomly access data records within the file without worrying about the current position within the file.
Reset sequence	The sequence of keys which must be pressed in order to cause a hardware reset. On the TRACKSTAR system, the reset sequence always begins with the two-key sequence F1, ESC.
Return	This is the RETURN (or ENTER) key on your IBM PC keyboard. It is usually used to terminate a line of input.
Sequential access	A sequential access file can only be accessed sequentially. That is, the program can only read the record which immediately follows the record last read.
Setting a bit	The operation of forcing a bit to one regardless of its previous value.

sexpr	An abbreviation for any general expression which returns a string result.
Slot	The Apple II and IBM PC systems provide plug-in slots into which peripheral cards may be plugged. A slot refers to one of these connectors on the IBM PC or Apple motherboard. Note that the TRACKSTAR does not support any Apple-compatible slots.
Soft switch	A pair of memory locations. Accessing one of the locations turns some peripheral port on, accessing the other location turns the device off.
String	An array of characters with a length attribute.
String literal	A collection of characters enclosed by quotation marks.
Subroutine	A section of code which is called from another point in the program and returns to the instruction immediately past the call to subroutine once the subroutine has completed execution.
Subscript	A numeric expression used to access a particular element of an array. Exactly one subscript is required for each dimension of an array.
Substring	A string extracted from a larger string.
Toggle	Synonym for invert.
Turnkey	A system which, when first booted, automatically begins executing an application program is called a turnkey system.
Variable	Memory used to hold a varying value maintained by the programmer.
Virtual slot	Although the TRACKSTAR doesn't support physical slots, the TRACKSTAR system uses special software to emulate several popular peripheral devices using the peripherals plugged into your IBM PC. Such slots are called "virtual slots" since they do not physically exist, but the software cannot tell that the actual hardware isn't present.

Item	Page
40-column modes	25
80-column display	21
80-column modes	25
Apple cassette port	27
Apple compatible disk drive	22
Apple CP/M	21
Apple disk drive assignments	29
Apple disk drive booting sequence	29
Apple editing keys	22
Apple Key	22
Applesoft requires upper case	22
Backspace	34
Bell character	34
Booting the TRACKSTAR system	29
Changing the contents of memory	39
Clearing the screen	37
Clearing the text to the end of the current line	37
Clearing the text to the end of the screen	37
Closed Apple Key	22
Comparing two ranges of bytes	40
COUT subroutine	33
Creating your own monitor commands	42
CTRL-C	34
CTRL-E	41
CTRL-K	42
CTRL-Open Apple-Reset	30
CTRL-P	41
CTRL-Reset	30
CTRL-S	34
CTRL-Y	42
Cursor control keys	23
Cursor keypad	22
Defining a function key	31
Defining the size of the display	34
Diassembling 6502 instructions	41
Display switches	25
Displaying function key definitions	31
DOS 3.3	21
Dumping a range of bytes	39
Erasing the screen	37
Erasing the text to the end of the current line	37
Erasing the text to the end of the screen	37
ESC @ command	37
ESC A command	37
ESC B command	37
ESC C command	37
ESC D command	37
ESC E command	37
ESC F command	37
ESC I command	37
ESC J command	37
ESC K command	37
ESC M command	37
Escape editing characters	37

Item	Page
Examining and changing registers	41
Examining memory locations	39
Executing a 6502 machine language program	40
Exiting function key definition mode	31
FLASHING text	35
Floppy disk usage during boot operation	29
Function keys	31
Function keys	22
GETLN subroutine	36
Half-tracks	21
Hexadecimal arithmetic	42
HIRES colors	26
HIRES graphics bit layout	26
HIRES graphics modes	26
HIRES graphics screen locations	24
I/O Scratchpad locations	24
Installation	2
INVERSE	41
INVERSE text	35
Keyboard port	23
Keyboard strobe port	23
Keyboard usage	22
Line feed	34
Logical screen	34
LORES colors	26
LORES dot layout	26
LORES graphics modes	25
LORES graphics screen locations	24
LORES screen memory usage	25
Monitor "+" command	42
Monitor "-" command	42
Monitor "." command	39
Monitor ":" command	39
Monitor "G" command	40
Monitor "I" command	41
Monitor "L" command	41
Monitor "M" command	40
Monitor "N" command	41
Monitor "V" command	40
Monitor cassette commands	42
Monitor CTRL-E command	41
Monitor CTRL-K command	42
Monitor CTRL-P command	41
Monitor CTRL-Y command	42
Monitor period (".") operator	39
Monitor program	39
Moving a range of memory	40
Moving bytes around in memory	40
Moving the cursor back one location on the screen	37
Moving the cursor down on the screen	37
Moving the cursor forward one location on the screen	37
Moving the cursor up on the screen	37
Normal text	41
Normal text	35

Item	Page
Numeric keypad	22
Open Apple Key	22
Output cursor	33
Physical screen	34
RDKEY subroutine	36
Re-booting an Apple floppy diskette	30
Reading a character from the keyboard	36
Reading a line of text from the keyboard	36
Reading the keyboard port	23
Redefining a function key	31
Redefining the size of the display	34
Reset menu	30
Reset sequence	29
Reset sequence	22
Reseting the TRACKSTAR	30
Returning to Apple mode after entering IBM mode	30
Returning to Apple mode after reset sequence	30
Returning to IBM mode after reset sequence	30
Running canned programs	29
Screen editing on the TRACKSTAR	37
Scrolling operation on the TRACKSTAR display	33
Speaker port	26
Special monitor memory locations	42
Standard input	36
Standard output	33
Stop-list feature	34
Stopping a scrolling display	34
Storing data into memory	39
Switching between video modes	24
Text modes	25
Text screen locations	24
Text window	34
Toggling the speaker	26
Turning on an input device from the monitor	42
Turning on an output device from the monitor	41
Upper case	22
Video display	24
Video modes	24
Window Variables	35

Trackstar™ DOS
Reference Manual

© 1984 by Diamond Computer

© 1984 by Diamond Computer

No part of this manual may be reproduced in any form without the express written consent of Diamond Computer.

TRACKSTAR™ DOS Reference Manual

Table of Contents

Chapter One: Getting Started With TRACKSTAR™ DOS

Getting STarted With TRACKSTAR™ DOS..... 2

Chapter Two: Program and File Commands

Initializing a New Diskette..... 5
RENAME, SAVE, CATALOG, and LOAD Commands..... 6
The Reset Sequence..... 8
Slot Restrictions..... 9
TRACKSTAR™ DOS Syntax and Options..... 9
FP, INT, and the Monitor..... 10
Executing DOS Commands Within Programs 11
Immediate and Deferred Modes..... 11

Chapter Three: Housekeeping Commands

The Protection Game..... 13
The Debugging Commands..... 15
MAXFILES..... 15
Debugging With TRACE..... 16

Chapter Four: Sequential Files

Text Files..... 18
More on the GET Command..... 21
OPEN, CLOSE, WRITE, and READ..... 21
APPEND and POSITION..... 22
Using Byte Parameters..... 25
The EXEC Command..... 25
Some Facts About EXEC..... 28
Selective Copy Using TEXT Files..... 28
Language Conversion and EXEC..... 29

Chapter Five: Random Acess Files

Random Access Text Files..... 32
Using Random Access Files..... 32

Chapter Six: Machine Language and TRACKSTAR™ DOS

TRACKSTAR™ DOS and Machine Language Files..... 36

Chapter Seven: Input and Output

TRACKSTAR™ Input and Output..... 39

Chapter One: Getting Started With TRACKSTAR DOS

Getting Started With TRACKSTAR DOS

TRACKSTAR DOS is a program that emulates Apple DOS on an IBM system. Like Apple DOS, TRACKSTAR DOS presupposes that you have a working knowledge of BASIC commands and programming since many DOS commands are extensions of BASIC commands. The TRACKSTAR DOS BASIC manual provides a good introduction to Applesoft programming for the IBM user.

TRACKSTAR DOS is quite different from IBM-PC DOS. You will not be able to run IBM-PC files under TRACKSTAR DOS, nor will you be able to run TRACKSTAR DOS files under PC DOS. In addition, TRACKSTAR DOS is somewhat "primitive" compared to PC DOS, particularly concerning the DOS/BASIC interface. TRACKSTAR BASIC, like Applesoft, does not contain built-in DOS commands. The DOS commands, though many are extensions of BASIC commands, are a set unto themselves. They do not always perform the same functions or obey the same rules that BASIC commands do. Therefore, the best way to acquaint yourself with TRACKSTAR DOS is with hands-on training. Try out each of the sample programs provided within the text to be sure that you completely understand each section.

Within the sample programs and instructions that are given in this manual, we will use standardized symbols to represent certain functions. For example, <RETURN> will be used to designate that the return key should be pushed.

Booting up TRACKSTAR DOS

The first step for learning how to use TRACKSTAR DOS is to learn how to boot it up in your IBM system. Booting TRACKSTAR DOS follows four simple steps:

1. Boot PC DOS

2. Execute TRACKSTAR program by typing "TRAKSTAR" after the A> prompt.

3. Select OPTION #1, "DOWN-load Apple and Apple CP/M System file"

4. When so prompted, insert a DOS 3.3 compatible diskette into Drive A (or optional Apple compatible drive).

From this point on in the manual when you need to start up TRACKSTAR DOS, we will refer to this entire booting sequence simply as "booting TRACKSTAR DOS." It will be assumed that you will first begin the procedure by booting IBM-PC DOS.

Now you are ready to start experimenting with TRACKSTAR DOS. At this moment DOS may not appear any different to you than TRACKSTAR BASIC. There are, however, some very subtle changes that may not become apparent to you until we progress a little farther. The most drastic change that has taken place is that the DOS has reset the HIMEM pointer to the highest memory location that you can use. Of course, most BASIC commands will still work, but some new commands have been added in addition to some old command features being enhanced.

Chapter Two: Program and File Commands

Initializing a New Diskette

RENAME, SAVE, CATALOG, and LOADing Files With TRACKSTAR DOS

The Reset Sequence

Slot Restrictions

TRACKSTAR DOS Syntax and Options

FP, INT, and the Monitor

Executing DOS Commands Within Programs

Immediate and Deferred Modes

Chapter Two:Getting Started

Initializing A New Diskette

Initializing slave diskettes is a very simple procedure with TRACKSTAR DOS. First, place the TRACKSTAR DOS diskette into your computer drive and boot it up. Then, remove the TRACKSTAR DOS disk and place the new, blank disk in the drive. Type in the word "NEW" and push return.

At this point, the initialization routine in TRACKSTAR DOS requires you to write a brief "greeting" program. The greeting program should contain any pertinent information that will help you identify what is stored on the disk (i.e., your name, the date you initialized your disk, and what type of system your disk was initialized on). A sample initialization routine would be as follows:

```
10 REM  HELLO PROGRAM
20 PRINT "DISKETTE INFORMATION"
30 PRINT "YOUR NAME AND DATE"
40 END
```

It's a good idea to RUN the short program to be certain that it works as it should. Once your program is in working order, type the following instruction:

```
INIT HELLO
```

As soon as you press <RETURN>, the computer will finish initializing your disk. Don't be alarmed if the disk spins for what seems rather a long time. This process may take up to two minutes before completion.

It wasn't mandatory to type "HELLO" after the INIT command, you could have named this greeting program anything you chose. However, since "HELLO" is the standard name given to this program, it helps to avoid confusion by using the conventional name.

When your computer is finished INITializing the disk, the prompt character will be returned to the screen. The initialized disk is now a proper "slave" diskette.

RENAME, SAVE, CATALOG, and LOAD Files With TRACKSTAR DOS

Now that you have an initialized disk we can begin to write programs with TRACKSTAR DOS. To understand how TRACKSTAR DOS names, loads, and saves files, let's practice on a sample file.

Boot your initialized diskette and enter the command:

```
NEW
```

This will clear any programs in memory. Next, type in the following program:

```
10 PRINT " THIS IS A SAMPLE PROGRAM"

20 END
```

You should run the program a few times to make sure it's in working order. Now we will save the program by entering the command:

```
SAVE SAMPLE PROGRAM
```

At this point let's check to make certain that the program is really saved to disk. Type

```
NEW
```

and then

```
LIST
```

to show that the program is no longer in memory. Enter the command:

```
RUN
```

and your program should appear on the monitor screen. Big deal, so far everything has been pretty much the same as PC DOS? Well, let's take a closer look.

Saving programs with TRACKSTAR DOS is a little different from IBM-PC DOS. TRACKSTAR DOS allows up to thirty legal characters in a file name. Actually, you can use more than thirty characters if you want, but TRACKSTAR DOS will only pay attention to the first thirty. A legal character is a letter, number, symbol, or a space, with the exception of a comma(,) or the return key. Since any symbolic key is accepted, there is nothing to prevent you from using a control character in a file name. Be careful with this! The control character will not be printed to the screen, but it will be imbedded in the filename nevertheless. If you forget what control character you typed, you can find yourself temporarily unable to access your own file. This usage of control characters in filenames may seem like a sneaky way to keep people out of your files, but don't get too smug. Any programmer worth his salt will know how to list

the files to show hidden control characters.

There are a few qualifications you should be aware of when naming files. Although TRACKSTAR DOS will accept any legal character in a file name, you must begin the name with a letter. If you begin a file name with any thing else, a number for example, you will get an illegal file name message. The same thing will happen if you include a comma in the name.

Suppose you are in the middle of writing a program and for some reason you can't complete it at that time. You would simply save that portion and finish it later, right? To gain re-entry to a saved file or to tell the computer which program on a disk you want to run, you use the LOAD command. Enter the command:

LOAD SAMPLE PROGRAM

and SAMPLE PROGRAM will again be put into memory, ready for you to RUN it or to do some editing.

Once you have saved some files to disk, you can see exactly what programs are stored on that disk with the command:

CATALOG

When you enter CATALOG, all of the programs on that disk are displayed on the monitor. Each program's name is preceded by a letter and by a set of numbers. The letter indicates what language the programs are in. For example, the letter "I" indicates the program is in Integer BASIC. The letter "A" stands for an Applesoft (TRACKSTAR BASIC) program. The numbers that precede the file name show the length, in sectors, of the stored program. A diskette can store a possible 403 sectors, with each sector holding up to 256 bytes of information. However, whenever a file encompasses more than 255 sectors, the record length begins again at 000. If there are more programs on your disk than can be displayed at the same time on the monitor screen, the CATALOG command will list the first eighteen programs. To view the rest of the programs listed on the disk, you simply press any key but the control, reset, or shift keys.

Sometimes you may wish to rename a file. Changing the name of a file is a very simple procedure. For example, let's say you don't like the name of our last program, SAMPLE PROGRAM, and wish to call it NEW NAME instead. Simply type:

RENAME SAMPLE PROGRAM, NEW NAME

and the disk will spin in the drive for a while. When the prompt returns, enter the command

CATALOG

and you will see the old file name has been substituted with the new name.

If you wish to remove a file from your disk, it can be done with the DELETE command. Suppose you decide you no longer need the NEW NAME program. Load it in memory with the command:

LOAD NEW NAME

Now enter the command:

DELETE NEW NAME

and catalog your disk. NEW NAME should no longer appear on the listing. However, when NEW NAME was loaded it was put into memory, so you can still retrieve the program at this point with the command:

SAVE NEW NAME

Catalog the disk again, and NEW NAME should once again be included.

The RESET Sequence

The RESET sequence consists of one of the three following command sequences.

1. F1,ESC,R

2. F1,ESC,B

3. F1,ESC,I

Use the RESET sequence with extreme caution. If you accidentally use a RESET sequence while you're in TRACKSTAR DOS and have gotten the monitor prompt, you may still be able to return to DOS at the point where you left if you type in

3D0G

If you were not yet in DOS at the point where you entered the RESET sequence then you may be able to recover your program by using CTRL-C. However, if you used a RESET sequence while the disk drives were going, you may have lost the information for good (or bad). It is possible to ruin a portion of your diskette this way. In that case, you may not be able to boot your disk. It's sometimes possible to save the files on the bad disk by first booting with another diskette. You then load programs from the damaged disk and save them onto the good disk. You can always re-initialize your disk to use again, but this means you loose all the files on that disk.

Slot Restrictions

TRACKSTAR DOS only supports a controller card in slot 6. Therefore, only two disk drives can be used (drives one and two). Unless you specify otherwise, TRACKSTAR DOS will use drive one as the default drive (meaning that all commands will be sent to the disk in drive one. To specify that drive two be engaged instead, you must type a command followed by a comma followed by the notation D2. For example, the command

```
LOAD PEACHES, D2
```

would load the file PEACHES from the disk in drive two into memory. After you have specified a particular drive, all further commands will refer to drive two unless you respecify drive one. Note: TRACKSTAR DOS must be booted from drive one, never drive two.

A volume number can be assigned to a diskette at initialization. Volume numbers are safeguards to prevent diskettes from being accidentally written over. A volume number must be an integer from 1 through 254. If you do not specify a volume number, any volume is accepted. If you wish to give your disk a volume number at initialization, you must use a command sequence like the following:

```
INIT HELLO, D2, V214
```

This sequence initializes the diskette in drive two with the greeting program HELLO and also gives the disk the volume number 214. The order that the drive command and volume command appear is interchangeable. If you do not specify a volume number after the "V" when you initialize the disk, the default value of 254 is assigned to the disk. If you want to find out the volume number of a diskette, it is displayed at the top of a catalog listing of that diskette.

You can type the volume number after any DOS command if you wish to check that the disk volume number and the volume option agree. If the numbers do not agree, you will receive a VOLUME MISMATCH error. If you don't specify a volume number after a DOS command, or if you type "V" by itself or followed by a zero, then the volume number of the disk will be ignored by DOS.

TRACKSTAR DOS Syntax Commands

Since we have just gone over the drive, slot, and volume options available to you with TRACKSTAR DOS, this would be a good time to review the syntax or structure of the various DOS commands. Because TRACKSTAR DOS will only support a controller in slot 6, you need not worry about specifying a slot number. You can, however, direct the commands to drives one or two, and you can also make use of volume numbers. Also, TRACKSTAR DOS will allow any numerical constant in a command to be expressed in hexidecimal notation (using a dollar sign in front of the number).

Some example of commands available to you are listed below. Optional portions of commands are enclosed in parentheses and follow the main command in any order. Lower case letter "d" stands for the drive number, and lower case letter "v" stands for the volume number. Sample file names follow each command.

```
INIT GREETING (,Dd) (,Vv)

LOAD MY FILE.4 (,Vv) (,Dd)

SAVE BANK ACCOUNT $$$ (,Vv) (,Dd)

CATALOG (,Dd)

DELETE BIOLOGY PART 1 (,Dd) (,Vv)
```

FP, INT, and the Monitor

Under TRACKSTAR DOS you can easily move from one language to another. For example if you are using TRACKSTAR BASIC and you want to use Integer BASIC, you simply enter the command:

```
INT
```

and the prompt will change to show you that you are now indeed in Integer BASIC (>). Conversely, if you are in Integer BASIC and want TRACKSTAR BASIC, you simply type:

```
FP
```

(for floating point) and TRACKSTAR BASIC is now running.

It is just as easy to enter the monitor. From either TRACKSTAR BASIC or Integer BASIC, just enter

```
CALL-151
```

and you will get the moniter prompt (*). Once in the monitor, entering

```
3D0G
```

will return you to whatever language you were in before the call to the monitor.

Be careful moving in and out of languages and the monitor since you will lose any program in memory whenever you change.

Executing DOS Commands within Programs

You may sometimes wish put DOS commands in your BASIC programs. To accomplish this with TRACKSTAR DOS, you must PRINT a string using CTRL-D followed by a command. A sample program will help explain this idea.

```
10 PRINT CHR$(4); "CATALOG"
20 END
30 REM CHR$(4) IS CTRL-D
```

A control character in a line will be erased if you use the right arrow key to copy over the line.

