SPORTSware's SUPER DISK

FOR THE TANDY COLOR COMPUTER 3 128 K DISK SYSTEM

SPORTSware
1251 South Reynolds Road
Suite 414
Toledo, Ohio 43615
(419) 389-1515

TABLE OF CONTENTS

SYSTEM REQUIREMENTS

THE SUPER DISK is for the TANDY COLOR COMPUTER 3 with at least 128K memory and one disk drive. A monitor capable of displaying 80 characters per line is needed for some of the modules and two disk drives are required for the READRITE ,COPYPLUS and DISKCOPY modules. All other modules require only one disk drive.

AN INTRODUCTION TO COPY PROTECTION

Copy protection is one of the hottest issues in the computer world today. On one side of the issue stand the software developers and merchants who have invested enormous amounts of time and money in the creation of computer applications. On the other side are the users who must contend with assorted schemes to prevent them from copying and distributing software to each other at no cost. One unpleasant effect of protection schemes is that not only can the original purchaser not give it away but in some cases cannot even protect himself from damage to the original disk by having working copies available for himself.

A small industry has been born from this dilemma, again on both sides of the fence. On one hand are the schemers who devise these dastardly protection techniques and on the other the devilish undoers who devise and sell applications to defeat protection schemes. As in any conflict, it always seems to be the arms merchants who profit. Everyone else loses, ie. higher development cost and higher price.

In the beginning, way back when microcomputers first started to take their place in the business world, copy protection made it's debut. Copy protection was rampant and understandably so. Those applications usually cost hundreds of dollars. As time passed, developers and distributors got so much flak about it that a compromise was reached, at least in the world of corporate giants. Copy protection would be dropped if companies who purchased the products adopted a written anti-piracy policy. This all came to pass and today most commercial applications are not copy protected. The personal computer world is different.

In the world of personal or home computers things are different. How are you going to get a few million personal computer users to sign an anti-piracy agreement? What can you do to them if they violate it? Fire them? Now we're talking primarily about the entertainment software market. Unfortunately, most computer users seem to have the same attitude about copying software as they do about cheating on taxes. What's wrong with giving good old Al a copy of this great game I just bought? Heck, I paid for it. It's mine now. Besides, what would Al think of me if I told him no?

Granted, most users don't go buy a program just so they can give copies away. Most of us are honest aren't we? Next time your friend shows you a new program, ask him to give you a copy. If he tells you no, it's wrong to do that, go buy your own copy, send me his name. I've always wanted to meet an honest man.

Maybe that's a little severe. But here's the crux of the issue. If you spend untold amounts of time and money developing a piece of software, even more on promotion and distribution, not to mention the associated risk involved in such ventures, do you or don't you have the right to protect your investment and in fact profit from it? One company did some research to try to determine how many illegal copies of it's major product were in circulation. They came up with a figure of five for every one sold legally. This is the kind of damage that can even destroy a company. And what incentive does this give them to take the risks involved in creating more new products?

As long as there is piracy, we will have software prices that are higher than they might otherwise be and copy-protection.

COPY PROTECTION TECHNIQUES

Copy-protection schemes have varied widely over the years. But most use what is referred to as a signature--special marks or oddly formatted sectors on the disk, and a checker routine that looks for the signature. There are also non-disk type protection schemes too. We'll look at several of these techniques.

One of the first was PROlock, produced by the Vault Corp. In this technique, one or several physical holes were burned into the disk in precise locations. On booting the disk, the checker routine would go look for a defect or defects in the disk in the precise locations. On finding them, the program would be allowed to proceed. If they were not found, the program would terminate in one fashion or another. This method was eventually abandoned by the big commercial applications in the U.S. but is still in use in some countries and possibly in the entertainment market.

SUPERlok has taken the place of PROlock as the current favorite. It uses a signature, but not a physical one. SUPERlok places data on a disk in a place the floppy disk controller doesn't normally expect to see data, in the gaps between sectors. Other SUPERlok techniques include overlapping sectors and sectors of unusual size. If the checker routine does not find the required signature, it knows the disk is a copy and terminates the program. Since the standard floppy disk controller cannot write to these locations or duplicate the format, the standard backup or copy commands fail to create a working copy.

In fact, most copy-protection schemes rely on the limitations of the floppy disk controller by placing strange data in the sector gaps, altering normal track and sector IDs, stretching the gaps between sectors, or changing the sync bytes, address marks or clock bits. The disk controller may be able to recognize that something unusual has been placed on the disk but it cannot duplicate it.

You can see that with all these variables to work with, the copy-protection author can create an almost endless number of different schemes just with the diskette. There are other non-disk methods too.

Nocopi International Inc. of Toronto makes and sells Nocopi paper. This is a dark purple paper that makes it virtually impossible to copy information printed on it using standard copy machines or fax machines. This technique is being used with computer games. Some critical information is printed on the sheet that must be referred to at various points in the game. Without the correct information the game cannot be continued. This method eliminates the need to use disk related methods but

is restricted to strategy or adventure type games.
Even though the information cannot be photocopied or faxed, what's to prevent the ardent pirate with more time than money from hand copying the information? Also, some feedback from the end user suggests that information printed on Nocopi paper is difficult to read even under ideal lighting conditions, thus something of an irritation.