Immediate and Deferred Modes

Most DOS commands can be used in both immediate and deferred modes. An immediate mode command is one that is executed at the time it is issued. DOS commands that are used within programs are called deferred commands since they don't do anything until the program is RUN. Some DOS commands are exclusively deferred commands:

```
OPEN
READ
WRITE
APPEND
POSITION
```

These commands are only used within a BASIC program following CTRL-D in a PRINT statement. The DOS commands

```
MAXFILES
INT
FP
```

should be used with extreme care in the deferred mode. In fact, it is wise to avoid using MAXFILES in the deferred mode. Note: MAXFILES should especially not be used from integer BASIC; it can wipe out strings in APPLESOFT. We will cover MAXFILES more thoroughly a bit later.

Chapter Three: Housekeeping Commands

The Protection Game

The Debugging Commands

MAXFILES

Debugging With TRACE

Chapter Three: Housekeeping Commands

The Protection Game

There are several ways you can protect your disk from the various mishaps that can occur to the unsuspecting diskette. We have already discussed volume numbers. They are one way to make certain that information does not put on the wrong disk. Other methods of protection include the LOCK function, the "turnkey" system, the VERIFY command, and, of course, copying the disk.

The LOCK command is safeguard against accidental erasure or deletion of a file. Once a file has been LOCKed, it can not be SAVEd, RENAMEd, or DELETEd until it is unlocked. The command

LOCK BANKING

locks the file named BANKING and prevents anything writing over it or deleting it. When the disk is cataloged, files that are locked are proceeded by an asterisk. To unlock a disk so it may amended or deleted, use the obvious command

UNLOCK BANKING

Locking a file will not prevent it from being LOADed or RUN. However, if you issue a command that tampers with the current contents of the file (such as SAVE or DELETE) you will receive a FILE LOCKED message. Of course, you can still SAVE a locked file once you've loaded it if you follow the SAVE command with a different file name. Locking a file will not prevent access to that file. It just safeguards the accidental wiping out of that file's information.

Another way to save your program from accidental tampering is the "turnkey" system. A turnkey system is a device that allows the computer to perform one certain function. For example, a secretary may need an accounting program. While the secretary may not know anything about computers, all she need do to use the accounting program is find the right diskette and turn on the computer. Since the program will interact with her in English, she need not learn any computer languages or know how the internal structuring of the program works.

This sample "turnkey" program executes everytime the diskette is booted.

1) Enter the program:

```
10 HOME, PRINT "MY DISK"
20 PRINT CHR$(4); "CATALOG"
30 END
```

2) Insert a BLANK diskette into the active disk drive.

3) Issue the DOS command:

```
INIT HELLO,V1
```

Whenever you boot this floppy disk, it will automatically execute the HELLO program.

The VERIFY command will check your diskette for self-contistency. When you create a file, DOS stores a checksum byte with the output buffer's contents into a disk sector. The VERIFY command will compare a new checksum byte against those stored in each file sector. If the two do not compare, then a I/O ERROR message is given. If the two do verify, no message is returned.

The VERIFY command is useful because information may occasionally not record correctly on a disk. This can happen if the disk has been mishandled or is faulty. VERIFY can be used on any type of file or program and can be called from the Monitor, Integer BASIC, or Applesoft. The VERIFY command follows the standard DOS command format. The command:

VERIFY BANKING,D2,V101

will VERIFY the file named BANKING with the volume number 101 in drive two.

As useful as VERIFY is, it can not tell you if the program contents have been messed up. If you SAVEd a file that was clobbered, it will still VERIFY if the disk sectors compare properly.

One of the best ways to protect your valuable files and programs from disaster is simply by making copies of your disk, especially during the creation of your files. In this way, should tragedy strike one disk (it is inadvertently placed in the microwave, Rover buries it, etc.), you can triumphantly whip out your back ups and laugh in the face of adversity.

Another wise idea is to SAVE your program every so often as you are developing it. If you SAVE your file every twenty minutes, you won't lose so much work should there be a power failure, natural or otherwise.

Finally, write-protecting your disk will guarantee that your programs will not be written over. By covering the notched cutout on the disk with tape or the stickers that are provided for this purpose with the disks, no one will be able to SAVE anything onto your disk without first removing the tape. Whenever an attempt is made to SAVE something on a write-protected disk, the message

WRITE PROTECTED

will be displayed.

The Debugging Commands

The MONitor command aids you in debugging a program by allowing you to see all the information exchanged between the disk and the computer that is normally not displayed on the monitor screen.

The command MON enables you to see various types of information. There are three parameters that work with the MON command:

C -Commands to the disk.
I -Input from the disk.
O -Output to the disk.

These three parameters are used in various combinations with the MON command in order to monitor various parts of the display. At least one of these parameters must follow the MON or NOMON commands or they will be ignored.

Used in conjunction with the MON command is the NOMON command. NOMON stands for NO MONitor and is used to turn off the parts of the display that MON has turned on.

The list below shows the seven different MON command forms and what the monitor:

1. MON C -Commands to the disk.
2. MON I -Input from the disk.
3. MON O -Output to the disk.
4. MON C,I -Commands to and input from the disk.
5. MON C,O -Commands to and output to the disk.
6. MON I,O -Input from and output to the disk.
7. MON C,I,O -Commands to, input from, and output to the disk.

To use NOMON to turn of a MONitor command, follow the same command sequence of separating the parameters with commas. As you become more proficient with the MONitor debugging commands, you will probably want to know how to use a MON command and cancel it without it showing on the screen. You can use the trick below to accomplish this. If you had used the command MON C,I,O you can cancel it with the following:

```
PRINT D$; "NOMON C,I,O": VTAB PEEK(37): CALL-868
```

MAXFILES

When you boot TRACKSTAR DOS, a command MAXFILES 3 is executed. This means that up to three files can be used at one time. TRACKSTAR DOS allows you to make up to a total of sixteen active files simultaneously. Unless it is ordered otherwise, the DOS system will automatically default to three active files when booted.

The MAXFILES command is what is used to adjust the number of active files. MAXFILE will reserve 595 bytes in memory for every file you call. This reserved space is called a file buffer. Since disk access speed is slower than memory speed, the extra space in the file buffer helps to regulate this difference.

When you access information on a disk, the DOS program will load 256 characters into a portion of the file buffer, known as the "input" section. DOS then delivers the portion of those 256 characters that contains the information you requested. However, if you are putting information onto a disk, then a reverse process occurs. DOS stores the characters you type into a portion of the file buffer known as the "output" section. After 256 characters have accumulated, they are sent to the diskette in a entire group.

Debugging With TRACE

The TRACE command in Applesoft is helpful to find bugs within a program but it should be used with care when in DOS. DOS commands in an Applesoft program will not work when TRACE is used because TRACE prints a line number with no RETURN before the DOS command. If you insert the following qualification for the D$ string, you can remedy most of the problems caused by TRACE:

```
10 D$ = CHR$(13) + CHR$(4)
20 PRINT D$; "CATALOG"
```

Chapter Four:Sequential Files

Text Files

Note on the GET Command

OPEN and CLOSE, WRITE and READ

APPEND and POSITION

Using Byte Parameters

The EXEC Command

Some Facts About EXEC

Selective Copy Using Text Files

Language Conversion and EXEC

Chapter Four: Sequential Files

Text Files

A text file, also referred to as a data file, is not a true program, but rather a means of storing information on a disk so that it can later be accessed by a program. There are two types of text files: sequential and random-access. On a disk's catalog directory, a text file is headed by the letter T. Text files are very useful for storing lists of information. For example, a program that keeps track of a store's inventory would probably use text files to hold information about various types of merchandise.

DOS commands are used to create and retrieve text files, although the DOS commands LOAD and RUN cannot be used in conjunction with a text file since these two commands expect a BASIC program file. You can, however, write a program in one language that creates a text file and then retrieve that text file from a program written in a different language. The DOS commands that are used with text files are as follows:

OPEN	LOCK
CLOSE	UNLOCK
READ	DELETE
WRITE	MON
APPEND	NOMON
POSITION	RENAME
EXEC	VERIFY
	CATALOG

The DOS commands in the first column are used from within a program to place and retrieve data in a text file. With the execption of the commands EXEC and CLOSE, the commands in the first column can be used only in the deferred execution mode (i.e., called from within a program). The DOS commands in the second column are used in the same way as they are when used with program files.

The sequential text file stores information just as its name suggests: in a linear sequence. The sequential text file is rather like a long tube into which groups of information can be stored. Each group of information is separated from the next by a <RETURN>. Each of these groups of information is called a field. All of the fields are stored in a long, continuous line. The characters for the information in each field is stored in their ASCII representation.

The easiest way to understand the idea of a sequential file is to develop one and then access it yourself. The following examples should prove helpful.

Remember our illustration of a text file being useful to keep an inventory of a store? The following program will create a text file for such inventory information to be stored on.

```
10 REM INVENTORY TEXT FILE
20 D$ = CHR$(4): REM CHR$(4) = CTRL-D
30 PRINT D$; "OPEN INVENTORY"
40 PRINT D$; "WRITE INVENTORY"
50 PRINT "SHAMPOOS"
60 PRINT "RAZORS"
70 PRINT "TOOTHBRUSHES"
80 PRINT "COMBS"
90 PRINT "MIRRORS"
100 PRINT "SOAPS"
110 PRINT "FLASHLIGHTS"
120 PRINT "BATTERIES"
130 PRINT D$;"CLOSE INVENTORY"
140 END
```

The "OPEN" command in line 30 opens the text file and adds the file "INVENTORY" to the catalog. The "WRITE" command in line 40 sends all the information in the following "PRINT" commands to the text file instead of to the monitor. The text file is shut with the "CLOSE" command in line 130. You can view the program if you issue the command MON C,O and then RUN the program. Each of the items in the INVENTORY program are now fields in the text file. The fields are of varying lengths depending upon the length of each name. There is a byte stored for each letter in a name and one more byte for the <RETURN> character. For example, the item "COMBS" takes up six bytes, five for the letters and one for the <RETURN> character that follows the item.

Now that we have created our INVENTORY text file, the next step is to write a program that will retrieve the information that is stored in it.

```
10 REM THIS PROGRAM RETRIEVES
20 REM THE INVENTORY INFORMATION
30 D$ = CHR$(4): REM  CTRL-D
40 PRINT D$; "OPEN INVENTORY"
50 PRINT D$; "READ INVENTORY"
60 FOR I = 1 TO 8
70 INPUT A$(I)
80 NEXT I
90 PRINT D$; "CLOSE INVENTORY"
100 END
```

In this program, line 40 OPENed our text file INVENTORY; line 50 READs our file. The READ command informs DOS that all following INPUT or GET commands are to refer to the information in the file INVENTORY. Thus, the INPUT command in line 70 causes one entire field (ending with the return character) in the INVENTORY file to be input. The loop in the program will consistently resubmit the INPUT command until all eight fields are input into the computer. You won't see anything on your monitor

screen unless you are using the MON C,I command. If so, you will see the following:

```
OPEN INVENTORY
READ INVENTORY
?SHAMPOOS
?RAZORS
?TOOTHBRUSHES
?COMBS
?MIRRORS
?SOAPS
?FLASHLIGHTS
?BATTERIES
CLOSE INVENTORY
```

The question mark (?) before each item is printed by the INPUT statement in the program.

It is possible to store more than one piece of information into a single field. A comma is used to separate several items in stored in one field. Look at the following program called INVENTORY2.

```
10 REM INVENTORY2 WORDS
20 REM STORED IN ONE FIELD
30 D$ = CHR$(4) : REM CTRL-D
40 PRINT D$; "OPEN INVENTORY2"
50 PRINT D$; "WRITE INVENTORY2"
60 PRINT "SHAMPOO, RAZORS, TOOTHBRUSHES,";
70 PRINT "COMBS, MIRRORS, SOAPS,";
80 PRINT "FLASHLIGHTS, BATTERIES"
90 PRINT D$; "CLOSE INVENTORY2"
100 END
```

All the items are now listed in one field. To retrieve the INVENTORY2 program, use the following:

```
10 REM THIS PROGRAM RETRIEVES
20 REM INVENTORY2
30 D$ = " " : REM " " MEANS CTRL-D
40 PRINT D$; "OPEN INVENTORY2"
50 PRINT D$; "READ INVENTORY2"
60 INPUT A1$,A2$,A3$,A4$,A5$,A6$,A7$,A8$
70 PRINT D$; "CLOSE INVENTORY2"
80 END
```

If you type MON C,I,O and then RUN the program above, you will see that all the words are listed one after another. Notice also that there is only one question mark (?) heading the list since only one field was INPUT. By following each word in the program with a comma, the INPUT statement with multiple variables was able to retrieve each separate word.

If you wish to replace or modify words in an existing program, you must DELETE the text file before re-RUNning the BASIC program. Otherwise, you may be left with parts of the old file still in existence. This can happen if what you add or change in the file does not completely write over the old stuff.

NOTE: if you are using Integer BASIC, you can separate several INPUT responses with commas, but you cannot use commas to separate string variables. <RETURN> characters are the only way to separate several responses to string variables used with INPUT. Therefore, in the above program, INVENTORY2, the entire field would have been assigned to the single variable A1$ (in Integer BASIC).

Note on the GET Command

The GET command is used to retrieve data from a text file, character by character. When the GET command is used to retrieve information from a text file, you may experience a few problems. If a DOS command is the first thing printed immediately after a GET command is issued, the DOS command may be ignored. This can happen because the <RETURN> that should precede the DOS command is missing. You can correct this by placing, in the same line, a <RETURN> character before the DOS command.

Another problem that may arise with GET is that when you use MON C,I,O the first character PRINTed after the GET will appear on the screen. Conversely, with NOMON C,I,O the first character PRINTed after the GET will not appear on the screen. You can use a simple trick to cure both of these problems. Place the non-printing character CTRL-A before the first desired PRINT character.

The programs we've created to READ our sequential text files have both read the correct number of items per field. However, if we had specified the wrong number of items to be READ in a field and then used MON C,I,O to view the process, we would have received a ?EXTRA IGNORED message. This would have occurred because of the inequality between variables specified and the actual INPUT responses in the fields.

OPEN and CLOSE, WRITE and READ

We have used the commands OPEN, CLOSE WRITE and READ in our example programs, but a closer examination of how they work could be helpful. As we have learned, the creation of a sequential text file basically involves the following commands:

OPEN
WRITE
PRINT
CLOSE

The following commands are used to retrieve a sequential text file:

OPEN
READ
INPUT
CLOSE

If you neglect to CLOSE a file that has been OPENed, you may lose data from that file.

When you use the DOS command OPEN, a 595-byte file buffer is set aside to store input and output. The OPEN command also readys the computer to READ or WRITE from the beginning of a file. After the CLOSE command is encountered, the file buffer that OPEN created is released.

The WRITE command allows you to enter data into a sequential file. The WRITE command cannot be used until a file is OPENed. After you specify the WRITE command, all PRINT commands that follow will be sent to the disk until any DOS command is used in a PRINT statement. You can also cancel the WRITE command by using INPUT, but the question mark character (?) that INPUT normally displays on the screen will be stored as the last text file character. If an error message is encountered before a DOS command or an INPUT statement, it will also cancel the WRITE command, but the error message will be stored as the last field in the text file.

If you WRITE to a previously existing file, you will overwrite the data in that file. As long as you enter the same length or more data into the file everything is fine. If, however, the new list you enter does not equal or exceed the old, the result will be a mixture of new and old data in the file. The simplest way to guarantee that you will not end up with old data still in the file is to DELETE the old file before you begin the new. This can be done from within your new program by first OPENing the file, and then DELETEing the file. Now you can OPEN the file once again and start afresh.

The READ command lets you retrieve a sequential text file. The file must first be OPENed before it can be READ. A DOS INPUT statement must follow READ. Like the WRITE command, the READ command is canceled when a DOS command follows a PRINT statement. PR# and IN# commands also cancel the READ statement.

APPEND and POSITION

There are two more DOS commands that are useful with sequential files: APPEND and POSITION. APPEND is used to add data to the end of text file. POSITION is used to access information from any specified field within a text file.

The OPEN command sets the position of the file pointer to byte 0 (which is the first character of the file). This is why you overwrite existing data when you OPEN and WRITE to an existing file. The APPEND command, however, allows you to set the position of the file pointer to one byte beyond the last character of the file. It is not necessary to first OPEN the file you wish to APPEND since APPEND is in itself a form of an opening.

To illustrate the APPEND command, let us first design a sample file.

```
10 REM SAMPLE FILE
20 D$ = CHR$(4): REM CHR$(4) = CTRL-D
30 PRINT D$;"DELETE SAMPLE"
40 REM 30 CLEARS ANY EXISTING FILE
50 PRINT D$; "OPEN SAMPLE"
60 PRINT D$; "WRITE SAMPLE"
70 PRINT "NUMBER 1"
80 PRINT "NUMBER 2"
90 PRINT D$; "CLOSE SAMPLE"
```

Now that we have established our SAMPLE file, let's create a program to APPEND it.

```
10 REM APPEND SAMPLE
20 D$ = CHR$(4) : REM CHR$(4)=CTRL-D
30 PRINT D$; "APPEND SAMPLE"
40 PRINT D$; "WRITE SAMPLE"
50 PRINT "NUMBER 3"
60 PRINT "NUMBER 4"
70 PRINT "NUMBER 5"
80 PRINT D$; "CLOSE SAMPLE"
```

This program will add the NUMBERs 3,4, and 5 to the end of the SAMPLE text file. You must always follow an APPEND command with a WRITE command.

To display the SAMPLE file we can use the following program.

```
10 REM THIS PROGRAM WILL RETRIEVE
20 REM THE SAMPLE FILE
30 D$ = CHR$(4) : REM CHR$(4)=CTRL-D
40 PRINT D$; "OPEN SAMPLE"
50 PRINT D$; "READ SAMPLE"
60 FOR I = 1 TO 5
70 INPUT A$
80 NEXT I
90 PRINT D$; "CLOSE SAMPLE"
```

APPEND allows you to add fields to the end of a file. If you want to WRITE or READ information at a particular point in a field, not just add on to the end, you must use the POSITION command.

The POSITION command moves the file pointer forward from its current position to however many fields you specify. To use the POSITION command, you must follow the command with the file name and then issue the Relative-field position (remember to separate them with a comma). The following is the syntax for the command:

POSITION f [,Rp]

The letter "f" stands for the filename, and Rp is the Relative-field position. You substitute for "p" the number of fields you wish to move the file pointer up from its current position. For example, when you OPEN a file the file pointer is automatically set to the beginning of the first field. So, if POSITION is used immediately after the OPEN command then the number you specify for "p" will correspond with the actual position of that field in the file. Otherwise you must figure out how many fields away is the field you wish to position the file pointer in. For example, after an OPEN command is given, the file pointer is set to the beginning of the first field (position zero). Therefore, if you wish to set it to point at the forth field, you issue the command:

POSITON FILE,R4

But suppose the file pointer is already set at the second field. If you specify ",R0" then the pointer will remain in the second field. In order to position the file pointer at the fourth field, you need to specify ",R2" and the pointer will be moved up two fields from its current position in the second field. If you specify a field that does not exist in your file (e.g., you specify the file pointer to move six fields up and your file only contains a total of four fields) then program execution stops and you will receive a END OF DATA message.

POSITION can be used only on an OPENed file. Also, POSITION is a DOS command so it will cancel a WRITE or READ command if it follows them. You must make sure to use the POSITION command before such commands.

The following program will illustrate how the POSITION command is used to retrieve certain fields from our SAMPLE program.

```
10 REM POSITION PROGRAM
20 REM FOR SAMPLE FILE
30 D$ = CHR$(4): REM CHR$(4)=CTRL-D
40 PRINT D$; "OPEN SAMPLE"
50 PRINT D$; "POSITION SAMPLE,R2"
60 PRINT D$; "READ SAMPLE"
70 INPUT A$
80 PRINT D$; "POSITION SAMPLE,R1"
90 PRINT D$; "READ SAMPLE"
100 INPUT B$
110 PRINT D$; "OPEN SAMPLE"
120 PRINT D$; "POSITION SAMPLE,R3"
130 PRINT D$; "READ SAMPLE"
140 INPUT C$
150 INPUT E$
160 PRINT D$; "CLOSE SAMPLE"
```

If you use MON C,I,O and RUN the above program, you will see the following:

```
OPEN SAMPLE
POSITION SAMPLE,R2
READ SAMPLE
?NUMBER 3
POSITION SAMPLE,R1
READ SAMPLE
?NUMBER 5
OPEN SAMPLE
POSITION SAMPLE,R3
READ SAMPLE
?NUMBER 4
?NUMBER 5
CLOSE SAMPLE
```

Remember that an OPEN automatically sets the file pointer to the first field in the file. Also, INPUT causes one field to be READ and then advances the file pointer to the beginning of the next field.

Using Byte Parameters

Remember our example text file of INVENTORY? We listed various items that a store might stock. However, such a list does not contain enough information to be helpful to a real inventory. We could have included a little more information about each item, such as how many bottles of shampoo are in stock and at what price. Then, any time we wanted to know how many bottles of shampoo we had on the shelf, we could find out. In fact, we could get this particular information without having to look at all the other information in that field, providing we knew a few things ahead of time. For example, if we knew what byte the information about the number of shampoo bottles was stored on, we could READ or WRITE in that exact spot.

If you arrange your field information in an exact oranization on the disk you will be able to use byte parameters to READ or WRITE on selected parts of the file or the fields themselves. In order to do this you must know exactly how many commas, spaces and other characters are in the file or fields. If you use the WRITE command and do not overwrite the exact amount of spaces as were in the original, strange things can happen. For example, if you do not print as many characters as were in that position before, the field will split off at the remaining old characters and create a second field. If you print more characters then were there originally, you will overwrite some characters in the beginning of the next field.

The byte (B) parameter is an absolute or actual position in the field. However, if R is specified, then the byte parameter is an actual position in a specified field. The syntax for using byte parameters with our example file SAMPLE is as follows:

```
WRITE SAMPLE, B13
```

The "B" prepares the computer to use the byte parameter. The number that

follows B refers to a position within the field. If no position is specified after the B, then the byte parameter uses the default of 0 (the first byte in the field).

Likewise, the syntax for using byte parameters with the READ command is as follows:

```
READ SAMPLE, B24
```

where the byte parameter is set to the 24th byte of the file.

The following program illustrates the usage of the byte parameter in our SAMPLE file.

```
10 REM BYTE PARAMETER WRITING
20 REM ON SAMPLE FILE
30 D$= CHR$(4) : REM CHR$(4)=CTRL-D
40 PRINT D$; "OPEN SAMPLE"
50 PRINT D$; "WRITE SAMPLE, B14"
60 PRINT "BYTE PARAMETER"
70 PRINT D$; "CLOSE SAMPLE"
```

Now, RUN our retrieve program using MON C,I,O. You should see that BYTE PARAMETER has overwritten the fields containing NUMBER 2 and NUMBER 3 and part of NUMBER 4. Only four fields now exist in the SAMPLE file, so when the fifth INPUT command is reached, the END OF DATA message should be displayed.

Use the following program to READ the new SAMPLE file.

```
10 REM BYTE PARAMETER READING
20 REM OF SAMPLE FILE
30 D$ = CHR$(4) : REM CHR$(4)=CTRL-D
40 PRINT D$; "OPEN SAMPLE"
50 PRINT D$; "READ SAMPLE"
60 INPUT A$
70 PRINT D$; "CLOSE SAMPLE"
```

The EXEC Command

Text files can be used to hold commands and programs as well as lists of information. To RUN these text files we use the DOS command EXEC.

An EXEC file is used to execute the commands or statements that are stored in a text file. When this is done it is as though the commands from the text file that are being executed have just been typed at the keyboard.

The EXEC command usually follows the syntax:

EXEC (filename)

The (filename) must be a sequential text file that contains program lines or BASIC commands. The first field of the file will be read into the computer as though it were being typed in from the keyboard. After a <RETURN> character is entered from the information in the field, the computer will try to execute what it assumes to be a BASIC command or a line in a program. When the first field has been read and acted upon, the second field is entered in the same fashion. This process continues until the entire file is read.

The following programs demonstrate how to create and use an EXEC file. We will have our EXEC file RUN a program, list some line numbers, and catalog the disk.

First, write and SAVE the following program:

```
10 REM DEMO
20 PRINT "THIS DEMO HAS BEEN RUN BY EXEC!"
```

Now we will create a program that will make a text file called SERVANT. When we use the EXEC command to run SERVANT, the SERVANT text file will list some line numbers, RUN the DEMO program, and then catalog the disk.

The following program, called CREATOR, will make the SERVANT text file.

```
10 REM CREATOR PROGRAM
20 D$ = CHR$(4): REM CHR$(4)=CTRL-D
30 PRINT D$; "OPEN SERVANT"
40 PRINT D$; "WRITE SERVANT"
50 PRINT "LIST 30,60"
60 PRINT "RUN DEMO"
70 PRINT "CATALOG"
80 PRINT D$; "CLOSE SERVANT"
```

You may be wondering why we didn't put D$ (CTRL-D) after the print commands in lines 50-70. These lines are entered into the SERVANT text file to be executed later by the EXEC command. Remember that EXEC expects a BASIC command or program within a text file so the D$ after a print statement is not needed.

SAVE the CREATOR program and then type RUN CREATOR to make the SERVANT text file. After this has been completed, issue the command:

EXEC SERVANT

and you should see lines 30-60 listed on the screen, and the DEMO program should be RUN. After that the disk's catalog should be displayed.

Some Facts About EXEC

1. Once you issue the EXEC command, you cannot stop it with CTRL-C. If an Applesoft program is RUNning under an EXEC command and you stop it with a CTRL-C, the rest of the EXEC file will probably not be executed properly.

2. The type of BASIC of a file being EXECed is not changed by the EXEC command unless an INT or FP ommand is encountered within the file itself.

3. If another EXEC command is encountered within the file you are EXECing, your initial EXEC command will be CLOSEd and the second EXEC command will be executed. Thus, it is important to realize that if there is an EXEC command within the file you are EXECing, the remainder of your original EXEC instructions will not be carried out. No two EXEC commands can be executed at the same time. Execution will be given to the most recent of the EXEC commands.

4. If a RUN command is encountered in a file being EXECed, the program referred to is RUN, and then attention is returned to the next EXEC file command. If, however, there is an INPUT statement in the program that is RUN under EXEC, the next field in the file will be used as the INPUT response. While the next field is being used as the INPUT response, if any immediate-execution DOS command is encountered it will be executed before the program continues.

5. EXEC can be used with MON C,I,O as a fast, primitive way to examine the contents of any text file. EXEC will return the message SYNTAX ERROR if a field in a text file does not contain a BASIC command or program line. Then EXEC will move on to the next line in the file. So, used with MON C,I,O, the EXEC command is a useful aid in debugging.

6. EXEC can start at any specified field in a file when it is used with a Relative-field position parameter. In such cases the Relative-field position is always figured from the beginning of the file. This is because the EXEC command automatically sets the file pointer to the first character in the the file. So, the parameter position must correspond to the file's absolute field. The syntax for this procedure is as follows:

EXEC f [,Rp]

Selective Copy Text Files

The EXEC command can be used to copy program listings into text files. This is very useful for taking commands from one program and placing them into another, even into that of an other program language. Using EXEC in such a fashion will allow you to insert subroutines into programs by EXECing a subroutine file, translate Integer BASIC programs into Applesoft, renumber and place parts of programs into other programs, and APPEND one program to another. You will also be able to repair programs with bad portions by copying the good portions into a text file, re-booting, and EXECing the program back into memory.

Use the following program to take selected line numbers from one program in memory and store them into a text file. In place of (LINE NUMBER),(LINE NUMBER) in line 60, you enter the line numbers you want to copy. In this example the text file we are storing our line numbers into is named SELECTION.

```
10 REM SELECTION
20 D$ = CHR$(4) : REM CHR$(4)=CTRL-D
30 PRINT D$; "OPEN SELECTION"
40 PRINT D$; "WRITE SELECTION"
50 POKE 33,30
60 LIST (LINE NUMBER),(LINE NUMBER)
70 PRINT D$; "CLOSE SELECTION"
80 TEXT : END
```

If you wish to add this program on to one of your own, just renumber the lines to fit within your program.

When you use this program to select and copy portions of program listings, they are stored in the text file with their line numbers intact. When the text file is EXECed, the lines will not be executed, but simply stored in the computer's memory.

Language Conversion and EXEC

The EXEC command can be used with the following program to convert a machine-language routine to a BASIC program portion which will POKE the machine-language routine into memory.

```
10 REM CODE-POKES CREATOR
20 D$ =CHR$(4): REM CTRL-D
40 PRINT D$; "DELETE CODE-POKES"
50 PRINT D$; "OPEN CODE-POKES"
60 PRINT D$; "WRITE CODE-POKES"
70 LINENUMBER = (?)
80 FOR PLACE = (?) TO (?)
90 COUNTER = COUNTER + 1
100 IF COUNTER = 10 THEN COUNTER = THEN COUNTER = 1
110 IF COUNTER <> 1 THEN 150
120 PRINT
130 PRINT LINENUMBER;
140 LINENUMBER = LINENUMBER + 1
150 PRINT " POKE " ;PLACE;",";  PEEK (PLACE);" :";
160 NEXT PLACE
170 PRINT
180 PRINT D$; "CLOSE CODE-POKES"
190 END
```

The line number of your BASIC program where the POKE program portion will start should be substituted for (?) in line 70. Likewise, the starting and ending decimal memory locations of the machine-language routine you are converting should replace the two (?) in the FOR loop in line 80.

RUN this program to create the text file CODE-POKES. When you issue the command EXEC CODE-POKES your machine-language routine will be placed into any other program at whatever line number you've specified.

Chapter Five:Random Access Files

Random-access Text Files

Using Random-access Files

Chapter Five:Random Access Files

Random-access Text Files

Remember how we compared the use of sequential text files to storing information in linear chunks within a tube? Well, random-access text files can be compared to storing information in a series of pigeonholes, called records.

Random access files are useful for applications where you need to gain access quickly to various parts of the file. It is easier to retrieve or store information in a random-access file than in a sequential text file.

However, unlike fields in a sequential file, each record in a random-access file must conform to a standard size within that file. This specified size is referred to as the file's fixed length. Equal space is reserved for all records in a file, even though some records may not need all that space. Thus random-access text files are not very memory efficient and should not be used for all applications.

Using Random-access Files

Since random-access files have fixed lengths, when you OPEN a file you must specify the length of the records. This is done using the following syntax:

```
OPEN filename,Lnnn
```

where nnn represents the length of all records.

You must also number each record since they are not accessed in a linear fashion. Whenever you issue a READ or WRITE command it should include the record number (R). Here are some examples:

```
WRITE filename,R1

READ filename,R23
```

In the following sample programs, we will create another inventory list, this time using random-access files.

```
10 REM MAKE INVENTORY
20 D$ = CHR$(4) : REM CHR$(4)=CTRL-D
30 INPUT "ITEM:      ";I$
40 INPUT "AMOUNT:    ";A$
50 INPUT "PRICE:     ";P$
60 PRINT D$; "OPEN INVENTORY,L150"
70 PRINT D$; "WRITE INVENTORY,R1"
```

```
80 PRINT I$ : PRINT A$ : PRINT P$
90 PRINT D$; "CLOSE INVENTORY"
```

When this program is RUN, it will ask for the various inventory information of the item, the amount in stock, and the price of the item. Do not enter any commas or colons with your responses. The program then sends this information to record 1 of the random-access file INVENTORY. The length of the records in this file (150 bytes) is specified in line 60 by L150. Line 70 says information will be sent to record 1 of the file and line 80 specifies that that information will be I$, A$, and P$ (item, amount, and price).

The next two programs will retrieve the information in record 1 of INVENTORY in slightly different ways.

```
10 REM FIRST RETRIEVE PROGRAM
20 REM FOR INVENTORY FILE
30 D$ = CHR$(4) : REM CHR$L(4)=CTRL-D
40 PRINT D$; "OPEN INVENTORY,L150"
50 PRINT D$; "READ INVENTORY,R1"
60 INPUT I1$,A1$,P1$
70 PRINT D$; "CLOSE INVENTORY"
```

```
10 REM SECOND RETRIEVE PROGRAM
20 REM FOR INVENTORY FILE
30 D$ = CHR$(4): REM CTRL-D
40 PRINT D$; "OPEN INVENTORY,L150"
50 PRINT D$; "READ INVENTORY,R1"
60 INPUT I1$
70 INPUT A1$
80 INPUT P1$
90 PRINT D$; "CLOSE INVENTORY"
100 END
```

This INVENTORY program only wrote one record of the file. All of the information (i.e., item, amount, and price) was sent to the first record.

We could have just as easily sent information to several records. The program below uses a FOR loop to write information to several records in the random-access file.

```
10 REM INVENTORY.2
20 REM WRITES TO SEVERAL RECORDS
30 D$ = CHR$(4) : REM CHR$(4)=CTRL-D
40 PRINT D$; "OPEN INVENTORY.2"
50 PRINT D$; "DELETE INVENTORY.2"
60 PRINT D$; "OPEN INVENTORY.2,L45"
70 FOR I = 10 TO 15
80 PRINT D$; "WRITE INVENTORY.2,R";I
90 PRINT "ITEM  PRICE ";I
100 NEXT I
110 PRINT D$; "CLOSE INVENTORY.2"
```

The fixed length for records in the file INVENTORY.2 is 45 bytes (line 60). The loop in lines 70-100 will send the information to records 10 through 15. Each time the program writes the information it will be followed by a record number.

To READ this text file you need to retrieve the information from records 10 through 15. The following program also uses a loop, this time to READ the records.

```
10 REM RETRIEVE INVENTORY.2
20 D$ = CHR$(4) : REM CHR$(4)=CTRL-D
30 PRINT D$; "OPEN INVENTORY.2,L45"
40 FOR J = 12 TO 15
50 PRINT D$; "READ INVENTORY.2,R";J
60 INPUT A$
70 NEXT J
80 PRINT D$; "CLOSE INVENTORY.2"
```

Once you've written a random-access file and specified the length of the records, it is extremely important to make a note of the record length of your file. If you forget the length, there really is no way to find out. Any time you wish to READ or WRITE to these files you will have to specify the length. If the record length you name is wrong, DOS will use this wrong length to calculate positions within the file. Therefore, be sure to keep track of the record lengths for all your random-access files.

It is also very important not to exceed the number of bytes you set for your record length. If the information you store in a record is too long for the length you have specified, you may overwrite another record or combine two records.

You can use byte parameters with READ and WRITE commands in random-access files. A byte parameter will specify the beginning byte of whatever record you specify. Using a byte parameter will move the file pointer forward or backwards from its current position. The default value of the byte parameter is 0. Use the following syntax for the byte parameters:

```
READ filename (,Rr) (,Bb)

WRITE filename (,Rr) (,Bb)
```

The POSITION command can also be used to move the file pointer ahead (not back) from within a record. Used in this way, POSITION cancels any READ or WRITE command that was issued before it. Therefore, you must re-issue a simple READ or WRITE without parameters.

Do not use CTRL-C to stop a READ. You can use the RESET sequence to accomplish this.

Chapter Six: Machine Language and TRACKSTAR DOS

TRACKSTAR DOS and Machine Language Files

The previous sections of this manual have all dealt with the computer's program memory and mainly with BASIC program commands. Binary files are used for the contents of any portion of the computer's memory that is in the uninterpreted machine language form. In this section we shall briefly discuss three DOS commands--BSAVE, BLOAD, and BRUN--that are used for binary files.

BSAVE will create a file and store the contents of a memory segment into it. With the BSAVE command it is necessary to specify not only the length parameter (L) but the starting address parameter (A) as well. Of course, you can also specify drive and volume parameters, but they are optional; the length and address are not.

The address parameter (A) specifies where the portion of memory to be stored begins. This address may be in decimal or hexidecimal code. If it is in hexidecimal code a dollar sign ($) must be placed in front of the address. The address parameter must be within the range of 0 through 65535 ($0..$FFFF) or a SYNTAX ERROR message will result.

The length parameter signifies how long a portion of memory is to be stored. You may not store fields larger than 32767 ($7FFF) bytes, without using two BSAVES. If you specify a length in the range of 32767 through 65535, a RANGE ERROR message is displayed. Also, a SYNTAX ERROR is displayed if you specify a length that is less than 0 or greater than 65535.

An example of the syntax of a BSAVE command is as follows:

BSAVE filename,Aa,Ll

where "a" and "l" represent the number of the address parameter and the number of the length parameter, respectively.

The DOS command BLOAD is used to load the contents of a binary file into memory. If there is a BASIC program already in your computer's memory, BLOAD usually doesn't erase it. However, if the binary file is BLOADed with a same address where the BASIC program is placed, you will lose the BASIC program.

You may specify an address parameter when BLOADing a file. If you do not, the file is placed at the same address that was specified when it was BSAVEd.

BRUN is used to run a machine-language program. The BRUN command may also be used with an optional address parameter. Like BLOAD, if no address parameter is mentioned than BRUN places the file at the same location that was BSAVEd.

The BRUN command BLOADs the specified file, then does a machine-language jump to whatever address was specified. At that point, if the file is a true machine-language program, BRUN then executes it.

Chapter Seven: Input and Output

Chapter Seven:TRACKSTAR Input and Output

The usual method of input is the keyboard, and the monitor screen is the standard form of output. However, you can redirect TRACKSTAR DOS by using input and output devices.

For example, when you type

```
IN# 6
```

the input will be obtained from the disk controller card in slot six (TRACKSTAR controller card).

Likewise, output can be redirected. The command

```
PR# 1
```

sends output to slot one which is usually the printer card.

```
PR# 0
```

will return the output to the monitor screen. The appropriate Appendices will contain more information about input and output options with TRACKSTAR DOS.

Input and output may be redirected from within a program. To do so from within your program, you must place the IN# and PR# statements in the form of DOS commands. For example,

```
PRINT D$; "IN# 0"

PRINT D$; "PR# 1"
```

where D$ stands for CTRL-D. Again, Appendices One and Two will contain more information on how to use IN# and PR# commands under TRACKSTAR DOS.