Another non-disk method requires the user to have the program manual in order to use the application. At various points in the application, the user is asked to enter a word located in a precise place in the manual. For example, when the application boots up the user is asked to enter the fourth word in the third paragraph on page thirty. Failure to enter the correct word will terminate the program. This process can be repeated as the user enters different phases of the application. This method too has limitations. If the manual is brief, a few minutes at the local copy center will defeat this method. Also, it seems limited to non-action type games and productivity software. The most determined pirate would probably be willing to pay the price to copy even an extensive manual rather than pay for the program.

The wheel method is yet another non-disk protection scheme. In this method, as the user progresses through the game, he is occasionally asked to enter a code number which can only be obtained from a wheel that came with the game. This wheel might consist of three progressively smaller circular pieces of stiff paper riveted together in the center. On the largest outer wheel are several words printed around the outer edge. On the center wheel are as many as several hundred code numbers printed in a circular manner. On the inner wheel are several holes, or windows, with names printed below each of them. At various points in the program, the user is asked to enter the code number in one of the windows when the wheel is aligned in a certain manner. If the wrong code is entered, the program terminates. This method is effective only if the user does not have a wheel. Unfortunately for the software merchant, these wheels are not too difficult to duplicate, again, a quick trip to the copy center, removal of the rivet, and a little cutting with a utility knife will do the trick. The only bad part is that the original purchaser has to be willing to ruin his paid for wheel so old Al can have a free copy.

As you can see, most non-disk copy-protection schemes end up being more of a nuisance to the user than an effective method of copy protection. Besides, they are highly visible to the user, a constant reminder that he is not trusted. Worst of all, they are not effective. Each can be overcome with no technical knowledge required, usually just a little time and a copy machine will do the trick.

Disk based copy-protection methods have much more to offer the software merchants. They usually operate unseen and unheard in the background and require a great deal of disk savvy to defeat. They can be very simple to extremely complex and only take up disk space that wasn't being used anyway. They are effective too. Usually specific applications are required to defeat them or create exact copies of them which end up being protected too.

Many software companies have gone the one-two punch route today. They use disk based protection and non-disk based disguised as part of the game. For example the game disk is protected perhaps allowing a copy to be made that will not function but can be used to restore a damaged original disk. Then, in the manual, there is certain information critical to completing the game.

Hardware based protection is yet another option. All of us are familiar with Rom Packs, sometimes referred to as program packs. This method is used primarily for game machines. One disadvantage for users with disk based systems is that the disk controller must be unplugged and the rom pack plugged in each time the application is used. Before disk drives became readily affordable, even serious applications like word processors and spreadsheets came in rom packs. Data files were saved to and retrieved from cassette tapes. Many rom packs hold as much as 32K of data today. Even these can be copied however. By covering the pin which causes the computer to recognize and execute the rom program, one can save the data to tape then load and save it to disk.

Another hardware based protection scheme is also used. Some applications come with a special hardware device such as a joystick adapter or serial device that must be plugged in. This device comes with the original software. Without it plugged in the software will not function properly so there is no need to add a protection scheme to the disk itself. This method is a classic case of protection that forces the cost of software up higher than it could be. Most companies do not use this method for that reason.

THE SUPER DISK

The SUPER DISK (SD) contains many separate applications designed to allow you, the programmer, to create your own unique copy protection system. Besides these applications you will find several disk utilities. These will allow you to do many things that would have cost you well over a hundred dollars had you purchased these utilities separately. It is assumed that you have a good understanding of the BASIC programing language and are familiar to some extent with ASSEMBLY language. In the appendix at the end of the manual are several program listings. Some of these are in assembly language and for you to use them effectively will require simple modification on your part. Full instructions are included in the appropriate section on each of these listings. None of these listed programs are copy-protected to make it as easy as possible for you to modify and use them.

There are other programs on the disk which require no programing knowledge to use. These will allow you to add some protection to diskettes very easily and to learn more about copy-protection by examining methods used by others. We think you will find THE SNOOPER extremely powerful and useful not only for protection but for general disk management and repair.

The next section of this manual is dedicated to familiarizing you with the disk operating system and diskette format. Without a good understanding of the operating system and it's limitations you cannot devise effective copy-protection techniques. The same applies to disk formats.

After that we'll take a detailed look at each of the applications on the disk and go through some examples of how to use each one and get the most from it.

Finally, we'll look at some different copy-protection techniques and make some copy-protected disks using these methods and the applications on the SUPER DISK.

You will find, after becoming familiar with the applications on the SUPER DISK, that copy-protection is a fascinating subject where your imagination is the only limitation. Examining and trying to defeat protection schemes may be one of the best puzzles ever created by man (as long as you are doing it for the sport of it). We've certainly enjoyed putting this package together and spent many hours studying the subject. There's nothing like that "AH HA!" when you find the secret code or hidden track unseen by the regular operating system. After all, we are dealing primarily with games aren't we.

Then, there is the serious side of it. As a programmer you want to protect your baby. Make it as difficult as possible for the pirates to steal even one copy, right? When you get to the point that you can say to yourself "There, if they can beat this, they deserve a free copy!" you probably are right.

But remember, anything that can be done can be undone.

DISK BASED SYSTEMS

In order to have a good understanding of copy-protection techniques, you must have a thorough understanding of floppy disk computer systems. The three key elements involved are floppy diskettes, disk drives including the controller and DOS which is short for Disk Operating System. We'll look at each of these in turn. Additional reference material you may want to read is available from Tandy Corp., Western Digital and Microsoft.

FLOPPY DISKETTES

The Color Computer Disk System Programming Manual that came with your disk drive is an excellent source of basic knowledge about your disk system and diskettes. We suggest you blow the dust off of it and spend a little time browsing through it.