Item	Page
APPEND	22
APPEND	18
APPEND	11
Appending data to the end of a sequential text file	23
Backup copies	14
BLOAD	36
Booting TRACKSTAR DOS	2
BRUN	37
BSAVE	36
Byte option for random access files	34
Byte parameters for READ	26
Byte parameters for WRITE	25
CATALOG	18
CATALOG	7
CATALOG	6
Changing the name of a program saved on the diskette	7
Checking a file to see if it is damaged	14
CLOSE	21
CLOSE	18
Converting floating point BASIC to Integer BASIC	29
Converting Integer BASIC to floating point BASIC	29
Creating a text file listing of a program	28
CTRL-C	34
Debugging commands	15
Debugging with the Applesoft TRACE command	16
Deferred DOS commands	11
Deferred execution mode	18
DELETE	18
DELETE	8
Deleting a file on the diskette	8
Disk directory	7
Disk drive slot numbers	9
Diskette volume number	9
Drive number syntax	9
EXEC	26
EXEC	18
EXEC'ing a file	28
Executing DOS commands within a program	11
Fields in a sequential text file	20
File buffers	15
Filename format	6
Formatting a diskette	5
FP	11
FP	10
GET	21
HELLO program	5
HIMEM	3
Immediate DOS commands	11
IN#	39
Initializing a new diskette	5
INT	11
INT	10
LOAD	6
Loading a BASIC program	7

Item	Page
Loading binary object code from disk	36
LOCK	18
LOCK	13
Machine language files	36
MAXFILES	15
MAXFILES	11
Maximum number of open files	15
MON	20
MON	18
MON	15
Moving the file pointer forward in a sequential text file	24
NOMON	18
NOMON	15
OPEN	21
OPEN	18
OPEN	11
Opening a random access file	32
Peripheral slots	9
POSITION	22
POSITION	18
POSITION	11
POSITION and random access files	34
PR#	39
Protecting a file from accidental erasure	13
Random access text files	32
READ	21
READ	18
READ	11
Reading a random access file	32
Reading data from a sequential file	22
Reading from a particular record in a random access file	32
Reading multiple fields per record in a text file	20
Record length	32
Record parameter	32
Redirecting keyboard input from a file	26
RENAME	18
RENAME	7
RENAME	6
Reset sequence	8
Restarting DOS from the monitor	10
Retrieving information from a text file	19
Running a machine language program from disk	37
SAVE	6
Saving BASIC programs with TRACKSTAR DOS	6
Saving binary object code to disk	36
Sequential file storage format	18
Sequential text files	18
Slot number syntax	9
Slot restrictions	9
Slots	9
Syntax for drive, slot, and volume numbers	9
Text files	18
TRAKSTAR program	2
Turnkey systems	13

Item	Page
UNLOCK	18
UNLOCK	13
VERIFY	18
VERIFY	14
Viewing the disk directory	7
Volume number syntax	9
Volume numbers	9
WRITE	21
WRITE	18
WRITE	11
Write protecting a diskette	14
Writing data to a sequential file	22
Writing to a random access file	32
Writing to a sequential text file	19

Trackstar™ BASIC Reference Manual

© 1984 by Diamond Computer

© 1984 by Diamond Computer

No part of this manual may be reproduced in any form without the express written consent of Diamond Computer.

Table of Contents

Chapter One: Introduction

Immediate Execution Commands..... 2
Deferred Execution Commands..... 3
Editing Features..... 4
Number Format..... 5
Variable Names..... 6
Array Variables..... 6
The LET Statement..... 7
TRACKSTAR™ BASIC Line Format..... 10
The PRINT Statement..... 11
The INPUT Statement..... 12
The GET Command..... 13
READ, DATA, and RESTORE..... 13
IN#, PR#, and I/O Redirction..... 14
DEF FN and FN name..... 15
The IF..THEN Statement..... 16
The FOR..NEXT Statement..... 17
GOSUB and RETURN..... 19
Some More Information About Variables..... 19

Chapter Two: System and Utility Commands

Load and Save..... 24
NEW..... 24
RUN..... 24
STOP, END, CTRL-C, reset, and CONT..... 24
TRACE and NOTRACE..... 25
PEEK..... 25
POKE..... 26
WAIT..... 26
CALL..... 26
HIMEM:..... 26
LOMEM:..... 27
USR..... 27

Chapter Three: Editing and Format Related Commands

LIST 29
DEL 29
REM 30
VTAB 30
HTAB 30
TAB 30
POS 31
SPC 31
HOME 31
CLEAR 31
FRE 31
FLASH, INVERSE, and NORMAL 32
SPEED= 32

Chapter Four: Arrays and Strings

DIM 34
LEN 35
STR$ 35
VAL 35
CHR$ 35
ASC 36
LEFT$ 36
RIGHT$ 36
MID$ 37
STORE and RECALL 37

Chapter Five: Input and Output

INPUT 39
GET 40
DATA 40
READ 41
RESTORE 41
PRINT 41
IN# 42
PR# 43

Chapter Six: Statements Which Control Program Execution

GOTO 45
IF..THEN and IF..GOTO 45
FOR..TO..STEP 46
NEXT 47
GOSUB 47
RETURN 48
POP 48
ON..GOTO 48
ON..GOSUB 48
ONERR..GOTO 48
RESUME 49

Chapter Seven: Graphics and Game Controls

TEXT 51
GR 51
COLOR 51
PLOT 52
HLIN 52
VLIN 53
SCRN 53
HGR 53
HGR2 53
HPLOT 53
PDL 54

Chapter One:Introduction

Diamond Computer's implementation of Applesoft (tm) BASIC closely adheres to the definition of standard Applesoft. This allows the TRACKSTAR user to run standard Applesoft programs without any program

Chapter 1:Introduction

Immediate Execution Commands

Deferred Execution Commands

Editing Features

Number Format

Variable Names

Array Variables

The LET Statement

TRACKSTAR BASIC Line Format

The PRINT Statement

The INPUT Statement

The GET Command

READ, DATA, and RESTORE

IN#, PR#, and I/O Redirection

DEF FN and FN <name>

The IF..THEN Statement

The FOR..NEXT Statement

GOSUB and RETURN

Some More Information About Variables

Chapter One:Introduction

Diamond Computer's implementation of Applesoft (tm) BASIC closely adheres to the definition of standard Applesoft. This allows the TRACKSTAR user to run standard Applesoft programs without any program conversions. By keeping TRACKSTAR's implementation identical to Apple's, Diamond Computer insures compatibility with most Apple II programs you will encounter.

If you are attempting to write an original program using Applesoft BASIC, there are several points to remember. First of all, Applesoft BASIC, although originally created by Microsoft (the same folks who brought you IBM PC BASIC), is based on version 2.0 of Microsoft BASIC. IBM PC BASIC is based on Microsoft version 5.0 BASIC. Hence Applesoft is rather primitive compared to IBM PC BASIC. The second thing to keep in mind is that Applesoft does not support any built-in DOS commands (for details about DOS commands consult the DOS 3.3 manual also included in your TRACKSTAR package). Despite the drawbacks present in the language, there is one very big advantage to writing programs in Applesoft--almost two million people own Apple IIs, Apple //es, and Apple //cs which can run Applesoft programs created on the TRACKSTAR system. Even if you don't plan to sell your masterpieces, most likely you have a friend who owns an Apple II system. Since the TRACKSTAR reads and writes Apple DOS 3.3 compatible disks, you can easily swap Applesoft programs with your friends.
NOTE: This manual assumes that you have some familiarity with IBM PC-BASIC and BASIC programming in general. This manual is not a tutorial. If you wish to obtain a good tutorial on the Applesoft programming language, consult "The Applesoft Tutorial" published by Apple Computer Inc. (Apple part #A2L0018 (030-0044-00).

Immediate-Execution Commands

Most TRACKSTAR BASIC (hereafter referred to as "BASIC") commands can be executed by simply typing them after the "]" prompt. For example, if you type the BASIC command:

```
PRINT "HELLO THERE"
```

BASIC immediately prints the string "HELLO THERE" after you press the carriage return. Since these commands are executed immediately after they are entered into the computer, they are called immediate mode commands.

Deferred-Execution Commands

While immediate mode commands are fine for quick calculations and for performing short program computations, substantial work can only be performed using a stored program. Another type of BASIC command is a deferred-execution command. Deferred execution commands always begin with a line number. For example, the statement:

```
10 PRINT "HELLO THERE"
```

is not executed immediately after you enter it from the keyboard. Instead, the execution of this statement is deferred until you type the

BASIC command RUN.

The beautiful thing about deferred execution commands is that you can collect several of them together into a BASIC program and execute them all at once with the RUN command. The statements:

```
10 PRINT "HELLO THERE"
20 PRINT "HOW ARE YOU"
```

will print the two lines:

```
HELLO THERE
HOW ARE YOU
```

when you type RUN after the "]" prompt character.

Like IBM PC-BASIC, TRACKSTAR BASIC uses a line number editing scheme for program storage. Every line in a program must be prefaced with a unique line number. Consider the short sample program:

```
10 PRINT "2+5 is equal to"
20 PRINT 2+5
30 END
```

The first line of the program is at line number 10, the second line is at line number 20, and the last line in the program is at line number 30. Although sequential lines in a BASIC program do not need sequential line numbers, their line numbers must be monotonic (in an increasing order).

Suppose you want to delete line 10 in your program. This task is easily accomplished by typing the line number 10 all by itself on a line. Anytime a line number appears by itself on a line, BASIC deletes that line from the program memory. If the line did not exist, BASIC ignores the delete command.

To replace a line in a program with a new statement, simply type the line number of the old statement followed by the program text you wish to replace the old line with. For example, to replace the PRINT "2+5 is equal to" statement with PRINT "The following should be five:", simple type the program line:

```
10 PRINT "The following should be five:"
```

and BASIC will automatically delete the old line before inserting the new one. Other familiar Microsoft BASIC editing commands include LIST, NEW, and DEL. These commands will be fully described in a later section.

Editing Features

Under PC-DOS, IBM PC-BASIC provides some fancy screen editing commands to help simplify your editing chores. These editing features are not available while running TRACKSTAR BASIC. The Apple uses a different screen editing command structure than the IBM.

Whereas the IBM uses a screen-buffer oriented screen editing system, Apple systems employ a line-buffer screen oriented editing system. Whenever you move the cursor around on the IBM's display screen, whatever logical line the cursor happens to be on is copied into the line buffer (this editing mode was copied from the venerable Commodore computer systems). Whenever you press return, the entire line is copied into the screen buffer regardless of the cursor position within the line.

On the Apple II (and on the TRACKSTAR board since it mimicks the Apple II) the logical line the cursor falls on is not immediately copied into the input line buffer. Instead, the Apple's input line buffer is built up from individual keys typed at the keyboard and copied from the screen. The right arrow key on the keyboard is used to copy the character currently under the cursor into the line input buffer. The back arrow key (backspace) is used to remove the previous character entered into the input buffer. CTRL-X is used to empty the input line buffer and repeat the line input process. Finally, several escape code sequences are used to position the cursor on the screen. An escape sequence is created by pressing the ESC key followed by some other key. Neither the ESC nor the key which is pressed after the ESC is entered into the input line buffer. The following ESC editing commands are available from the keyboard:

ESC A Moves the cursor one position to the right; does not affect the contents of the input line buffer.

ESC B Moves the cursor one position to the left; does not affect the contents of the input line buffer.

ESC C Moves the cursor down one line to the line immediately below the one the cursor is on.

ESC D Moves the cursor up one line leaving the cursor in the same horizontal column.

ESC E Clears the screen from the cursor position to the end of the current line.

ESC F Clears the screen from the cursor position to the end of the screen.

ESC @ Homes the cursor to the upper left hand corner and clears the screen.

ESC I Cursor up (repeated). Identical to ESC-D except that you don't have to repeatedly type ESC between successive "I"s. While in this free running cursor editing mode, pressing I,J,K, or M will move the cursor around on the screen. To exit the free-running ESC editing mode, simply

press the space key once.

ESC J Moves the cursor to the left. This command also enters you into the free-running ESC editing mode. Successive I, J, K, and M characters will reposition the cursor and leave you in the free-running ESC editing mode. This command is the free-running equivalent of the ESC-B command.

ESC K Free-running ESC editing command which moves the cursor one position to the right.

ESC M Free-running ESC editing command which moves the cursor down one line.

Number Format

TRACKSTAR BASIC uses an Applesoft compatible five-byte floating point format. Unlike IBM BASIC, which supplies two floating point formats, single and double precision, Applesoft only provides one. The five-byte format was chosen as a compromise between the limited precision of a single precision floating point value and the inefficiency (both in terms of execution speed and storage size) of a double precision number. TRACKSTAR BASIC floating point values provide better than nine significant digits of precision. Values larger than 1E+38 will produce an overflow error; values less than 3E-39 will be converted to zero.

Unlike IBM BASIC, which provides several fancy output formatting commands for printing floating point numbers, Applesoft BASIC does not provide floating point output formatting capabilities. When a number is printed, the following rules are used to determine the output of the number:

1) If the number is negative, a leading minus sign is output in front of the number.

2) If the absolute value of the number is an integer in the range 0 through 999999999, then the number is printed as an integer using an integer format (i.e., no trailing decimal point or zeroes after the decimal point).

3) If the absolute value of the number is greater than or equal to 0.01 and it is less than 999999999.2, the number is printed using a fixed point format (i.e., in the format "iii.fff").

4) If the number is less than 0.01 or greater than 999999999.2, then scientific notation is used to display the number.

Whenever a number is printed using scientific notation, the format "sm.mmmmmmmmEnxx" is used. "s" is the sign of the number which is either a blank or a minus sign, "m" represents the mantissa digits, "n" is the sign of the exponent (either "+" or "-"), and xx are the two exponent digits.

A number typed at the keyboard or within a BASIC program may contain up to 38 digits. However, only the first ten digits are significant and the tenth digit may be rounded.

Variable Names

BASIC allows you to enter variable names up to 238 characters in length. Unlike IBM BASIC, however, TRACKSTAR BASIC (like Applesoft) only recognizes the first two characters in the variable name. The two variable names "SATURATED" and "SAMEASSATURATED" are valid, but they both refer to the same variable name since the first two letters of both variable names are "SA".

Variable names must begin with an alphabetic character. All other characters in a variable name must be alphabetic or numeric characters. Finally, a variable name must not contain an embedded BASIC reserved word. When an input line is processed by BASIC, all spaces are ignored. Therefore, the statements "PRINT X=5" and "PRINTX = 5" are treated as the same statement. Since all spaces are ignored, any occurrence of a BASIC reserved word within a line is treated specially. Hence, reserved words cannot appear within a variable name.

Variables ending with a "%" or "$" character have a special meaning. A variable followed by a percent sign ("%") is an integer variable. Such variables can only be used to hold integer values in the range -32768 through +32767. A variable followed by a dollar sign ("$") is a string variable. Integer and string variable will be discussed later.

Array Variables

BASIC arrays can be declared in one of two ways: via an auto-dimension mechanism or with the DIM statement. If you specify a variable name followed by an index list (e.g., A(5,6), "(5,6)" is the index list) then BASIC automatically allocates storage for the array. An upper bounds of 10 is used for each dimension declared with an auto-dimensioned variable.

The DIM statement provides the preferred method of dimensioning an array. DIM uses the syntax:

DIM vn(d1,d2,...,dn)

where vn is the variable name, and d1 through dn are the maximum values for each dimension. Array bounds always range from zero to the upper bound declared in the DIM statement (or 10 if the array was auto-dimensioned).

An array of floating point values requires five bytes for each element in the array plus a few extra bytes of overhead. An integer array requires two bytes for each element of the array plus a few bytes of overhead. String arrays require three bytes for each array element plus the required overhead. Note: Applesoft performs all numeric calculations using floating point arithmetic. Unlike most high level languages, integer arithmetic in Applesoft BASIC is actually slower than floating point arithmetic since each integer must be converted to a floating point value before a calculation and then must be converted back to an integer when stored back into the integer variable. Due to the way BASIC operates, you should only use integers when you need to declare a large integer array and you can't afford five bytes for each element.

The LET Statement

Perhaps the most important statement in BASIC is also the least glamorous: the LET statement. The LET statement is used to assign a new value to a variable. The general format for the LET statement is

{LET} <varname> = <expression>

The braces around the LET are used to show that the LET is optional. That is, the statement:

<varname> = <expression>

performs the same operation as the LET statement.

The LET statement computes the value of <expression> and stores this value into the variable specified by <varname>. If <expression> evaluates to a string value, then the variable specified by <varname> must be a string variable. If <expression> evaluates to a numeric value greater than 32767 or less than -32768, then <varname> must be the name of a real (floating point) variable. If <expression> evaluates to a numeric value in the range -32768 through +32767 then <varname> must be the name of a real or integer variable. You will get a type mismatch error if the assignment doesn't follow one of these three rules.

Within a numeric expression, BASIC supports several operators. They are:

- Unary minus, negates the value which follows.
^ Exponentiation
* Multiplication
/ Division
+ Addition
- Subtraction

Relational operators:

= Equality
<> Not equal
< Less than
> Greater than
<= Less than or equal
>= Greater than or equal

NOT Logical "not"
AND Logical "and"
OR Logical "or"

The relational operators return one if the relation is true, zero if the relation is false. Since these operators are the same as the ones found in IBM BASIC, further discussion is not required.

In addition to the standard arithmetic operator, BASIC also supports several arithmetic functions.

PEEK(x) returns the byte at address x. x must be in the range -32768..65535 or an illegal quantity error will result. The negative addresses in the range -32768 to -1 are exactly the same as the positive addresses in the range 32768 to 65535.

USR(x) copies the value x into the BASIC floating point accumulator at address $9D through $A3 (the "$" is used to denote a hexadecimal address) and then calls the machine language subroutine at address $0A. You must set up a 6502 JMP instruction at address $0A before using this function. Whenever the user-written subroutine returns to BASIC, whatever value is left in the BASIC floating point accumulator is returned as the function value.

The POS(x) function returns the current horizontal cursor position. The parameter (x in this case) is ignored by the POS function; you should simply place a zero here.

The FRE(x) function performs a garbage collection operation on the string space and then returns the number of bytes of free space as the function result. The parameter passed to the FRE function is ignored and should contain zero.

The LEN(x$) function returns an integer value denoting the length of the string passed as a parameter to LEN. The parameter must be a valid string expression or BASIC will give you a type mismatch error.

The VAL(x$) function converts the parameter, which is a string containing a series of digits in a valid floating point format, into a numeric value and this value is returned by the VAL function. For example, VAL("123") returns the numeric value 123.

The SCRN(x,y) function returns the color code of the LORES point at coordinate (x,y) on the LORES graphics screen. This number is meaningful only if LORES graphics is currently turned on.

The PDL(x) function returns the current setting of game paddle x. X must be a value in the range zero through three. PDL returns a value in the range 0 through 255.

BASIC supports four trigonometric functions: SIN, COS, TAN, and ATN. The SIN(x) function returns the trigonometric sine of x where x is assumed to be in radians. COS(x) returns the cosine of x (again, x is in radians). TAN(X) returns the tangent of x (x in radians). ATN(x) returns the arctangent, in radians, of x.

BASIC supports two logarithmic functions, EXP and LOG. EXP(x) returns e (2.718289) raised to the xth power. LOG(x) returns the natural logarithm (ln) of x.

The ABS(x) function returns the absolute value of x. If x is positive, ABS(x) simply returns x. If x is negative, ABS(x) returns the positive equivalent of x.

The SGN(x) function returns the sign of x. If x is positive, SGN(x) returns one. If x is zero, SGN(x) returns zero. If x is negative, SGN(x) returns -1.

The INT(x) function returns the integer portion of x. The integer portion of x is the largest integer which is less than or equal to x. This function effectively strips off all digits to the right of the decimal point in x.

The SQR(x) function returns the positive square root of x. X must be a positive number or you will get an illegal quantity error.

The last numeric function supported by BASIC is the RND(x) function. RND produces random numbers. If x is positive, then the parameter is ignored and RND returns a new random number between 0 and 0.999999999. If x is zero, RND returns the last random number generated. If x is a negative value, RND returns a fixed value for each negative value (i.e., RND(-3) always returns the same value, but this value is different than the value returned by RND(-4)). After passing a negative number to RND, a new sequence of random numbers will be generated until you call RND with another negative number.