The Coco operating system uses diskettes formatted with 35 tracks each consisting of 18 sectors each of which hold 256 bytes of data. THIS IS THE KEY LIMITATION OF THE OPERATING SYSTEM.

The truth of the matter is that the drives supplied by Tandy for use with the Coco will actually format 40 tracks. The number of sectors per track can be varied. And the manner in which the diskette is formatted can also be varied (the information on the disk between the sectors and tracks and sector length).

When a diskette is formatted normally (using the DSKINI command), each track looks like this;

Bytes	0000-0031	$4E	this is the leading gap (32 bytes)
E	0000-0007	$00	this is a sync field (8 bytes)
a	0008-0010	$F5	sets CRC bytes to 1s (3 bytes)
c	0011	$FE	the address ID mark (points to info below)
h	0012	( )	track number is inserted here
	0013	( )	side number (always zero w/RSDOS)
s	0014	( )	sector number is inserted here
e	0015	$01	sector size 0=128 1=256 2=512 3=1024 bytes
c	0016-0017	$F7	CRC request (Cyclic redundancy check)
t	0018-0039	$4E	this is the post ID gap (22 bytes)
o	0040-0051	$00	second sync field (12 bytes)
r	0052-0054	$F5	reset CRC bytes (3 bytes)
	0055	$FB	this byte marks the start of data field
	0056-0311	$FF	data field is filled w/$FF (256 bytes)
	0312-0313	$F7	another CRC request
	0314-0337	$4E	this is the post data gap (24 bytes)
		$4E	end of track gap (200 bytes)

When you format a diskette with the DSKINI command, above is exactly what information will be placed between each track and sector. (You can see this for yourself by formatting a disk and then booting up the utility called READTRAK, try it!)

The heart of most copy-protection schemes is to CHANGE some of the above information and then have the checker routine look to see if the disk in the drive has the correct (changed) information on it. If any of it is not correct, the checker decides that the disk is a phony and terminates the program.

Let's look at each field and see how it can be used for copy protection purposes.

THE LEADING GAP which normally consists of 32 bytes can be changed to a different length. This can be done from BASIC by poking the length into location $D695. The checker could then load an entire track into memory as the program first executes and count the number of bytes it sees before it encounters the sync field. (There are better ways)

THE SYNC FIELD is there to allow the disk controller to keep on track, not get too far ahead or behind. We do not recommend changing the length of these fields. They must be zeros also.

The three $F5s that follow the sync fields reset the CRC counter to all ones. These should not be changed either.

The code **$FE** is the address ID mark. This mark tells the controller that the next four bytes are the track, side, sector and sector size numbers. This byte should not be changed.

Here is where the fun begins. The track number, side, sector number and sector size can be and usually are changed for copy protection purposes. Track numbers can legally be from 0-255. The side number can be any value from 0-255. Sector numbers can be any number from 1-255. Sector size (the length of the data field can be 128, 256, 512 or 1024 bytes long. As the size of the data field increases, the number of tracks on a disk must be decreased correspondingly. Let's look at each of these ID marks individually to see what can be done with them.

TRACK NUMBER: This byte identifies the track. Normally, this number ranges from 0-34 giving us 35 tracks. Actually, the number inserted here can be anything from 0-255. Let's say we have a program we want to protect. We format tracks 0-34 normally. But when we get to track 35 we change the number to 200. On this track we place one sector of information or program code we'll have the checker routine look for.

Suppose an enterprising would be pirate suspects that there are more than 35 tracks. He formats a disk with 40 tracks and uses the BACKUP command to copy our disk. What happens? It doesn't work for him because as far as DOS can see, there is no track 35-39. His copy fails to work.

SIDE NUMBER: Tandy DOS only recognizes side zero. We can use this byte as a code number for our checker to look for.

SECTOR NUMBER: This number can be from 1-255. In fact you could actually build a track that really has 255 sectors. You would end up with an un-copyable disk. Let's look at our example again.

Suppose besides numbering track 35 as track 200 we also play some games with the sector numbers. Let's number all of the sectors as 36 except the one we actually have data we want to use. Let's number that one 20. Again the normal BACKUP and COPY commands will fail to detect our secret track and sectors.

SECTOR SIZE: The four legal sector sizes are 128, 256, 512 and 1024 bytes. These are set during the format process by the value sent to the controller in this byte. 0=128, 1=256, 2=512 and 3=1024 bytes per sector. If this number is not 1, the normally formatted (256 bytes per sector) disk will fail to copy the data. So besides changing the track number, side number and sector number, we can even change the size of the sectors on our secret track.

You can see already the countless number of variations available. These schemes are not easy to duplicate either without specialized software. The average would be pirate without special software and/or a lot of technical savvy will soon give up trying to copy our disk.

The $F7 byte tells the controller to calculate the CRC value. This value is calculated with the following formula. G(x)=x16+x12+x6+1. The CRC includes all information from the address mark up to the CRC characters. The CRC is used by the system to verify data integrity. We can use it to verify that none of our program has been changed since any change to the data will change the CRCs.

After the CRC is a gap called the post ID gap which is 22 bytes long. This gap can be varied in length and checked as a copy protection scheme also. The $4E can be changed to another value but data in the gaps is often misread by the controller because of getting out of sync. The actual data value does not present us with a totally reliable method.

Another sync field follows this gap. It's length is 12 bytes and the controller will begin looking for the $FB (which marks the beginning of the data field) after encountering these zeros.

Again the CRC bytes are reset by the 3 $F5s.

The $FB should not be changed as it marks the beginning of the data field.

The next series of bytes are the actual data saved in the sector just identified. This of course can be anything, data, program code or a secret code we've placed here for our checker to look for. On a newly formatted disk with 256 bytes per sector (as DOS would give us) each of these bytes is $FF (255 decimal). They don't have to be $FF though. You can change it to almost anything to fill the data field when formatting.

After the data field a new CRC is calculated. If we know the CRC values, we can check them to see if any changes have been made to the program. Here is an example. Suppose a talented pirate decodes our checker routine. Let's say that routine read track 35 sector 20 and checked for a code word we put in the sector. Failing to find track 35, sector 20 or the right code there our checker would cause the program to erase itself and make the computer start randomly executing commands forcing the user to do a cold start to regain control.

Our pirate friend changes the program to bypass the checker completely so any copy will execute like the original. Fine. But we have another checker routine a little later in our code that reads the CRCs for the sector that contained our first checker that he changed. Bingo, the CRCs are wrong and we send the computer on a trip to outer space. His attempt to bypass the checker fails and now he has to start looking for the next one. We can do this as many times as we want to.

After the CRCs is another gap, the post data gap. This gap is 24 bytes long and again can be longer or shorter. Here again we can check the length of the gap between the end of the data field and the start of the next sync field as an example.

At the end of each track there is another gap of 200 bytes. This gap can also be varied in length.

To become really comfortable with the disk format we suggest you use the READTRAK and MAKETRAK programs on the SUPER DISK to "play" with different formats and examine what we've done to the tracks beyond track 34 on the SUPER DISK. Note that some of the programs on the SUPER DISK are not copy protected for your ease of use. Others are.

FLOPPY DISK CONTROLLERS

Although there are several floppy disk controllers available for the Tandy Color Computer 3, we'll restrict our discussion to the FDC 1793 manufactured by Western Digital and sold by Tandy. Most of the information in this section applies to all COCO compatible controllers though. We highly recommend that you contact Western Digital and obtain a copy of the manual for the 1793. This manual is free for the asking as of the date of this writing.
The address is; WESTERN DIGITAL CORPORATION
2445 McCABE WAY
IRVINE, CALIFORNIA 92714

Ask for a copy of FD179X-02 Floppy Disk Formatter/Controller Family.

This manual fully explains the hardware and software aspects of our trusty controller. To gain a full understanding of it's operations you need this manual. We will look only at the software aspects here.

FDC COMMANDS

There are 11 commands that can be sent to the 1793. These are;

1.	$03	RESTORE	restores read/write head to track zero
2.	$17	SEEK	seek a specified track
3.	$23	STEP	step in the direction of the last step
4.	$43	STEP OUT	move out 1 track, decrement track count
5.	$53	STEP IN	move in 1 track, increment track count
6.	$80	READ SECTOR	read 1 sector
7.	$A0	WRITE SECTOR	write 1 sector
8.	$C0	READ ADDRESS	read next encountered address marks
9.	$E4	READ TRACK	read 1 entire track
10.	$F4	WRITE TRACK	write 1 entire track
11.	$D0	FORCE INTERRUPT	cease current operation

RESTORE: This command will cause the read/write head (RWH) of the disk drive to go from it's current location to track zero. Upon arrival there an interrupt is generated and control is returned to the computer. No read or write operations are conducted.

SEEK: The seek command can be used to place the RWH at the beginning of any track without contacting the diskette. No read or write operation is performed.

STEP: This command will cause the RWH to step (move) one track in the same direction as the last step command. That is IN (toward the center of the disk, the higher numbered tracks) or OUT (toward the outside of the disk, the lower numbered tracks).

STEP IN: The controller will cause the RWH to move in one track from it's current location upon receipt of this command.

STEP OUT: This command moves the RWH out or down one track from it's current location.

READ SECTOR: Upon receipt of this command the controller will read the sector specified in the sector register and transfer the data to the ram location specified.

WRITE SECTOR: This command will cause the transfer of data pointed to in ram to be written to the specified sector on the disk.

READ ADDRESS: This command will look for the next encountered $FE and transfer the six bytes following it to ram. Those six bytes should be the track number, side number, sector number, sector size and two CRC bytes.

READ TRACK: When this command is executed, the desired track is transferred to the pointed to ram address. This includes the leading gap, all data between the sectors, the data in the sectors, and the end of track gap.

WRITE TRACK: This command will write an entire track to disk. All of the required data (gaps,sync bytes,address data,etc.) RSDOS uses this command when formatting diskettes. It can also be used to copy entire tracks from one disk to another.

FORCE INTERRUPT: This command will force the controller to stop executing its current operation and give control back to the computer.

If you examine copy protection schemes, you'll find that many of them do not rely on the standard DOS commands but access the disk via routine written by the author. One reason for this is that many of the operations conducted in copy protection have no DOS equivalent. As an example, how would you read an entire track or load the CRCs for the data for track 16 sector 10 from DOS?

If you are a BASIC programmer, you might think you can't take advantage of some of the more sophisticated copy protection techniques. Not true. Several of the BASIC utilities on the SUPER DISK will let you do these things. See the section on BASIC PROTECTION.

ACCESSING THE 1793

The registers used to access the 1793 Floppy Disk Controller from the COCO are $FF48, $FF49, $FF4A, $FF4B, $FF40 AND $0986.