The ASC(x$) function returns the ASCII value of the first character in the X$ string. This value is usually in the range 0..127, although you can assign larger values to characters in a string with the CHR$ function.

The VAL(x$) takes the string contained in x$, and converts it to a floating point number. BASIC assumes that x$ contains a string of valid digits for a floating point number. VAL stops the conversion on the first non-floating point compatible character.

Built-in string functions include STR$, CHR$, LEFT$, MID$, and RIGHT$. STR$(x) is the converse operation to VAL. It converts the numeric value x into a string of characters. Note that PRINT STR$(x) performs the same operation as PRINT x.

CHR$(x) converts takes the numeric value x (which must be less than 256) and converts it to a one byte string containing the character whose ASCII code corresponds to x. If x is not in the range 0..255, you will get an illegal quantity error.

LEFT$(A$,x) creates a string composed of the leftmost x characters of A$. There must be at least x characters in A$ or an error will result.

RIGHT$(A$,x) creates a string composed of the rightmost x characters of A$. There must be at least x characters in A$ or an error will result.

MID$(A$,x,y) returns a string made up from A$, starting at the xth character in A$ and of length y. There must be at least x+y characters in A$ or an error will result.

Like most high level languages, BASIC evaluates expressions from left to right. The order of evaluation for TRACKSTAR BASIC is the same as other Microsoft BASICs, including IBM PC-BASIC. The order of evaluation can be changed by grouping subexpressions within parentheses. For further information on the order of evaluation, consult your IBM BASIC manual.

TRACKSTAR BASIC only provides one string operator: string concatenation. The "+" symbol is used to denote string concatenation. If the plus symbol appears between two string operands, the result is the string on the left of the plus sign concatenated with the string on the right side of the plus sign. Either two numeric or two string operands must appear on either side of the plus sign. If a string appears on one side and a numeric quantity appears on the other, you will get a type mismatch error.

```
I+J--------Correct
A$+B$------Correct
I+A$-------Incorrect
```

TRACKSTAR BASIC Line Format

The examples discussed so far have only presented a single BASIC statement per program line. Like most reasonable BASICs, TRACKSTAR BASIC allows multiple statements per line.

Before describing how to write a program with multiple statements per line, a description of exactly what constitutes a line is in order. Due to the limited size (40 columns) of the original Apple screen, BASIC statement lines were allowed to continue across several lines. A physical line is the actual 40- or 80-column line which makes up one line of text

on the video display. A physical line is always 40 or 80 columns long, depending on whether or not you have activated the TRACKSTAR 80-column display. Physical line dimensions are important to the programs you write since you want to make sure your output always fits on a physical line. But as far as BASIC is concerned, the physical line dimension is meaningless.

BASIC uses a logical line format. Logical lines can be up to 255 characters long. Whenever the end of a physical line is encountered while entering or listing a BASIC line, the cursor "wraps around" to the beginning of the next line on the display.

You will rarely, if ever, use 255 characters in any BASIC statement. In fact, most of the time 255 characters are far more than you will ever need. BASIC allows you to enter multiple BASIC statements on a single line by simply separating the BASIC statements with a colon. For example, the two statements:

```
10 PRINT "Hello there"
20 PRINT "How are you"
```

can be combined into a single line as:

```
10 PRINT "Hello there" : PRINT "How are you"
```

Multiple statements per line execute faster than an equivalent program with individual statements per line (when discussing BASIC programs, "line" will always mean a logical line). Furthermore, a program which employs multiple statements per line will require less memory than an equivalent program with a single statement on each line. However, a program containing multiple statements per line, without regard to the structure of the program, is much harder to read and maintain than a program with single statements per line.

The PRINT Statement

TRACKSTAR BASIC supports several variations of the important PRINT command. The formats for this statement are:

```
PRINT
PRINT expr [{,|;} expr]
PRINT expr [{,|;} expr] {,|;}
PRINT ,
PRINT ;
```

The square brackets indicate items which can be repeated zero or more times; the items inside the curly braces separated by vertical bars ("|") represent a selection, you must choose one of the items inside the curly braces separated by the vertical bars; expr represents a general BASIC string or numeric expression. A little less formally, the PRINT statement allows zero or more expressions, separated by a comma or semicolon, with a

possible leading or trailing comma or semicolon.

The question mark ("?") may be used as an abbreviation for PRINT; when listing your program, BASIC will list the question mark as PRINT.

Without any options, the PRINT statement prints a carriage return/line feed pair which moves the cursor to the beginning of the next line on the screen. If some expressions are specified, BASIC prints the values of these expressions. If neither a comma or a semicolon ends the list of expressions, the PRINT statement terminates the print operation by printing a carriage return/line feed combination.

Items separated by semicolons in the PRINT list are printed without any interleaving spaces. That is, if a semicolon appears between two items in the output list, they will be printed adjacent to one another. When a semicolon appears at the end of the line, BASIC does not print a carriage return and line feed. Thus, the very next time BASIC performs an output operation, the new output will be printed directly at the end of the last data that was output by PRINT statement. PRINT followed by only a semicolon performs no operation.

If an item in the output list is followed by a comma, then the next item is printed in the next tab field. BASIC supports three tab fields. In the forty-column mode, the first tab field comprises columns one through sixteen; the second tab field occupies columns seventeen through thirty-two; and the third tab field occupies the remaining eight columns. In the eighty column mode, the third tab field comprises columns 33 through 80.

Whenever a comma is encountered in the PRINT list, the output cursor is moved to the beginning of the next tab position. If the output cursor is already past the third tab column, then PRINT moves the cursor to the first tab column on the next physical line. If a PRINT statement simply contains a comma in the output list, then the PRINT routine simply positions the cursor at the beginning of the next tab field.

Unlike the more powerful BASICs such as the IBM PC-BASIC, Applesoft (and TRACKSTAR BASIC) does not provide any output formatting facilities for floating point numbers. There is no PRINT USING or similar function in BASIC. A moderate amount of formatting can be performed by converting a floating point number to a string (with the STR$ function) and then using string functions to format the data. Several utility packages are also available from Apple dealers to extend Applesoft's formatting abilities. Most of these packages are also compatible with TRACKSTAR BASIC.

The INPUT Statement

TRACKSTAR BASIC provides the standard INPUT statement for entering data from the keyboard. There are two forms of the input statement. They are:

```
INPUT "prompt"; <input list>
INPUT <input list>
```

The first form of the INPUT statement prints the specified prompt string before requesting input. The second form of the INPUT statement simply prints a question mark before requesting input.

The input list consists of one or more variable names separated by commas. If the variable is a numeric variable, the INPUT statement will read a number from the current input device (see IN# below for a discussion of input devices). If the variable is a string variable, then the INPUT statement reads a string from the current input device.

Input values must be separated by a comma or a carriage return. If you do not enter enough items to satisfy the input list, BASIC prints two question marks as a prompt and requests further input. If a colon is encountered in the middle of the input, the remainder of the line is ignored; i.e., INPUT treats the colon as a carriage return during execution. If you wish to enter a comma or a colon as part of a string during input, simply enclose the string within quotation marks.

INPUT cannot be used in the immediate mode, any attempt do so will cause an ?ILLEGAL DIRECT ERROR message to be displayed. CTRL-C can be used to terminate program execution in an INPUT statement, but only if the CTRL-C appears as the first character of the input statement. The program will halt when return is pressed.

The GET Command

The GET command allows you to read a single character from the keyboard. A single string or numeric variable name must follow the GET command. For example, GET A$ reads a single character from the Apple keyboard (you do not have to press return) and stores this character into A$.

GET can be used with numeric variables, but it exhibits so many peculiarities that you should avoid such use. Instead, read the character into a string variable and use the VAL function to convert it to a number.

READ, DATA, and RESTORE

The READ, DATA, and RESTORE statements are used to initialize data within a BASIC program. READ's syntax is identical to INPUT. The only difference is that READ does not allow a prompt string immediately after the READ command. An example of a READ statement is

```
READ A,B,C$
```

The READ statement reads the data for the variables in the input list from DATA statements in the program rather than from the current input device. The data which immediately follows a DATA statement is identical to the information which is allowed as data for the INPUT statement. Whenever a READ statement is encountered in a program for the first time, BASIC scans the program from the beginning of the program for the first DATA statement in the program. The necessary data is read from this DATA statement and a data pointer is left pointing to the first data item unprocessed by the

READ statement. Whenever the end of a DATA statement is encountered, the data pointer is set to point at the beginning of the next DATA statement in the program. Attempting to read data beyond the last DATA statement in the program will produce an out of data error.

The READ command sequentially reads data from successive DATA statements until the data list is exhausted. This sequential access isn't always the most convenient method for reading data. The RESTORE command resets the data pointer to point at the first DATA statement in the program. This allows you to re-read the data should your program need to re-initialize the data. RESTORE accepts no parameters. If you follow RESTORE with anything other than a colon or a return, you will get a ?SYNTAX ERROR.

IN#, PR#, and I/O Redirection

The Apple II computer system provides seven peripheral slots specifically designed for character I/O devices. The IN# and PR# commands are used to access such devices. IN#n (where n is a digit in the range 0..7) is used to redirect the input channel; PR#n (n is in the range 0..7) is used to redirect the output channel.

IN#0 and PR#0 tell BASIC to obtain its input from the keyboard and send its output to the video display (respectively). Slot zero on the Apple II always refers to the built-in video display and keyboard console. On the TRACKSTAR board, slot zero always refers to the IBM PC video display and keyboard. Therefore, IN#0 and PR#0 are used to turn off peripheral devices turned on with a PR#n or IN#n command.

PR#n redirects the standard output channel to the device present in the specified slot. For example, PR#1 sends all output to the device in slot one, PR#3 sends all output to the device in slot three, etc. Note that these slots do not correspond to cards plugged into the slots of an IBM PC. These commands refer to slots present in an Apple II. The TRACKSTAR board, since it doesn't supply seven Apple compatible slots, emulates these slots by using a virtual slot mechanism. Most Apple owners place a printer interface card in slot one, an eighty-column card in slot three, and a disk drive controller card in slot six. Therefore, the TRACKSTAR board emulates these important devices using the printer interface and disk drives on your IBM PC and the eighty-column display on the TRACKSTAR board. For more information on the peripheral devices supported by the TRACKSTAR board, consult Appendices 1 and 2.

The PR# command is a true I/O redirection command. All output which normally goes to the video display will be sent instead to the device in the specified slot. For example, PR#1 will direct all output on the TRACKSTAR board to the printer interface. All data which would normally be sent to the video screen via PRINT, LIST, INPUT, and the editing functions, is sent to the printer interface on your IBM PC. To turn the printer off and resume sending information to the video screen, the PR#0 command is used.

PR#6 is a special case of the PR# command. The TRACKSTAR board emulates an Apple disk drive in slot six. The PR#6 command is used to boot an Apple diskette in drive A of your IBM PC (or in the optional Apple drive if you have installed one). Once TRACKSTAR DOS boots, the DOS handles redirection of I/O using its own built-in commands and the PR# command is not used. NOTE: in fact, once TRACKSTAR DOS is booted, you cannot use the standard PR# and IN# commands since these commands would disconnect the DOS. TRACKSTAR DOS provides its own versions of the IN# and PR# commands. Consult the TRACKSTAR DOS manual for more details.

The IN#n command is used to redirect the standard input channel on an Apple II. For example, IN#4 instructs BASIC to obtain further input from the peripheral device in slot four. After typing IN#4, all INPUT statements, GET statements, and lines of text read in the immediate mode will come from the peripheral device in slot four.

The value which follows the IN#n or PR#n command must be in the range 0..7. Values larger than 255 will produce a range error, values in the range 8..255 may cause TRACKSTAR BASIC to hang up (facilitating the use of the TRACKSTAR reset sequence). If a peripheral slot is not supported by the TRACKSTAR system and you specify that slot number in a PR#n or IN#n command, this may also cause the system to hang. Consult Appendices 1 and 2 for the slots supported by the TRACKSTAR system.

DEF FN and FN <name>

The DEF FN statement allows you to define a function in BASIC. The syntax for the DEF FN command is

```
DEF FNname (rvar) = expr
```

where "name" is the function name (which must be a valid TRACKSTAR BASIC variable name), "rvar" is a REAL (floating point) variable name, and "expr" is an expression which returns a numeric result. You cannot create a string or integer function with the DEF command.

A function can be redefined during the execution of a BASIC program. The execution of a DEF statement performs the function definition. Since BASIC only recognizes the first two characters of an identifier, care must be taken when defining functions since you can inadvertently redefine a function without realizing what's happening. Consider the two lines of BASIC code:

```
10 DEF FNLAZER(X) = X*5+X
20 DEF FNLATER(Y) = Y+1
```

The first statement creates the function LA (zer) for BASIC. The second statement creates the function LA (ter). Since the function names are the same for the first two characters, BASIC treats the names as the same and the second statement is simply a redefinition of the function declared in the first statement. The end result is that the definition of LAZER is lost.

The rvar parameter in the function definition statement is called a dummy argument. During the execution of the function, the value passed in the parentheses is substituted for the rvar variable in the function expression. If you have a variable named rvar in your program, it is not affected by the execution of the function. However, if rvar is the name of a variable in your program, then that variable is not available to your function unless you specifically pass that variable as the parameter to the function. User defined functions must always have exactly one parameter and the parameter must be a numeric value.

To actually employ a user defined function, the FNname command is used. The FN function is only allowed in an expression. It can only be used wherever a real variable is allowed. For example, to assign the value of the LAZER function to the variable Y, you should use the BASIC statement:

```
10 Y = FNLAZER(expr)
```

where expr is the parameter passed to the LAZER function.

The DEFinition's rvar parameter doesn't have to be present in the expression to the right of the equals sign in the function definition. However, an expression must always be supplied when invoking the function and since this expression is always evaluated, it must be legal.

The IF..THEN Statement

The IF..THEN statement is used to conditionally execute some sequence of statements. There are three forms of the IF..THEN statement:

```
10 IF expr THEN stmt:stmt:stmt:...

10 IF expr GOTO number

10 IF expr THEN number
```

The first form of the IF..THEN statement executes all of the statements on the same line if the expr expression evaluates to one (true). If the expression evaluates to zero (false), then the IF..THEN statement skips the remaining statements on the current line and continues execution with the next sequential program line.

The second and third forms of the IF..THEN statement perform identical operations. If the expression evaluates to one (true), then BASIC jumps to the program line number labeled "number". If the expression evaluates to zero (false), then the IF statement continues execution at the next sequential program line. Note that BASIC cannot execute any statements which follow this form of the IF statement on the same logical line. All statements following an IF..GOTO or IF..THEN number statement on the same logical line are ignored.

The relational operators, =, <>, <, >, <=, and >= are extremely useful in conjunction with the IF..THEN statement. These operators compare the operand to the left of the operator to the operand on the right of the operator and return zero if the relation is false, one if the relation is true. For example, "IF X<10 THEN X=X+1" increments X if and only if X is less than 10.

The relational operators can be used in any expression, not just expression inside an IF..THEN statement. The BASIC statement X= Y<10 stores zero in X if Y is greater than or equal to ten, it stores one into X if Y is less than ten.

The FOR..NEXT Statement

The FOR..NEXT statement provides BASIC with a looping construct. The syntax for the FOR..NEXT loop is

```
10  FOR rvar = start TO end {STEP stepsize}

    {BASIC statements}

90  NEXT {rvar}
```

The braces surround optional items.

The FOR..NEXT loop begins by assigning start (which must be a numeric value) to rvar (which must be a real variable name). Then the BASIC statements between the FOR and the NEXT statements are executed. When the NEXT is encountered, the FOR statement adds stepsize to rvar (FOR adds one to rvar if stepsize isn't specified) and then compares rvar to end. If the stepsize was not specified or if the stepsize is positive, the FOR..NEXT loop will repeat the execution of the loop as long as rvar is less than or equal to end. If the stepsize is negative, then the FOR..NEXT loop repeats the body of the loop (all of the statements between the FOR and NEXT statements) until rvar is less than end.

Unlike later versions of Microsoft BASIC (like IBM PC BASIC), Applesoft and TRACKSTAR BASIC always execute the body of the loop at least once regardless of the starting and ending values. So a loop of the form:

```
10 FOR I=10 TO 1 STEP 4
20 PRINT I
30 NEXT I
```

will print 10 even though the loop really shouldn't execute at all. If you need to insure that such a loop doesn't get executed if the initial values are out of range, a simple IF..THEN statement can be used to skip the loop. For example, the following code behaves like a simple FOR..NEXT loop in IBM PC BASIC:

```
100 IF X > Y THEN 500
```

```
110 FOR Z = X TO Y

   {Body of the loop}

490 NEXT Z
500 REM
```

The preceding IF..THEN statement will force BASIC to skip over the loop if the starting value is out of range.

FOR loops can be "nested" within a program. That is to say, you can include a FOR loop within the body of another FOR loop. The only restriction is that the NEXT statements must match the appropriate FOR loop. The following is an example of a pair of nested FOR loops:

```
10 FOR I=1 TO 10
20 FOR J=10 TO 1 STEP -1
30 PRINT I,J
40 NEXT J
50 NEXT I
```

Note that the first NEXT statement encountered must match the ~~last~~ FOR statement encountered. In this case, the NEXT J statement appears first because the FOR J=10 TO 1 STEP -1 statement was the last FOR statement encountered.

If you reverse the order of the NEXT statements (e.g., if NEXT I appeared before NEXT J in the previous example), then BASIC ignores the innermost loop. Upon encountering the out of order NEXT statement after completion of the outermost loop, the system will stop and print a "NEXT WITHOUT FOR" error message.

Whenever two NEXT statements appear next to one another (as in the previous example), BASIC allows a shorthand method of describing the NEXT statements. The true syntax for the NEXT statement is

```
NEXT var1 [,var2]
```

where the information in the square brackets can be repeated as many times a necessary. This form of the NEXT statement behaves exactly like a "NEXT var1" statement until the FOR loop associated with NEXT var1 completes. Then this NEXT statement behaves like a NEXT var2 statement. This process is repeated for each variable name specified after the NEXT statement. Like individual NEXT statements, the variables must appear in the reverse order of the FOR statements or the system will ignore any innermost loops and give you a "NEXT WITHOUT FOR" error message when one of the inner variable names is encountered. The previous example, correctly coded to handle this form of the NEXT statement, is

```
10 FOR I=1 TO 10
20 FOR J=10 TO 1 STEP -1

   {Body of the loop goes here}
```

```
90 NEXT J,I
```

GOSUB and RETURN

GOSUB and RETURN provide the subroutine mechanism necessary for writing sophisticated programs. Whenever a GOSUB is encountered, BASIC branches to the line number which must immediately follow the GOSUB command. BASIC saves the address of the next statement which follows the GOSUB statement so, upon encountering a RETURN instruction, BASIC can return and continue program execution at the statement immediately following the GOSUB command. The syntax for the GOSUB command is

```
GOSUB linenumber
```

When executed, BASIC jumps to the subroutine specified by linenumber (which must be an integer value in the range 0..63999). The syntax for the RETURN statement is simply:

```
RETURN
```

No parameters are allowed. If a RETURN is executed without a corresponding GOSUB, BASIC prints a "RETURN WITHOUT GOSUB" error message.

Some More Information About Variables

There are three primitive data types supported by BASIC: floating point (real) variables, integer variables, and string variables. BASIC also allows you to create arrays of any of these three types.

A real variable uses the five-byte Microsoft floating point format. This format supports better than nine significant digits of precision. BASIC converts the decimal numbers you enter via program statements INPUT and READ into a binary format that the computer understands. This binary format is converted back into a decimal format with the PRINT statement. Because of rounding and other inaccuracies present in the BASIC floating point package, certain mathematical operations like multiply, divide, and the transcendental functions do not always return the exact result you may expect.

To help overcome inaccuracies present in any floating point package (be it binary or decimal), you can round the value you're working with by using the formula:

```
X = INT(X*10^D+.5)/INT(10^D+.5)
```

D is the number of decimal digits to the right of the decimal point you wish to allow. A faster and more compact method for performing this computation is to use the formula:

```
X = INT(X*P+.5)/P
```

where P was loaded with 10 for one decimal place, 100 for two decimal places, etc. This formula works for values in the range 1..999,999,999. Since the formula is so useful, you may want to create some user-defined functions to perform this calculation for you. For example, R2 (below) rounds the value passed to two digits; R3 rounds the value passed to three digits to the right of the decimal point:

```
10 DEF FNR2(X)=INT(X*100+.5)/100
20 DEF FNR3(X)=INT(X*1000+.5)/1000
```

Integer variables can be used to hold integer values which are limited to the range -32768..+32767. To create an integer variable, simply append a percent sign ("%") to the end of the variable name. For example, I, when encountered for the first time in a BASIC program, creates a real variable; I% creates an integer variable. BASIC reserves the same amount of space for simple integer variables as it does for real variables. Therefore, there is no memory savings in using a simple integer variable. Furthermore, all calculations are performed in floating point. Therefore, unlike most high level languages, using integer variables actually produces slower code than using pure floating point variables since the integer variable requires the extra step of integer to floating point and floating point to integer conversion while the variable is being loaded and stored in main memory.

Integer arrays, however, only require two bytes per element (compared to five bytes for floating point arrays). Therefore, an integer array is more compact than an equivalent floating point array. So if you are running out of memory and you are working with integer values in the range -32768..+32767, then an integer array may work for you.

String variables are created by post-fixing a dollar sign ("$") to the end of the variable name. For example, when X$ is encountered in a program for the first time, a string variable is created. Simple string variables (i.e., string variables which are not arrays) require the same amount of space as do simple floating point and integer variables assuming that you do not assign a literal string value to the string. String arrays require three bytes per array element: a one-byte length and a two-byte pointer to the string in the string heap space.

Whenever a string variable is assigned a non-empty string, the string literal is copied into the dynamic string heap, and the length and pointer fields of the string record are set appropriately. Due to BASIC's efficient use of string memory, only the necessary number of bytes required to hold the string are allocated. You do not have to predeclare the length of a string. The drawback is garbage collection. As strings are assigned different values, string memory is fragmented into several sections of used and unused memory blocks. Once the memory allocation code in BASIC uses up all of the string heap, a garbage collection operation takes place to free up all of the unused memory blocks in the string heap. In some rare cases this garbage collection operation has been known to take as long as five minutes. Usually, however, garbage collection is completed in less than 20 seconds. So if the computer

mysteriously hangs up on you, be patient, BASIC may be performing a garbage collection operation.

BASIC supports several string-oriented functions. As mentioned in a previous section, these functions are ASC, VAL, STR$, CHR$, LEFT$, RIGHT$, MID$, and string concatenation.

ASC and CHR$ are useful for converting characters to the ASCII code and vice versa. The syntax for the ASC function is ASC(svar$) where svar$ is the name of a currently defined string. ASC returns a value in the range 0..127 which represents the ASCII code of the first character in the svar$ string. If the length of the string is zero, or if the string is not defined, BASIC will give you an illegal quantity error.

The CHR$ function is the converse operation to the ASC function. The syntax for CHR$ is CHR$(nvar) where nvar is a numeric variable containing a value in the range 0..255. If nvar is greater than 255, BASIC will give you an "?ILLEGAL QUANTITY ERROR".

The CHR$ and ASC functions can be used together to perform certain conversions. For example, if you want to index into an array using the alphabetic characters "A".."Z", the following code could be used:

```
AR(ASC(INDEX$)-ASC("A"))
```

This code computes the array index by converting the first character in the INDEX$ string, converting it to its ASCII code using the ASC function, and then subtracting the ASCII code for "A" from the ASCII code for the first character of INDEX$. This produces a value in the range 0..25, where 0 corresponds to A, 1 corresponds to B, etc. This trick relies on the fact that the ASCII character set defines the alphabetic characters using sequential numeric codes.

The VAL and STR$ functions convert strings to numbers and numbers to strings (respectively). The syntax for VAL is VAL(svar$) where svar$ is a string variable containing a string of digits which correspond to a valid floating point number. VAL converts this string to the corresponding floating point number and returns this number as the function result.

The STR$ function performs the opposite operation--it converts a numeric value into a string containing the decimal representation of the floating point value passed as the STR$ parameter. The syntax for the STR$ function is STR$(nvar) where nvar is some numeric variable.

VAL and STR$ can be used to format floating point values for output. If you have a number in the range 1..999,999,999, the following code will format it so that exactly two digits appear after the decimal point:

```
100 X  = INT(X*100+.5): REM Convert to two decimal places
110 S$ = STR$(X) :Convert to a string.
120 IF LEN(S$) = 1 THEN PRINT "0.0";S$ :REM Format is 0.0n
130 IF LEN(S$) = 2 THEN PRINT "0.";S$  :REM Format is 0.nn
140 IF LEN(S$) > 2 THEN PRINT
     LEFT$(S$,LEN(S$)-2);".";RIGHT$(A$,2):REM Format is nn.nn
```

The LEN function returns the length of the string passed as a parameter. In the example above, the LEN function was used to determine if the string was equal to one, two, or more characters in length. If the length of the string was less than three, then the number was less than one. If the length of the string was greater than or equal to three, the number was greater than or equal to one.

The LEFT$, RIGHT$, and MID$ functions are used to extract a portion of a string. In the example above, LEFT$ was used to extract the integer portion of the number (which consisted of all characters to the left of the second character in the string, assuming that the string was longer than two characters). The RIGHT$ function was used to extract the fractional portion of the string when the string was greater than two characters in length.

Variables using different type suffixes are unique, regardless of the variables' names. For example, the variables X, X%, and X$ are all different variables. Furthermore, array variables are different from simple (also called scalar) variables. All of the following variables are different in BASIC:

```
X, X(1), X%, X%(1), X$, and X$(1)
```

Although BASIC allows you to create different variables using these different suffixes, you should avoid this type of programming practice since it makes your programs much harder to read.

Chapter Two: System and Utility Commands

LOAD and SAVE

NEW

RUN

STOP, END, CTRL-C, reset, and CONT

TRACE and NOTRACE

PEEK

POKE

WAIT

CALL

HIMEM:

LOMEM:

USR

Chapter Two: System and Utility Commands

LOAD and SAVE

The LOAD and SAVE commands on the original Apple IIs were used to load and save data on cassette tape. Since the TRACKSTAR board does not support the cassette tape hardware, attempting to use the BASIC LOAD or SAVE command will cause the system to hang. Refer to the TRACKSTAR DOS manual for information concerning loading and saving BASIC programs to disk.

NEW

The NEW command can be used in the immediate or deferred mode of operation. It immediately clears the program and variable space, deleting any program currently in memory. If the NEW command is executed in the deferred mode (i.e., while a program is executing), then the execution of the program is stopped.

RUN {lineno}

RUN clears all variables, stacks, and other execution time pointers and begins executing the program in memory at the line number specified after the RUN command. If the optional line number is not specified, then BASIC begins executing the program in memory at the first line of the program. BASIC returns control to the user if there is no program in memory when the RUN command is issued.

STOP, END, CTRL-C, reset, and CONT

STOP, when encountered during the execution of a BASIC program, immediately ceases execution and returns control to the user. The STOP command also prints the line number of the line containing the STOP command. STOP is useful for setting breakpoints in your program since whenever STOP is encountered it tells you where the program stopped.

The END statement terminates program execution, like the STOP command, except it doesn't print the nasty little message about how the program stopped. END provides a "cleaner" method of terminating program execution than does STOP.

CTRL-C has the same effect as encountering a STOP command in your program. Whenever you press CTRL-C, the program stops and prints a little message telling you the line number where the program was stopped. CTRL-C can also be used to terminate a program listing. It can also be used to interrupt an INPUT, but only if the CTRL-C character was pressed as the first character of the INPUT statement. The execution of the INPUT statement will not be interrupted until a return is pressed.

The TRACKSTAR reset sequence stops any BASIC program and returns you to the BASIC editor mode.

The BASIC CONT command can be used to restart a program stopped with the STOP, END, and CTRL-C commands. Program execution resumes at the next statement, not the next line number in the file. If a program has not been halted, CONT does nothing. If a program's execution was terminated by using the TRACKSTAR reset seqence, the effect of the CONT command is unpredictable. If an INPUT statement was stopped by the CTRL-C command, attempting to CONTinue the program execution will produce a syntax error. Once a program is stopped, it cannot be CONTinued if you make modifications to the source file or if you perform any operation which results in an error message.

If CONT is encountered in the deferred execution mode (i.e., as a program statement) the system will hang. You can regain control of the system by pressing CTRL-C, however, you cannot restart the program by pressing CONT; this will only cause the system to hang again.

TRACE and NOTRACE

TRACE and NOTRACE are debugging aids to help you trace through your program. When TRACE is set (via the TRACE command), BASIC displays the line number of each line executed. This trace display is turned off by the execution of the NOTRACE command. Neither command allows any parameters.

Note: the TRACE command is of limited use when you are running TRACKSTAR DOS. For more information, consult the TRACKSTAR DOS manual.

PEEK

PEEK is a function which returns the byte at the address specified by PEEK's single parameter. The syntax for the PEEK command is:

PEEK(expr)

where expr is a numeric expression which evaluates to a value in the range -65535 to +65535. Generally you should use positive addresses, however, many programs converted from Apple's older Integer BASIC will specify negative addresses, so TRACKSTAR BASIC allows the input of negative addresses for compatibility reasons. If the address is outside this range, BASIC will give you an "?ILLEGAL QUANTITY ERROR".

POKE adrs,data

This command stores the value of the second expression (data) into the address specified by the first expression (adrs). The data parameter must be a value in the range 0..255 or you will get an "?ILLEGAL QUANTITY ERROR". Likewise, the adrs parameter must be a value in the range -65535..+65535 or you will get the "?ILLEGAL QUANTITY ERROR" message.

POKE is a dangerous command! By inadvertently POKE'ing data into arbitrary locations you may wipe out the BASIC program in memory or you may alter the state of the TRACKSTAR program, invalidating the results of your program's execution.

WAIT

The WAIT command is used to test a hardware port. The syntax for the WAIT statement is

```
10 WAIT adrs, mask {, invrt}
```

where adrs is a numeric expression which evaluates to a value between -65535 and +65535; mask and invrt are expressions which evaluate to a number in the range 0..255.

WAIT fetches the byte at the memory location specified by adrs, logically ANDs the value fetched with the second parameter, and then exclusive-ORs this data with the value of the invrt parameter. The invrt parameter is optional, if it is not present, zero is used. WAIT enters a delay loop until this value is non-zero. The only way to abort the WAIT operation is to use the reset sequence.

CALL expr

The CALL command calls the 6502 subroutine at the address specified in the single parameter following the CALL command. The parameter, expr, must evaluate to a number in the range -65535 through +65535. If expr is outside of this range, BASIC will give you an "?ILLEGAL QUANTITY ERROR".

Equivalent positive and negative addresses may be used interchangeably, for example, "CALL -936" and "CALL 64600" perform the same operation.

Control is returned to BASIC whenever the 6502 machine language program executes a return from subroutine instruction (RTS) without any nested subroutine calls on the stack.

HIMEM: expr

HIMEM: sets the highest memory location available to BASIC. The single parameter, expr, is the address of the last memory location available to BASIC. Without TRACKSTAR DOS installed, the maximum address available to BASIC is 49152. With TRACKSTAR DOS installed, the highest address available to BASIC is 38399. The HIMEM command is normally used

CTRL-C has the same effect as encountering a STOP command in your program. Whenever you press CTRL-C, the program stops and prints a little message telling you the line number where the program was stopped. CTRL-C can also be used to terminate a program listing. It can also be used to interrupt an INPUT, but only if the CTRL-C character was pressed as the first character of the INPUT statement. The execution of the INPUT statement will not be interrupted until a return is pressed.

The TRACKSTAR reset sequence stops any BASIC program and returns you to the BASIC editor mode.

The BASIC CONT command can be used to restart a program stopped with the STOP, END, and CTRL-C commands. Program execution resumes at the next statement, not the next line number in the file. If a program has not been halted, CONT does nothing. If a program's execution was terminated by using the TRACKSTAR reset seqence, the effect of the CONT command is unpredictable. If an INPUT statement was stopped by the CTRL-C command, attempting to CONTinue the program execution will produce a syntax error. Once a program is stopped, it cannot be CONTinued if you make modifications to the source file or if you perform any operation which results in an error message.

If CONT is encountered in the deferred execution mode (i.e., as a program statement) the system will hang. You can regain control of the system by pressing CTRL-C, however, you cannot restart the program by pressing CONT; this will only cause the system to hang again.

TRACE and NOTRACE

TRACE and NOTRACE are debugging aids to help you trace through your program. When TRACE is set (via the TRACE command), BASIC displays the line number of each line executed. This trace display is turned off by the execution of the NOTRACE command. Neither command allows any parameters.

Note: the TRACE command is of limited use when you are running TRACKSTAR DOS. For more information, consult the TRACKSTAR DOS manual.

PEEK

PEEK is a function which returns the byte at the address specified by PEEK's single parameter. The syntax for the PEEK command is:

PEEK(expr)

where expr is a numeric expression which evaluates to a value in the range -65535 to +65535. Generally you should use positive addresses, however, many programs converted from Apple's older Integer BASIC will specify negative addresses, so TRACKSTAR BASIC allows the input of negative addresses for compatibility reasons. If the address is outside this range, BASIC will give you an "?ILLEGAL QUANTITY ERROR".

POKE adrs,data

This command stores the value of the second expression (data) into the address specified by the first expression (adrs). The data parameter must be a value in the range 0..255 or you will get an "?ILLEGAL QUANTITY ERROR". Likewise, the adrs parameter must be a value in the range -65535..+65535 or you will get the "?ILLEGAL QUANTITY ERROR" message.

POKE is a dangerous command! By inadvertently POKE'ing data into arbitrary locations you may wipe out the BASIC program in memory or you may alter the state of the TRACKSTAR program, invalidating the results of your program's execution.

WAIT

The WAIT command is used to test a hardware port. The syntax for the WAIT statement is

```
10 WAIT adrs, mask {, invrt}
```

where adrs is a numeric expression which evaluates to a value between -65535 and +65535; mask and invrt are expressions which evaluate to a number in the range 0..255.

WAIT fetches the byte at the memory location specified by adrs, logically ANDs the value fetched with the second parameter, and then exclusive-ORs this data with the value of the invrt parameter. The invrt parameter is optional, if it is not present, zero is used. WAIT enters a delay loop until this value is non-zero. The only way to abort the WAIT operation is to use the reset sequence.

CALL expr

The CALL command calls the 6502 subroutine at the address specified in the single parameter following the CALL command. The parameter, expr, must evaluate to a number in the range -65535 through +65535. If expr is outside of this range, BASIC will give you an "?ILLEGAL QUANTITY ERROR".

Equivalent positive and negative addresses may be used interchangeably, for example, "CALL -936" and "CALL 64600" perform the same operation.

Control is returned to BASIC whenever the 6502 machine language program executes a return from subroutine instruction (RTS) without any nested subroutine calls on the stack.

HIMEM: expr

HIMEM: sets the highest memory location available to BASIC. The single parameter, expr, is the address of the last memory location available to BASIC. Without TRACKSTAR DOS installed, the maximum address available to BASIC is 49152. With TRACKSTAR DOS installed, the highest address available to BASIC is 38399. The HIMEM command is normally used

to reserve space for a 6502 machine language program in the memory space. With DOS installed, all of the memory between HIMEM: and 38399 ($95FF) is available for machine language programs (assuming the default MAXFILES value, see the TRACKSTAR DOS manual).

If you need the current value of HIMEM: for some calculation (such as computing the load address of a 6502 machine language program), it can be obtained by looking at memory locations 116 and 115 (decimal). E.g., PRINT PEEK(116)*256+PEEK(115) prints the current HIMEM: value.

LOMEM: expr

LOMEM: is used to set the lowest memory address available to a BASIC program's variable space. It does not set the lowest memory address available to BASIC. The BASIC source program is always stored below LOMEM: (between addresses $801 and LOMEM:).

LOMEM: is primarily used to protect variables from being overwritten by the HIRES graphics page. By setting LOMEM:16384 you can protect the variables from the HIRES graphics screen number. This does not, however, protect your BASIC source program from the HIRES page. You must ensure that your program is less than 6K long to prevent it from being overwritten by the HIRES graphics page.

USR(expr)

USR is a user-definable machine language function. Whenever USR appears within an expression, BASIC parses the expression and stores the floating point value into locations $9D..$A3 (the BASIC floating point accumulator) and then jumps to location $A. Before using the USR function, you must store a JMP instruction at address $A, $B, and $C which will transfer control to the appropriate subroutine to handle the USR function. The 6502 machine language program returns to BASIC via a return from subroutine instruction (RTS). Whatever value is stored in the floating point accumulator will be returned as the function value.

Chapter Three: Editing and Format-Related Commands

LIST

DEL

REM

VTAB

HTAB

TAB

POS

SPC

HOME

CLEAR

FRE

FLASH, INVERSE, and NORMAL

SPEED

Chapter Three: Editing and Format-Related Commands

LIST {linenol} {-lineno2}
LIST {linenol} {,lineno2}

Note: optional items are enclosed by brackets.

LIST is used to display, or LIST, the text which makes up the BASIC program. If none of the optional parameters are specified, LIST will display the entire program. If a single line number follows the LIST command, then BASIC will list only the line specified by the LIST command.

If two line numbers are specified, separated by a comma or a dash, then BASIC will list the specified range of lines. The second line number must be greater than the first line number or BASIC will ignore the LIST request. There is one exception to this rule. If the second line number is zero, BASIC will list from the first line specified to the end of the program.

BASIC attempts to list program lines in a form suitable for a 40-column display. Unfortunately, the listing algorithm leaves a little to be desired. If you would like to produce a straight listing (especially important when sending a listing to a printer) then issue the two commands:

```
HOME
POKE 33,33
```

These commands clear the screen and then set the text window to 33 columns. BASIC begins formatting text when text is stored into the 34th column, so setting the text window to 33 columns prevents the formatting from taking place. After you are through making your listing, restore the screen window to 40 columns using the command POKE 33,40.

LISTing can be aborted by pressing CTRL-C.

DEL linenol, lineno2

DEL is used to delete a range of lines from a BASIC program. It can only be used to delete a range of lines. To delete a single line, simply type the line number of the line you wish to delete.

When using DEL, linenol must be less than lineno2 and they must both be positive, non-zero numbers. If linenol is zero, then DEL only deletes line number zero, regardless of whatever value lineno2 happens to be. Lineno2 is ignored in this case.

If DEL is encountered in the deferred execution mode, it works exactly as described here, but then it stops the program. Such a halted program cannot be restarted with the CONT command.

REM text

The REM statement is used to allow the storage of comments within a program. A REM statement can only be terminated with a carriage return. All characters, including quotes, commas, and colons may freely appear with the REM statement.

VTAB expr

The VTAB (vertical tab) command is used to vertically position the cursor on the screen. The cursor is moved to the line specified by expr. The top line is one, the bottom line on the screen is line 24. If expr evaluates to a number outside the range 1..24, BASIC will give you an "?ILLEGAL QUANTITY ERROR" message.