$FF48: All commands are sent to the controller by placing the command byte in $FF48. For example, to read a track, $FF48 is loaded with $E4. The status of all read operations is returned in $FF48 upon completion of the operation.

$FF49: This is the track register. The desired track is read from location $EC and placed in $FF49. Most DOS commands load $EC with the track to access. $097E-$0981 contain the current location of the RWH for drives 0-3 respectively. If we want to access track 16, we load $EC with #16. Execute the desired command. The 1793 then gets the current track location of the

RWH from $097E (if drive zero) and compares it with $EC. If they match, the head is on the right track now. If not, the SEEK command is executed and the head moved to the right track.

$FF4A: This is the sector register. $ED (the DOS sector variable) is loaded into $FF4A after the RWH is on the correct track.

$FF4B: This is the data register. All data passed between the computer and the FDC (read or write) goes via $FF4B.

$FF40: This register controls the following functions.

Byte	7	if set halt enabled, if 0 halt disabled
	6	selects drive 3 if set
	5	if set double density, if 0 single density
	4	write precompensation, if 0 no, if set yes
	3	if set motors on, if 0 motors off
	2	selects drive 2 if set
	1	selects drive 1 if set
	0	selects drive 0 if set

$0986: is the ram image of $FF40 since $FF40 is a write only location. To determine the contents of $FF40, read $0986.

DOS

The disk operating system resides in memory from $C000 to $DFFF. Covering the operating system requires a book in itself. Just such a book is available, written by Spectral Associates. This book covers both Disk Basic 1.0 and 1.1. Their address is;

Spectral Associates
3320 South 90th Street
Tacoma, WA 98409

Some of the more important memory locations you should be familiar with for copy protection purposes are listed and explained below.

$00EA	Disk command variable. 00 = RESTORE 01 = NO OPERATION 02 = READ SECTOR 03 = WRITE SECTOR
$00EB	Drive number 0-3
$00EC	Track number 0-39
$00ED	Sector number 1-255
$00EE	Most significant byte of data buffer
$00EF	Least significant byte of data buffer
$00F0	Status register, = 0 if no error in operation
$D75F	DSKCON conducts DOS commands

With just the variables listed above you can access many disk operations from basic. For example, the program below will read any sector on a disk and display it's contents on the 32 column text screen and print the status of the read operation.

```
0010 WIDTH 32 : CLS      ; SELECT LORES SCREEN / CLEAR IT
0020 POKE &HEA,2         ; READ OP CODE
0030 POKE &HEB,0         ; SELECT DRIVE ZERO
0040 POKE &HEC,17        ; SELECT TRACK 17
0050 POKE &HED,3         ; FIRST PAGE OF DIRECTORY
0060 POKE &HEE,4         ; MSB OF $0400, TEXT SCREEN TOP
0070 POKE &HEF,0         ; LSB
0080 EXEC &HD75F         ; JUMP TO DSKCON (BASIC 1.1)
0090 X=PEEK(&HF0)        ; GET STATUS OF OPERATION
0100 PRINT @288,X        ; PRINT STATUS VALUE ON SCREEN
0110 END                 ; FINISHED
```

To change the track number change the 17 in line 40 to the desired track number. To change the sector number change the 3 in line 50 to the desired sector number. For BASIC 1.0 change the &HD75F in line 80 to &HD66C.

Many other locations in ram are used by DOS. The program called DOSVARIA on the SUPER DISK will help you become familiar with them if you are not already.

THE SNOOPER

The SNOOPER is a powerful collection of disk utilities. With it you can examine individual sectors, look for specific data, change data, fill all or part of a sector with a specified byte pattern, do screen prints, recover lost files and perform several copy protection functions. Each function of the SNOOPER is explained below.

STARTING the SNOOPER

Place your SUPER DISK in drive zero and type <LOADM "SNOOPER" ENTER>. The SNOOPER will load and execute. On the opening screen you will be informed that the program is ready to read a sector. At the prompts, enter the drive the disk is in that you wish to read a sector from, the track number and sector number. The values you enter are not restricted to tracks 0-34 or sectors 1-18 to give you the maximum freedom while creating or examining copy protection. If the track or sector entered is not found, you will be informed that a read error has occurred and returned to the opening screen. [To see the first sector of the directory of the SUPER DISK, enter drive zero, track 17, sector 3.]

After you have successfully loaded a sector, you will see the currently accessed drive, track and sector number displayed at the top of the screen. All 256 bytes are displayed in hex numbers in 16 lines of 16 bytes each. $00 and $FF are highlighted to make data fields easily identifiable. To the left of these bytes are the byte numbers of the first byte in that row. To the right of the hex display, each byte is displayed in ASCII format. [If you are looking at a directory sector or ASCII file, you will be able to read the entries.]

Below this display are listed options you may exercise in the SNOOPER.

1. (F1) Select drive. After you press F1, you may enter the number of the drive you wish to access on the next command. Drives 0-3 are supported. Ram drives cannot be accessed with the SNOOPER. This function allows you to switch between drives easily without restarting the program.

2. (T) Press T to change the track number you wish to access next. You will also be asked to enter the sector number of the track. After you press ENTER, the track and sector selected will be displayed. If the track or sector selected cannot be found, you will be notified and returned to the startup screen. In some copy protection schemes, the track numbers beyond track 34 are changed to numbers larger than 39 (track 40). The operating system cannot access these tracks. You will have to use the READTRAK module to access them.

3. (S) Press S to select a sector in the same track to be displayed. If the desired track has been found, you can access any sector numbered 1-255 in the track. In some copy protection schemes, several sectors may have been given the same sector

number. [For example, suppose 16 of the sectors on track 35 all have the sector number of 55 but each of the sectors have different data in them (such as different code numbers for the checker routine to look for). If you select sector 55 with the SNOOPER, one of these sectors will be found and displayed. Selecting sector 55 again may display the same sector again or a different one. The best way to read these sectors is with the READTRAK module.]

4. (D) Press D to convert a hexadecimal number to a decimal number. All hex numbers less than $8000 will display their decimal equivalent. The routine in this program does not convert numbers equal to or larger than $8000. This function will come in handy for converting directory information and file allocation information to decimals so you can find the starting track and sector of different files. You will seldom need to convert a number larger than $FF (255 decimal).

5. (W) Press W to write a sector to disk. After you press W, you will be asked which drive, track and sector to write to. If you press W by mistake or change your mind, press A to abort the write. While exploring the wonders of copy protection, you may want to try to delete certain steps in a file. You could do this by making a regular backup of a disk, using the backup, fill certain areas of code with NOPs (no operation op codes) then write the sector back to disk. Also this function combined with the (C)hange byte function makes it easy to enter code sequences in hidden sectors of your own for your unique copy protection schemes. You can also make a reserve copy of the directory track on an empty or extra track, one sector at a time.

6. (X) Press X to do a screen dump to your printer. This function will print all of the information currently on the screen to the printer if it is on line. If the printer is not online, after a short delay, control will return to you. This function is set to operate at 600 baud to be COCO compatible.

7. (P) Press P to look for preambles and postambles. Preambles are the headers which tell the computer what type of file is being loaded, where in memory to load it and how many bytes to load. Preambles and postambles are discussed in detail at the end of this section.

8. (ENTER) Press the enter key to load the next sequentially numbered sector. If the current sector is the last sector on the current track, the first sector of the next track will be loaded.

9. (Q) Press Q to leave the SNOOPER and return to disk basic. Once you leave the SNOOPER you will have to reload it from disk to reenter it.

10. (L) Press L to look for a specific hex byte. Every occurrence of that hex number in the sector will be highlighted. This function is very useful for finding specific references to disk variables, etc. without having to manually read every byte.

11. (F) Press F to fill any portion of the sector with a specified byte. [For example, to remove all data from a sector and make it like a newly formatted sector, select $FF as the fill character. Start at byte zero. End at byte 255. Press Y to confirm that you want to go ahead. All of the bytes will be changed to $FF. NOTE: No data on the disk is changed unless you then perform a WRITE to disk.]

12. (C) Press C to change a single byte value in a file. You will be prompted to enter the number of the byte to change and the value to change it to. This function can be used in many ways. Some uses are to change a file name or extension, make a disk appear to be full of data by filling all of the unused granules with numbers that will make the computer think that there is no free space left on the disk and is useful when trying to recover a file that was deleted or repair accidentally damaged data in a file. This function is not listed on the on screen menu.

PREAMBLES & POSTAMBLES

When a file is written to disk using DOS, the first 5 bytes written to the first sector are called the preamble. The first byte will be a $00 if the file is binary or $FF if the file is a basic program. Bytes 2 and 3 identify the length of the data file. Bytes 4 and 5 contain the load address. This information is required by the DOS so it knows what it is loading, how long it is and where to put it.

There can be 1 or several preambles in a file. In most machine language protected files this is the case for several reasons. First, the program may alter parts of the operating system at several locations in ram. Loading changes to interrupt vectors directly into their vector locations saves a lot of code as opposed to writing a routine to put them there. Second, the program may be intentionally segmented, broken up into several pieces so it cannot be easily saved to a different disk. DOS does not provide for saving segmented files. Third, it's another way to confuse pirates.

At the end of every file saved to disk using DOS is a postamble. Again, this is made up of 5 bytes. The first byte is $FF, the postamble flag. Bytes 2 and 3 are always $00,$00. Bytes 4 and 5 tell the computer the EXEC address, where the actual start of the program is.

When using this function, always start on sector 1 or 10 of any track other than 17. Track 17 is reserve for the directory (sectors 3-11) and file allocation table (sector 2). Sector 1 and sectors 12-18 of track 17 are not used by the system.

FINDING THE START OF A FILE

All directory listings contain 32 bytes. Bytes 0 thru 15 contain information. Bytes 16 thru 31 are unused and contain zeros.

Bytes 00-07	contain the filename		
08-10	the extension		
11	File type;	00 = Basic program	
		01 = Basic data	
		02 = Binary	
		03 = text editor source	
12	the ASCII flag;	00 = Binary or Basic	
		FF = ASCII file	
13	First granule of the file		
14-15	number of bytes in the last sector		
16-31	not used, all 00s		

To find the track and sector where a file starts then we need to see byte 13. This gives us the first granule of the file. This number must be converted to a track and sector number. There are 68 granules on a 35 track disk. Track 17 is skipped, so that leaves 34 tracks. That means 2 granules per track. One granule is sectors 1-9. The other is sectors 10-18. Even the shortest BASIC program (say 200 bytes, less then 1 sector) will use up 9 sectors on the disk. Use the table below to easily convert granules to tracks and sectors.