HTAB expr

HTAB positions the cursor in the horizontal plane. HTAB allows expr to be any value between 1 and 255. If the value is between 1 and 40, HTAB moves the cursor to the specified column on the current line. If the expression is a value between 41 and 80, HTAB moves the cursor to column 1 through 40 on the next line down. If the expression turns out to be a value between 81 and 120, HTAB moves the cursor to column 1 through 40 on the second line down, etc. If you specify a value greater than 255, HTAB will give you an "?ILLEGAL QUANTITY ERROR". If the parameter to HTAB is zero, HTAB moves the cursor to column 256.

TAB(expr)

TAB can only be used in a PRINT statement. It moves the cursor to the position which is "expr" spaces from the left hand side of the screen, but only if the new cursor position is greater than the current cursor position. If expr evaluates to a column which is less than the current cursor position, TAB leaves the cursor alone.

If expr is a value greater than forty, then TAB moves the cursor to the beginning of the next line, subtracts forty from expr, and tries again. If expr is zero, then TAB treats this value like 256. If expr is greater than 256, BASIC will give you an "?ILLEGAL QUANTITY ERROR". Note that the expression must be surrounded by parentheses. The actual BASIC token for TAB is TAB(. BASIC will not recognize TAB unless it is also followed by a left parenthesis.

POS(expr)

POS returns the current cursor position relative to the left hand margin of the text window. At the left hand column, POS returns zero. The expression passed to the POS function is a dummy expression (like the expression passed to the FRE function). The value of expr isn't used by the POS function, but it must contain a valid exression which returns a number or a string or BASIC will give you a syntax error.

Be aware of the fact that POS (and SPC as you will soon see) uses values starting at zero, while HTAB and VTAB work with values starting at one.

SPC(expr)

SPC prints the number of spaces specified by expr. SPC can only appear within a PRINT statement, if it appears anywhere else you will be given a syntax error. SPC(0) prints no spaces, SPC(1) prints one space, SPC(2) prints two spaces, etc. Up to 255 spaces can be printed with the SPC command.

HOME

HOME clears the screen and moves the cursor to the upper left hand corner of the screen. This command is identical to "CALL -936".

CLEAR

This command clears all the numeric and string variables and it undimensions all arrays. Normally it is used in the immediate execution mode to clear out the variable space, but it can be used in the deferred execution mode if you want to unallocate storage for the variables previously defined. In the deferred execution mode the CLEAR statement can be used to clear out all variables to free up any unused variables and arrays (thus making more memory available to your program).

FRE(expr)

FRE is a function which returns the number of free bytes available. Before returning the amount of free memory, FRE performs a garbage collection operation to compact the string heap. Thus, FRE always returns the true number of bytes available between your arrays and the string heap.

The expression is not used by the FRE function, but it is evaluated so it must be a legal expression or BASIC will give you a syntax error.

FLASH, INVERSE, NORMAL

These commands do not allow any parameters. Subsequent output will be converted to the specified video state. Note that these commands do not affect the status of characters already on the screen nor do they affect characters typed as input. They only affect characters printed to the screen by BASIC.

FLASH sets the video mode to flashing. All further output will be sent to the screen using the flashing character set.

INVERSE switches the video mode to inverse. After the INVERSE command, any output to the screen will be displayed in inverse video.

The NORMAL command returns the output mode to normal.

SPEED=expr

SPEED sets the rate at which characters are printed to the screen. SPEED=0 produces the slowest listing; SPEED=255 produces the fastest listing speed. This command is useful when scanning program listings since the TRACKSTAR board normally lists the program much too fast to read. SPEED=255 is the default value set by TRACKSTAR BASIC.

Expr must be a value in the range 0..255 or BASIC will give you an "?ILLEGAL QUANTITY ERROR" message. The equals sign must be the first non-blank character following the SPEED for BASIC to recognize SPEED= as a reserved word.

Chapter Four: Arrays and Strings

DIM

LEN

STR$

VAL

CHR$

ASC

LEFT$

RIGHT$

MID$

STORE and RECALL

Chapter Four:Arrays and Strings

DIM var(sub {,sub}) {,var(sub {,sub})}

Note: the items in the braces are optional and may be repeated as often as necessary.

DIM, when executed in a BASIC program, allocates storage for an array. DIM can be used to allocate storage for a real, integer, or string array. To allocate storage for an integer or string array, simply insert the "%" or "$" type-definition character between the variable name and the parenthesis in the DIM statement.

BASIC's subscripts always range from zero to the maximum subscript defined in the DIM statement. For example, DIM A(5) produces an array with six elements, i.e., A(0) through A(5).

The maximum number of dimensions supported by BASIC is 88. In practice, however, you will never be able to declare an array with this many dimensions since you would run out of memory long before the limitation of 88 dimensions is reached. An array requires five bytes, plus two bytes for each dimension and five more bytes for each real element, two bytes for each integer element, or three bytes for each string array element. The number of elements can be computed by multiplying each of the maximum subscripts plus one together. E.g., an array dimensioned as A(5,4,2) requires 6*5*3 or 90 elements.

If a variable with a subscript is encountered before the variable is DIMensioned with the DIM statement, BASIC automatically allocates eleven elements (subscripts zero through ten) for the array. This auto-dimensioning feature should be avoided since it makes programs harder to read and maintain. You should always use the DIM statement to dimension an array.

If you attempt to DIMension a variable which has already been dimensioned (either through the use of the DIM statement or via the auto-dimension mechanism), BASIC will print the error "?REDIM'D ARRAY ERROR"

Using an array variable with the incorrect number of subscripts, or with a subscript which is less than zero or greater than the maximum value allowed for that subscript produces a "?BAD SUBSCRIPT ERROR" message.

The actual string data associated with a string array is not actually stored as part of the array. Each element of a string array points at the character data in the string heap space. Therefore, elements of a string array actually consume three bytes plus one byte for each character in the string.

All arrays are undimensioned when the RUN or CLEAR commands are executed.

LEN(sexpr)

The LEN function returns the length of the string passed as a parameter to the LEN function. If the parameter is not a string expression, BASIC will print a "?TYPE MISMATCH ERROR" message.

If the parameter is a concatenation of strings and the resulting string is longer than 255 characters, BASIC will print the "?STRING TOO LONG" error message.

STR$(expr)

STR$ takes the parameter, which must be a numeric expression, and converts it to a string of characters. STR$ produces a string which contains the same characters as though the expression were printed with the PRINT command. I.e., PRINT STR$(VALUE) produces the same output as PRINT VALUE. STR$(100 000 000 000) produces the string "1E+11". If the expression evaluates to a number outside the legal range for real numbers, BASIC aborts processing and an "?OVERFLOW ERROR" message is displayed.

VAL(sexpr)

The VAL function tries to evaluate the string expression parameter as a floating point number and returns that value as the function result.

The first non-blank character in the string must be an acceptable character for an integer or floating point number, or VAL will return zero. If the first non-blank character is an acceptable character for a number, VAL will convert the string, up to the first non-allowable character, to a number and this number is returned by the VAL function.

If the string expression consists of two or more strings concatenated together which produce a string longer than 255 characters, BASIC will print a "?STRING TOO LONG ERROR" message. If the value converted is greater than 1E38, or if it contains more than 38 digits, BASIC prints an "?OVERFLOW ERROR" message.

CHR$(expr)

This function returns a character string comprised of the single character whose ASCII code is equal to the value of the expression passed as the parameter. The parameter must be a value in the range 0..255 or BASIC will print an "?ILLEGAL QUANTITY ERROR" message. If the expression is a real value with a fractional part, the fractional part is truncated and the value is converted to an integer.

Note that strings created with the CHR$(128) through CHR$(255) calls print the same character on the screen as the comparable CHR$(0)..CHR$(127) calls. However, strings created with values in the range 128 through 255 are not equal to strings created using CHR$ in the range 0 through 127. I.e., the expression "CHR$(0) = CHR$(128)" always returns zero. This trick can be used to display certain strings but flag them so that they cannot be compared to an equivalent string entered from the keyboard. For example, the following code sets the H.O. bit of all of the characters in the string S$:

```
10 X$=""
20 FOR I=1 TO LEN(S$)
30 X$=X$+CHR$( ASC(MID$(S$,I,1))+128)
40 NEXT I
```

ASC(sexpr)

ASC returns the numeric value corresponding to the ASCII code of the first character in the string expression passed as a parameter to the ASC function.

If an empty string is passed as a parameter to the ASC function, BASIC prints an "?ILLEGAL QUANTITY ERROR" and aborts further processing. If a string literal is used and the first character of the string literal is CTRL-@, then BASIC aborts with a "?SYNTAX ERROR" message.

LEFT$(sexpr,expr)

This function returns a string composed of the first leftmost "expr" characters out of the "sexpr" string. The first parameter must be a string expression and the second parameter must be a numeric parameter or BASIC will abort with a "?TYPE MISMATCH ERROR" message.

If expr is greater than the length of the string specified by sexpr, then the entire string is returned and the excess character positions specified by expr are ignored. If expr is less than one or greater than 255, BASIC aborts with an "?ILLEGAL QUANTITY ERROR" message.

RIGHT$(sexpr,expr)

RIGHT$ is a function which returns the rightmost "expr" characters from the string specified by sexpr. The first parameter, sexpr, must evaluate to a string and the second parameter, expr, must evaluate to a numeric quantity or BASIC will abort with a "?TYPE MISMATCH ERROR" message. If expr is greater than the length of the string specified by sexpr, then RIGHT$ returns the entire string. If expr is less than one or greater than 255, BASIC aborts processing and the message "?ILLEGAL QUANTITY ERROR" is displayed.

MID$(sexpr, exprl [, expr2])

Note: expr2 is optional.

MID$ returns a substring consisting of the string formed by indexing into sexpr "exprl" characters and of length expr2. If expr2 is not specified, the substring consists of all of the characters in sexpr beginning with the exprlth character through the end of the sexpr string.

If exprl is greater than the length of sexpr, then MID$ returns the empty string. If exprl+expr2 is greater than the length of sexpr and exprl is less than the length of sexpr, then MID$ treats expr2 as though it were not included in the parameter list; i.e., MID$ returns all of the characters from the exprlth position to the end of the string.

sexpr must be a string expression and exprl and expr2 must be numeric values or BASIC will print a "?TYPE MISMATCH ERROR" message. The exprl and expr2 values must be in the range 0 through 255 or BASIC will print an "?ILLEGAL QUANTITY ERROR" message.

STORE and RECALL

These commands were used by the original Applesoft BASIC program to store and load array values to cassette tape. Since the TRACKSTAR board does not support cassette I/O, these commands will cause the system to hang. Use the TRACKSTAR reset sequence to recover if STORE or RECALL is executed.

Chapter Five: Input/Output Commands

INPUT

GET

DATA

READ

RESTORE

PRINT

IN#

PR#

Chapter Five: Input/Output Commands

INPUT {"string";} var {[,var]}

Note: the "string"; part is optional. Additional variable names are also optional, and you can repeat the additional variable names as much as you desire.

If the optional string is present, BASIC prints the string as a prompt before requesting any input. If the string is not present, BASIC simply prints a question mark.

The INPUT statement reads a line of text from the keyboard. If a variable name in the input list is a numeric variable, BASIC will accept only an integer or floating point value. The only non-numeric characters allowed in a numeric value are "+", "-", ".", "E", and spaces. If the first non-blank character is not a real number, an integer, a comma, or a colon, BASIC prints "?REENTER" and prompts you to reenter the value.

If the input list variable is the name of a string variable, the INPUT statement reads the characters entered at the keyboard and assigns them to the string variable. If the first character encountered is a quotation mark, then BASIC reads all characters until a closing quotation mark is encountered. If the first non-blank character is not a quotation mark, then BASIC reads characters until a comma, colon, or carriage return is encountered.

If you simply press return when INPUT is attempting to read an numeric quantity, BASIC will prompt you to reenter the data. If the INPUT statement is reading data into a string, however, the string variable is assigned the empty string and processing continues.

Successive variables in the input list are assigned successively typed values. You can mix string variables and numeric variables in the same input list, but for interactive input from the keyboard this technique is not recommended because it makes the program harder to use.

If a colon is encountered on an input line which does not begin with a quotation mark, all subsequent characters in the input line are ignored by BASIC. The only way to enter the values for any remaining variables in the input list in this case is to press return and enter the additional values.

If the carriage return is encountered before all of the items in the INPUT list have been processed, then BASIC will prompt you to enter the additional data required by the input list. If you enter more items than the input list expects, then BASIC prints an "?EXTRA IGNORED" message and continues execution with the next statement after the input statement.

CTRL-C can be used to terminate program execution during the execution of an input statement if the CTRL-C is encountered as the first character of the input line. Since INPUT always reads a complete line of text, the CTRL-C character will not be recognized until the return key is pressed. If CTRL-C is encountered on the line in a character position other than the first character on the line, BASIC will treat the CTRL-C like any other character.

The INPUT statement can only be used in the deferred execution mode. Attempting to use INPUT in the immediate mode produces the "?ILLEGAL DIRECT ERROR" message.

GET var

GET reads a single character from the keyboard without displaying that character on the screen and without requiring you to press the return key to end the input.

Normally, GET is used to read a character into a string variable. It is possible, however, to GET a value into a numeric variable as well. However, it is always much safer to read the numeric character into a string variable and use the VAL function to convert this character into a numeric value.

GET can only be used in the deferred execution mode. If you attempt to execute the GET command from the immediate execution mode, you will get an "?ILLEGAL DIRECT ERROR" message.

DATA data

The DATA statement is used to store data which is read by the BASIC READ statement (see below). Almost anything that is valid for a BASIC INPUT statement is valid for a DATA statement. There are, however, a few exceptions. First of all, unless they are encountered within a quoted string, colons are recognized as end of line characters. Also, blanks at the end of some string data not enclosed by quotation marks are ignored. See INPUT for more information about the data format.

The DATA statements within a program are treated a bit differently from the rest of the program. First of all, whenever BASIC encounters a DATA statement during the execution of the program, it simply ignores it; the DATA statement is treated like the REM statement. Whenever a READ is executed for the first time in an executing BASIC program, BASIC searches for the first DATA statement in the program by starting at the first line of the program and searching forwards for a DATA statement. When the first DATA statement is encountered, the READ command will read its data from the DATA statement. If the READ statement does not read all of the data present in the DATA statement, then the next READ statement picks up where the current one left off. If a READ statement requires more input than is provided in a DATA statement, BASIC searches sequentially through memory until it finds the next DATA statement. The READ statement continues reading data at that point.

READ var {,var}

Note: the var within the braces is optional. It can be repeated as often as necessary.

READ reads data for the variables in the read list from data stored in the data list (the data list is composed of all the data created by appending all the DATA statements in the program together). READ operates identically to INPUT except that the data is read from the DATA statement rather than the keyboard, and CTRL-C, when encountered in the data list, does not cause the program to abort.

If you attempt to READ more data than there are items in the data list, BASIC will abort and give you an "?OUT OF DATA ERROR IN nnnn" error message, where "nnnn" is the line number of the READ statement where the error occurred.

You can use READ in the immediate mode, but it can only be used to read data from DATA statements within the program stored in memory. If there are no DATA statements in the program currently in memory (or if there is no program stored in memory), then BASIC will give you an "?OUT OF DATA ERROR" message.

Unlike the INPUT statement, it's ok to read only a portion of the data in the data list. Unread data, even in the same DATA statement, will be used the next time the READ command is encountered.

RESTORE

RESTORE is used to reset the data list pointer back to the beginning of the program. The next READ command executed after RESTORE is encountered will read its data from the first DATA statement in the program. RESTORE is used when you wish to reinitialize some variables to the original default values stored in the display list. RESTORE can be executed as many times as necessary within your program.

PRINT print list

PRINT is used to output data to the current output device (usually the video screen). The print list consists of any number of expressions, the TAB function, or the SPC function separated by commas or semicolons. If the expression is a numeric expression, BASIC outputs the number using the floating point output format (see Chapter One). If the expression is a string expression, BASIC outputs the characters which make up the string.

The print list is optional; if it is empty, BASIC simply outputs a carriage return and a line feed and continues execution with the next statement.

If the items in the print list are separated by semicolons, then the output of the various expressions is concatenated together with no interleaving blanks. If the items in the print list are separated by commas, then each item in the output list (which appears after a comma) is printed in the next tab field. The first tab field begins in column one,

the second tab field begins in column 17, the third tab field begins in column 33, the fourth tab field starts over again in column one on the next line, the fifth tab field is on the next line in column 17, etc.

If the print list is present and it is not terminated with a comma or a line feed, then a return line feed pair are emitted after all of the expressions in the print list are output.

If the print list is terminated with a semicolon, then BASIC outputs the items in the print list, but nothing else is output. Any additional output performed by your program will be concatenated on to the end of the output produced by a PRINT statement terminated with a semicolon.

If a comma appears as the last item in the print list, the cursor is moved to the next tab field before the PRINT statement terminates. Subsequent output will be sent to the beginning of the next tab field.

If the text window is set to less than 33 characters (with a POKE 33,nn command), then the third tab field will not work properly. In fact, HTAB can also cause problems if you attempt to move the cursor to a location outside the current text window.

Items separated by only a blank in the output list will be handled as though they were separated by a semicolon if BASIC cannot treat the concatenated items as any other type of construct. For example, PRINT 1.1 0.1 will properly print 1.10.1 because two decimal points are not allowed in a number (actually, BASIC treats this example like PRINT 1.10 .1, but the output is the same so you wouldn't be able to tell the difference between the two).

IN# expr

IN# expr modifies the TRACKSTAR's input hook so that the following characters read via INPUT and immediate read line operations come from the peripheral card in slot "expr". Unlike the Apple II, the TRACKSTAR board does not support physical slots. Therefore the TRACKSTAR board has defined several virtual slots with virtual I/O devices corresponding to popular devices plugged into an Apple II. Consult the appendix for a list of the peripheral devices supported by the TRACKSTAR board.

IN#0 selects the TRACKSTAR keyboard. It is used to turn off the most recently activated peripheral device activated with the previously executed IN#expr instruction.

NOTE: when running BASIC under TRACKSTAR DOS, the IN# command should only be executed in the immediate execution mode. Consult the TRACKSTAR DOS manual for details on executing the IN# command in the deferred execution mode.

PR# expr

The PR# expr command turns on the peripheral device in slot "expr". All output normally sent to the screen will be sent, instead, to the peripheral device in slot expr. PR#0 turns off any active peripheral devices and resumes video screen output.

Like the IN# command, the PR# command should only be executed in the immediate execution mode when operating under TRACKSTAR DOS. Consult the TRACKSTAR DOS manual for details about how to execute the PR# command from the deferred execution mode.

WARNING: IN# and PR# commands which specify an expression corresponding to a non-existent slot or to a slot not containing an active virtual peripheral device may cause BASIC to hang or perform strangely. Slot numbers in the range 8 through 255 will also cause problems since the standard Apple II does not support more than seven slots. Values outside the range 0 through 255 will produce an "?ILLEGAL QUANTITY ERROR". TRACKSTAR DOS users should consult the TRACKSTAR DOS manual for more details.

Chapter Six: Statements Which Control Program Execution

GOTO

IF..THEN and IF..GOTO

FOR..TO..STEP

NEXT

GOSUB

RETURN

POP

ON..GOTO

ON..GOSUB

ONERR..GOTO

RESUME

Chapter Six: Statements Which Control Program Executio

GOTO lineno

GOTO is used to unconditionally transfer control to another part of the program. GOTO must be followed by a valid line number which is present in the program. If you attempt to branch to a non-existent line in the program, BASIC will give you an "?UNDEF'D STATEMENT ERROR IN lineno" error message.