GRANULE # =TRACK NUMBER / STARTING SECTOR NUMBER

00=00/01	01=00/10	02=01/01	03=01/10
04=02/01	05=02/10	06=03/01	07=03/10
08=04/01	09=04/10	10=05/01	11=05/10
12=06/01	13=06/10	14=07/01	15=07/10
16=08/01	17=08/10	18=09/01	19=09/10
20=10/01	21=10/10	22=11/01	23=11/10
24=12/01	25=12/10	26=13/01	27=13/10
28=14/01	29=14/10	30=15/01	31=15/10
32=16/01	33=16/10	34=18/01	35=18/10
36=19/01	37=19/10	38=20/01	39=20/10
40=21/01	41=21/10	42=22/01	43=22/10
44=23/01	45=23/10	46=24/01	47=24/10
48=25/01	49=25/10	50=26/01	51=26/10
52=27/01	53=27/10	54=28/01	55=28/10
56=29/01	57=29/10	58=30/01	59=30/10
60=31/01	61=31/10	62=32/01	63=32/10
64=33/01	65=33/10	66=34/01	67=34/10

Once you've determined the track and sector number where a program begins you can use the preamble search module to find out where in memory the program resides and executes. You won't need this module for creating copy protected software but it is extremely useful for playing detective and examining other programs. Don't be surprised if you see lots of preambles. [NOTE; The preamble search extends only to the end of the sector in memory. If the number of bytes in the segment takes you past the end of the sector you must load the next sector and begin the search at the last byte indicated by the last preamble. If you don't, you will get false information. Also, the preamble search is not designed to recognize BASIC programs. Remember that BASIC programs preambles start with $FF not $00. BASIC programs are not segmented either, so are easy to follow.

READTRAK and READRITE

The READTRAK (RT) module allows you to load any track that exists on a disk and scroll through the entire track. This includes everything from and including the leading gap to the end gap. All gaps, sync bytes, CRCs, address marks, ID fields and data fields can be seen and examined. This applies to both protected and unprotected disks.

LOADING INSTRUCTIONS

To execute this module, put the SUPER DISK in drive zero and type [LOADM "READTRAK" ENTER]. The module will load and execute. You should see the title screen and be prompted to enter a drive number. You can remove the SUPER DISK at this time if you like and place a disk to examine in drive zero. All drives 0-3 can be used. Ram disk drives cannot be accessed by this module. Select a drive number and press enter.

Next you will be asked what track to read. Enter the number of the track you want to see and press enter. The module will fill it's ram buffer with zeros before it attempts to read the track. After the track is read you will be prompted to press any key to view the track. Press any key. If the track dose not exist or cannot be read for some reason, you will see a buffer filled with zeros instead of data.

The screen will begin to fill with data from the ram buffer. You will first see the leading gap followed by the sync bytes. Next you should see a $FE. This is the first address mark. The next four bytes after the $FE are the track number, side number (should be a zero), the sector number, and the sector size (should be a one). The next two bytes are the CRC calculated for the address field. This is followed by another gap, another sync field and a $FB. This $FB marks the beginning of the data field (the next 256 bytes). So far all this has been in white letters on a blue background. The data field appears in white on a red background to make the data fields easy to see. At the end of the data field the colors switch back to white on blue. Right after the data field are it's CRC bytes. Then the whole sequence repeats again for the next sector. This continues until all 18 sectors are displayed and the track end gap.

To PAUSE the display, press shift and @ at the same time. To continue scrolling through the track, press any key.

To TERMINATE the scroll, press the F1 key. This will end the scrolling and present you with three options.

1. READ TRACK Select this option if you want to read in a different track. The buffer will be refilled with zeros and you can select another track to read in.

2. VIEW SAME TRACK Select this option to see the same track from the beginning again.

3. QUIT Select this option to exit READTRAK and go back to DISK BASIC. To restart READTRAK you must reinsert the SUPER DISK and LOADM "READTRAK" again.

It is a good idea to do a cold start after running any of the SUPER DISK modules because most of them modify some of disk basics routines.

Pressing the BREAK key at any time except during disk access will terminate the program. Do not press the break key unless you want to exit the program.

READTRAK is extremely useful for examining information on protected disks that you were not meant to see. You can learn a lot about copy protection by looking at how the author set up the different tracks, the track numbers used, side numbers, sector numbers, data fields, length of gap etc.

As far as using it for creating your own copy protection is concerned, here is how to use it. After you have designed your own unique track format (using FORMAT and DSKIMODI) you can load the entire track and verify that it is as you expect it to be. You can also select by exact byte count where the data you want your checker to find is located. Then you can have the checker routine load the entire track, certain accessible sectors only or even just the address field or data field CRCs. The point is, the data your checker is to look for must be where it is expected to be. This module will help you make sure it is.

SOME PECULIARITIES

Reading entire tracks of data is not a standard part of the DOS. No where in DOS is it called for. Writing entire tracks is called for in DOS only in the DSKINI routine to format a disk. Two major problems occur when reading and writing entire tracks (except while formatting).

First, the drive seems to loose sync. You will notice that some of the bytes in the gaps between the sectors that are supposed to be $4E (the bytes sent when the disk was formatted) are not. They are $12 or some other value. This is because the drive gets out of sync and misreads the byte. It gets back in sync just before the address id field and the data field. But it can soon loose sync again very quickly. This alone presents little problem because the address field comes out correctly. The start of the data field usually is correct but you may notice that some of the bytes in the data field are not correct. Try this test to see what I mean. Use the SNOOPER to load a sector that contains data, say sector 3 of track 17, the directory. Print it out if you have a printer. Exit the SNOOPER and read in track 17 with READTRAK. Find sector 3. Compare them. Especially note that the $FFs may have been changed to $00. This problem gets worse when a data field contains program code. See why in the next paragraph.