Note that lineno must be an actual integer number, not an expression. GOTO stops parsing the line number when the first non-digit is encountered. If the line number begins with a non-digit (e.g., if you try to use a variable name instead of a line number), BASIC will jump to line zero (if it exists).

If a GOTO statement appears on a line, BASIC ignores all other statements on the same line as the GOTO except the DATA statement. Therefore, you should never place executable statements after a GOTO statement on the same logical line. BASIC would never be able to execute them and they would simply be wasting memory.

If the GOTO statement is executed in the immediate mode, it behaves exactly like the RUN lineno command except that BASIC does not clear out the variable space before program execution begins.

IF expr THEN statement {:statement}
IF expr THEN lineno
IF expr GOTO lineno

Note: the statement within the braces is optional and can be repeated as many times as necessary.

The BASIC IF statement is used to conditionally execute another BASIC command. The expression between the IF and the THEN (or GOTO) is evaluated. If the value of the expression is zero (or if its absolute value is less than roughly 2.93873E-39), then BASIC continues execution with the next line of code in the program. BASIC does not execute any remaining statements on the same line as the IF statement. Instead, it drops down to the next numbered line of BASIC code and continues execution there.

If the expression in the IF statement evaluates to a non-zero value, then the action of the IF statement depends on which version of the IF statement you are using. The last two forms, IF..THEN lineno and IF..GOTO lineno, transfer control to the specified line number if the expression is non-zero. Any remaining statements on the line (except a DATA statement) are ignored by the BASIC interpreter. Like the straight GOTO statement, such statements would simply be wasting space.

If an IF statement of the form IF expr THEN statement {:statement} is encountered and the expression evaluates to non-zero (true), then BASIC executes the remaining statements on the same line as the IF statement. Note that additional BASIC statements on the same line as the IF statement, but separated from the IF statement by colons, can only be executed if the expression is true. If the expression in the IF statement is false (zero), the BASIC skips over the remaining statements on the line, including those separated by colons, to the beginning of the next line in the file that begins with a line number.

If THEN is encountered in the program without a corresponding IF, BASIC will print a "?SYNTAX ERROR" message and abort processing.

BASIC expects an arithmetic expression between the IF and the THEN reserved words. However, an error is not printed if a string expression appears between the IF and THEN tokens. In such a case the expression almost always evaluates to true; however, there are some bizarre cases where it evaluates to false. You should avoid placing string expressions between the IF and THEN symbols.

Although string expressions shouldn't be used between the IF and THEN reserved words, arithmetic expressions involving string sub-expressions are perfectly legitimate. For example, the statement "IF A$ < B$ THEN C$=A$:A$=B$:B$=C$" swaps the strings A$ and B$ if the string A$ is less than B$. Although the expression involves strings, it is still an arithmetic expression since the relational operator "<" returns zero or one.

If the letter A appears before the THEN reserved word, BASIC has problems parsing the line. For example, if you enter:

```
IF LTRA THEN 400
```

BASIC tries to interpret this as:

```
IF LTR AT HEN400
```

which produces a syntax error when the statement is executed. This problem arises since BASIC uses a bottom up parsing technique and AT is also a BASIC reserved word.

FOR var = expr1 TO expr2 {STEP expr3}

FOR sets the variable specified by var to the value of expr1 and executes the following BASIC statements until a NEXT statement is encountered. When the NEXT is encountered, expr3 (if the optional STEP portion is included, expr3 is equal to one if it is not present) is added to var and the resulting value is compared to expr2. If the STEP portion was not specified or expr3 is positive, var is compared to see if it is greater than expr2. If it is not, then the lines of BASIC code between the FOR and the NEXT are repeated until var is greater than expr3. If expr3 is negative, then var is compared to expr2, and the lines of code between the FOR statement and the NEXT statement are repeated if and only if var is greater than or equal to expr2.

The variable name specified for the loop control variable (var in the definition above) must be a real variable. Integer and string variables are not allowed.

FOR loops may be nested up to ten levels. If you attempt to nest a FOR loop beyond the tenth level, BASIC will give you an "?OUT OF MEMORY ERROR" message. Nested FOR loops must be matched with corresponding NEXT statements or a "?NEXT WITHOUT FOR ERROR" message will be printed. For more details, see the next section.

If you execute a FOR..NEXT loop in the immediate execution mode, the NEXT statement must be present on the same line as the FOR statement or the FOR statement may not behave properly.

NEXT {var [,var]}

Note: the items inside the braces are optional. The variable name inside the square brackets can be repeated zero or more times as desired.

The NEXT command is used to terminate a FOR..NEXT loop. There are three forms of the NEXT statement. If NEXT appears by itself, then the NEXT statement terminates the most recently activated FOR loop. This form of the NEXT statement executes the fastest and requires the least amount of memory to define. However, good programming practice dictates that you should avoid this form of the NEXT statement since your programs will be harder to debug if you extensively use this form.

The second form of the NEXT statement is "NEXT var" where var is the name of the loop control variable specified in the FOR statement. All nested FOR loops between the FOR loop which defines the loop control variable specified by var are immediately terminated regardless of whatever value their loop control value happens to be. If there is no FOR loop which contains the loop control variable specified by the NEXT statement, BASIC prints a "?NEXT WITHOUT FOR ERROR" message and aborts.

The last form of the NEXT statement is actually a shorthand form of the previous version. A NEXT statement of the form "NEXT var1,var2,..." is identical to the sequence of statements "NEXT var1: NEXT var2:...". Note that the loop control variables must be specified in the reverse order they were encountered as FOR loop control variables.

GOSUB lineno

GOSUB transfers control to the line number specified after the GOSUB reserved word, and it saves a pointer to the next statement following the GOSUB statement on the BASIC stack. Whenever a BASIC RETURN command is executed, BASIC pops the pointer off of the stack and resumes program execution at the statement just past the GOSUB command.

If the line number which appears after the GOSUB command is not present in the BASIC source file, then BASIC prints an "?UNDEF'D STATEMENT ERROR IN lineno" message. If GOSUBs are nested more than 25 levels deep, BASIC prints an "?OUT OF MEMORY ERROR" message.

RETURN

The RETURN command does not allow any parameters. When executed, it pops the last GOSUB return address off of the stack and BASIC continues program execution at the address popped off of the stack (which is the address of the next statement following the GOSUB command).

If BASIC executes a RETURN instruction without a pending GOSUB return address on the stack, BASIC prints a "?RETURN WITHOUT GOSUB ERROR" and aborts processing.

POP

The POP command is used to remove a GOSUB return address from the stack without transferring control to the statement following the GOSUB. If a POP is encountered when there are no GOSUB return addresses on the stack, BASIC prints a "?RETURN WITHOUT GOSUB ERROR" message.

ON expr GOTO lineno {,lineno}

Note: the line number inside the braces can be repeated zero or more times as necessary.

When an ON..GOTO statement is encountered, BASIC evaluates the expression between the ON and GOTO. This expression must evaluate to a numeric quantity in the range 0..255 or BASIC will give you an "?ILLEGAL QUANTITY ERROR".

If the expression evaluates to one, BASIC transfers control to the first line number in the list of line numbers which follow the GOTO token. If the expression evaluates to two, BASIC jumps to the line specified by the second line number after the GOTO. If the expression is three, BASIC jumps to the third line number, etc.

If the expression evaluates to zero, or it evaluates to a number greater than the number of line numbers specified after the GOTO reserved word, BASIC continues program execution with the next statement in the program after the ON..GOTO instruction.

ON expr GOSUB lineno list

ON..GOSUB operates in a fashion identical to ON..GOTO except the subroutine at the specified line number is called rather than a simple transfer of control. When a RETURN is encountered, BASIC returns to the statement immediately past the ON..GOSUB command.

ONERR GOTO lineno

Whenever an error occurs in the deferred execution mode, BASIC normally prints an appropriate error message and aborts further processing. Sometimes you may want to perform error handling within your program to prevent the system from bombing whenever the user of the program enters some incorrect data. By executing the ONERR GOTO command,

a flag is set in the BASIC interpreter. The next time an error occurs, BASIC jumps to the line number specified after the ONERR GOTO statement rather than aborting and printing an error message.

When an error occurs within a program, the error code can be found in memory location 222. By PEEKing at this memory location you can see which type of error occurred. The codes for the various errors are

```
 0 NEXT without FOR
16 Syntax
22 RETURN without GOSUB
42 Out of DATA
53 Illegal quantity
69 Overflow
77 Out of memory
90 Undefined statement
107 Bad subscript
120 Redimensioned Array
133 Division by zero
163 Type mismatch
176 String too long
191 Formula too complex
224 Undefined function
254 Bad response to INPUT statement
255 CTRL-C interrupt attempted.
```

A POKE 216,0 resets the ONERR flag so that further errors will be automatically processed by the BASIC interpreter.

RESUME

RESUME, when used at the end of an error handler routine invoked by ONERR GOTO, resumes execution of the BASIC program at the beginning of the statement where the error occurred.

If RESUME is encountered before an error occurs, BASIC may give you a SYNTAX error or the system may simply hang. If an error occurs in an error handling subroutine, RESUME may cause an infinite loop. You will have to use the TRACKSTAR reset sequence to abort.

RESUME should not be executed in the immediate execution mode since the results are unpredictable.

Chapter Seven: Graphics and Game Controls

TEXT

GR

COLOR

PLOT

HLIN

VLIN

SCRN

HGR

HGR2

HCOLOR

HPLOT

PDL

Chapter Seven: Graphics and Game Controls

TEXT

The TEXT command turns off any graphics mode (LORES or HIRES graphics) and restores the text window to the standard 40x24 screen. TEXT also positions the cursor to the bottom left hand side of the screen.

TEXT does not allow any parameters.

GR

The GR command selects the 40x40 LORES graphics mode. The page one LORES graphics screen is selected by this command. In this mode there are four lines of text at the bottom of the screen. If you wish to select the full screen LORES graphics mode, perform a POKE -16302,0 after you issue the GR command.

Warning: if the GR command is issued while HGR is in effect, the GR command behaves normally. However, if the GR command is issued while the HGR2 command is in effect, the GR command will select the LORES graphics page two screen. To ensure that the LORES graphics page one screen is selected, execute the TEXT command before executing the GR command.

COLOR= expr

This command sets the color which will be used for plotting by the LORES PLOT subroutine. If expr is outside the range 0..255, an "?ILLEGAL QUANTITY ERROR" message is displayed. If expr is outside the range 0..15, the remainder of expr divided by sixteen is used.

The colors corresponding to the various values are

0 black
1 magenta
2 dark blue
3 purple
4 dark green
5 grey
6 medium blue
7 light blue
8 brown
9 orange
10 grey
11 pink
12 green
13 yellow
14 aqua
15 white

COLOR is set to zero by the GR instruction.

PLOT exprx, expry

This command plots a low-resolution dot on the screen at the x,y coordinate (exprx,expry). The x-coordinate value must be in the range 0..39 and the y-coordinate value must lie in the range 0..47 or BASIC will give you an "?ILLEGAL QUANTITY ERROR" message. The point is plotted using the color specified in the last COLOR= command. If COLOR= wasn't executed since the last GR command, the color plotted is black.

Unlike most graphic displays, the graphics displays on the TRACKSTAR board defines the origin, (0,0), to be the upper left hand corner of the screen. The x-coordinate increases in the left to right direction, the y-coordinate increases in the top to bottom direction.

If PLOT is used while in the text mode, it will overwrite a portion of the text screen. Using PLOT in the high-resolution graphics mode produces no visible effect until you turn the text mode back on with the TEXT command.

HLIN expr1, expr2 AT expr3

HLIN is used to draw a horizontal line on the LORES graphics screen. It draws a line from (expr1,expr3) to (expr2,expr3) using the color specified by the last COLOR= command. BASIC will give you an "?ILLEGAL QUANTITY ERROR" message if expr1 or expr2 is outside the range 0..39 or if expr3 is outside the range 0..47.

VLIN expr1, expr2 AT expr3

VLIN plots a LORES graphics point from (expr3,expr1) to (expr3,expr2) on the LORES graphics screen. Expr1 and expr2 must be in the range 0..47, expr3 must be in the range 0..39.

SCRN(exprx,expry)

SCRN returns the color of the LORES graphics point at (exprx,expry) on the screen. The x- and y-coordinates must lie in the valid ranges for LORES points.

HGR and HGR2

HGR selects and clears the page one high-resolution graphics screen. HGR2 selects and clears the page two high resolution graphics screen. HCOLOR is not affected by these commands.

HGR selects the mixed text and graphics mode. In this mode, the HIRES screen is limited to 280x160 with four lines of text at the bottom of the screen. To select full screen graphics, issue the BASIC command POKE -16302,0.

The HGR2 command selects the full screen (280x192) graphics mode. The command POKE -16301,0 can be used to turn on the mixed text and graphics mode. However, the four lines of text displayed at the bottom of the screen come from the page two text screen which is not easily accessible from BASIC.

HCOLOR=expr

HCOLOR=expr sets the high-resolution color which will be used by the HPLOT command. Expr must be a value in the range 0..7.

HPLOT exprx,expry
HPLOT TO exprx,expry
HPLOT expr1,expr2 TO expr3,expr4 {TO exprx,expry}

Note: the items inside the braces are optional and may be repeated as many times as necessary.

The first form of the HPLOT command plots a single point at the X and Y coordinate specified by the two parameters. Exprx must be a value in the range 0 to 279, expry must be a value in the range 0 through 191. The color of the point plotted depends on the color which was last set by the HCOLOR= command. If HCOLOR= was not executed prior to the execution of an HPLOT command, the color plotted is indeterminate.

The second form of the HPLOT command draws a line from the last point plotted (or the endpoint of the last line drawn) to the point specified by (exprx,expry). The line is drawn using the color last set by HCOLOR=.

The last form of the HPLOT command is used to draw lines. HPLOT begins by drawing a line from (expr1,expr2) to (expr3,expr4). If any of the optional (exprx,expry) points are specified, HPLOT continues drawing lines from the last point plotted to (exprx,expry). The command:

```
HPLOT 0,0 TO 279,0 TO 279,159 TO 0,0
```

draws a rectangle around the border of the 280x160 graphics screen.

PDL(expr)

PDL returns a numeric value between 0 and 255 corresponding to the setting of the game controller selected by expr. Expr must be in the range 0..3 or the BASIC may react unpredictably.

When performing two consecutive paddle read operations, allow a slight delay between the reads to give the paddle read circuitry time to reset itself. A short delay like "FOR I=1 TO 10:NEXT" is sufficient.

Item	Page
"?" statement (PRINT)	12
ABS	9
Applesoft error codes	49
Arithmetic operators	8
Array variables	6
ASC	21
ASC	9
ASC	36
Assigning a value to a variable	7
ATN	9
Auto-dimensioning	34
BASIC editing commands	29
BASIC line format	10
BASIC screen editing features	4
Built-in functions	8
CALL	26
CHR$	21
CHR$	10
CHR$	35
CLEAR	31
Clearing BASIC's variable space	31
Clearing program storage	24
Clearing the screen	31
Colons and the INPUT statement	39
COLOR=	51
Comma in a PRINT list	12
Commas at the end of a print list	42
Comments within your BASIC program	30
Concatenation	10
CONT	24
Continuing a stopped program	25
COS	9
CTRL-C	24
CTRL-C	40
DATA	13
DATA	40
Data pointer	13
Data types	19
DEF FN	15
Deferred execution commands	2
DEL	29
Deleting a line	3
Deleting lines from a BASIC program	29
DIM	34
DIM statement	6
Dimensioning an array	34
Drawing a horizontal LORES graphics line	52
Drawing a line in HIRES	53
Drawing a vertical LORES graphics line	52
Dummy arguments	16
Editing a BASIC program	3
END	24
Error handling	48
Error recovery	49

Item	Page
ESC @	4
ESC A	4
ESC B	4
ESC C	4
ESC D	4
ESC E	4
ESC F	4
ESC I	4
ESC J	5
ESC K	5
ESC M	5
Executing a BASIC program	24
EXP	9
FLASH	32
Flashing characters	32
Floating point input format	39
Floating point output format	5
Floating point variables	19
FN «name»	16
FN«name»	16
FOR..NEXT	17
FOR..NEXT	46
FRE	31
FRE	8
Garbage collection	31
Garbage collection	20
GET	13
GET	40
GOSUB	19
GOSUB	47
GOTO	45
GR	51
HCOLOR=	53
HGR	53
HGR2	53
HIMEM	26
HIRES color values	53
HLINE	52
HOME	31
HPLOT	53
HTAB	30
IF..GOTO	16
IF..GOTO	45
IF..THEN	16
IF..THEN	45
Immediate execution commands	2
IN#	42
IN# command	14
Initializing arrays from BASIC	13
INPUT	12
INPUT	39
Input data containing colons	39
Input list	39
INPUT prompt string	13

Item	Page
INPUT prompt string	39
INT	9
Integer arrays	20
Integer variables	19
Integer variables	6
INVERSE	32
Inverse characters	32
LEFT$	22
LEFT$	10
LEFT$	36
LEN	22
LEN	8
LEN	35
LET statement	7
Line numbers	3
LIST	29
LOAD	24
LOG	9
Logical line format	11
LOMEM	27
Loop control variables	47
LORES colors	52
MID$	22
MID$	10
MID$	37
Multiple statements per line	11
Multiple variable names after the NEXT statement	18
Nested FOR loops	18
NEW	24
NEXT	17
NEXT	47
NEXT	46
NORMAL	32
Normal characters	32
NOTRACE	25
Number format	5
Obtaining the current cursor position	31
ON..GOSUB	48
ON..GOTO	48
ONERR GOTO	48
Output format for floating point values	5
Output formatting	12
Outputting spaces	31
Parameters	16
PDL	9
PDL	54
PEEK	25
PEEK	8
PLOT	52
Plotting a HIRES point	53
Plotting a LORES point	52
POKE	26
POP	48
POS	31

Item	Page
POS	8
Positioning the cursor to a specific column on the screen	30
Positioning the cursor to a specific line on the screen	30
PR#	43
PR# command	14
PRINT	11
PRINT	41
Print list	42
Program loops	46
READ	13
READ	41
Reading data from the keyboard	39
Reading the color of a LORES point on the screen	53
Reading the game paddles	54
Real variables	19
RECALL	37
Redirection of I/O	14
Relational operators	8
REM statement	30
Replacing a line	3
Reserving space for a machine language program	26
Reset	24
Reset sequence	25
Restarting a program after an error	49
RESTORE	13
RESTORE	41
RESUME	49
RETURN	19
RETURN	48
RIGHT$	22
RIGHT$	10
RIGHT$	36
RND	9
Rounding floating point values	19
RUN	24
SAVE	24
SCRN	9
Semicolon in a PRINT list	12
Semicolons at the end of a print list	42
Setting the HIRES color value	53
Setting the LORES color value	51
SGN	9
Significant digits	6
SIN	9
Slowing down the display	32
SPC	31
SPEED=	32
SQR	9
Standard input channel	15
Standard output channel	14
STOP	24
Stopping a program during execution	24
STORE	37
STR$	21

Item	Page
STR$	10
STR$	35
String arrays	20
String arrays	34
String variables	19
String variables	6
Subroutine calls	47
Subroutines	19
Subscripts	34
TAB	30
TAN	9
TEXT	51
THEN	16
TRACE	25
Turning off a graphics mode	51
Turning off flashing or inverse characters	32
Turning on LORES graphics	51
Turning on the 40x24 text screen	51
Turning on the LORES graphics display	53
Type suffixes	22
USR	27
USR	8
VAL	21
VAL	10
VAL	10
VAL	35
Variable name length	6
Variable names	6
Virtual slot	14
VLIN	52
VTAB	30
WAIT	26