The second problem occurs mostly in the write track command. This command was designed to be used to format diskettes only. It works perfectly for that purpose. Suppose you want to copy a disk by first reading each track in it's entirety from one drive and then writing it to another drive. You can do it! SEE BELOW.

READRITE

First, do a cold start and take a new disk and format it using the regular DSKINI command. Have a new unformatted disk ready but do not format it.

Now, place the SUPER DISK in drive zero and type [LOADM "READRITE" ENTER]. READRITE will load but not execute. Place the FORMATTED disk in drive zero and the UNFORMATTED disk in drive 1. (This module requires 2 drives). Type [EXEC]. READRITE will copy track by track from drive zero to drive 1. When it is finished, the disk in drive 1 will be an exact copy of the disk in drive zero and it can be used just like any other newly formatted disk.

Next, do a cold start and reload READRITE. Place any disk that contains programs and data on it in drive zero and a new unformatted disk in drive 1. Type [EXEC]. Again, the module will copy track by track the disk in drive zero to the disk in drive 1. Do a cold start and type [DIR1]. The directory may or may not come up correctly. If it does, try to load and run one of the programs on the disk. Did it work? Rarely it will. Sometimes, if you then do a sector by sector copy, the disk will function perfectly. Sometimes not.

Here's the problem. Even if the data was read from the disk in drive zero without error the write will not be correct. Whenever the write track function of the floppy disk controller encounters certain codes in the data it tries to perform certain functions. For example, if the FDC reads a $FE (address ID mark) from it's data register while writing a track, it wants to create an address field. If it sees an $FB, it thinks it's supposed to create a data field. If it sees a $F7, it writes two CRC bytes. $F5 is the code for it to reset the CRC counter.

These codes $F5-$FE cannot appear in the data field, gaps or ID fields. They raise havoc with the controller. Just use the SNOOPER and look through some program code sectors. Use the LOOK function to quickly see how often these codes, $F5-$FE, occur. See the problem?

READRITE knows this problem and looks for the offending codes and replaces them with $FF. The disk usually ends up being formatted correctly, but the data fields are not identical if they contained any $F5s through $FE. The only solution is to do a sector by sector copy after using READRITE.

Copy protection authors know these things and often make sure that tracks used for protection contain lots of these codes. After all, you aren't supposed to be able to copy it!

READRITE will copy disks with any number of tracks from 1 to 40. The module checks after each read to see if it received data. If not, the execution ends.

You can reexecute READRITE by typing [EXEC] again.

NOCOPY

Put this one in the dirty tricks file. NOCOPY is a module that will make a disk not copyable with the DOS BACKUP command. This routine is especially effective under some circumstances. First, if the program on the disk can be copied using the DOS COPY command, NOCOPY is useless unless the program also has a checker routine. That way, even if it is copied, it wont work because the data the checker looks for is on a track that the BACKUP command does not look for. Second, NOCOPY makes it impossible for the user to make a copy that can be used to restore a damaged original. Many companies now instruct you to make a backup to save in case something happen to the original. If the original fails to work, you use the copy to restore the original.

Before you use NOCOPY you must decide that you don't want the user to be able to make a copy for any purpose. If you do decide that, you had better have a good replacement policy or you are going to have some unhappy customers out there.

NOCOPY works most effectively this way. Your disk contains only one directory entry. That is a loader program that points to a track and sector number where the actual program begins. The loader then loads consecutive sectors into memory until the program is loaded at which time it executes itself. You must do a little work to accomplish this.

First you take a formatted disk and save your program on it. If you use the SNOOPER to look at track 17, sector 3 you will see the file listed and the granule number where it starts. Let's pretend it says granule number 30. That converts to track 15, sector 1. Let's also say that the file fills 12 sectors. That means it resides on track 15, sectors 1 through 12. That's 2 granules then, number 30 and 31. Now you write a short loader program that will load sectors 1 through 12 of track 15 into ram at the address you specify. You save it on the disk and find that it resides on track 16, sectors 10-17. That's one granule, number 33. You run the loader and it works.

Now, use the SNOOPER again and change all 32 of the bytes of the directory that show your main program's name etc. to $FF. In other words, erase all trace of it from the directory. Write the new directory, which contains only the loader program's name back to disk.

Again, with the SNOOPER, load sector 2 of track 17, the file allocation table. The first 68 bytes are the file allocation table. Each byte's value tells the computer if the granule it represents is unused or used. If it is unused it contains a $FF. If it is part of a file it points to the next granule of the file or tells the computer how many sectors in the granule contain data. Our example would show all $FFs except for granules 30,31 and 33. We know 33 is the loader so we don't want to change that one, we need it.

We can hide our real program and make the disk appear full at the same time. Why make the disk appear full? That way no one can write to it and possibly overwrite our hidden file. Also, the whereabouts of our file become a mystery.

So we do this. Using the SNOOPER's FILL function, we fill the first 29 bytes with $20 and the bytes from 34 to 68 with $20. Using CHANGE, we change byte 32 from $FF to $20 also.
The only one that's different is 33, which is data about the loader program. Our real program has disappeared from the directory and file allocation table, but it's still there on track 15. If you type DIR, you see only the name of the loader program. If you type PRINT FREE (0), you get a no free space response. Neat, isn't it.

If someone tries to copy the program with the DOS COPY command, all they get is the loader, no program! The backup command would copy all of it though. This is where NOCOPY comes in.

Now we load NOCOPY by putting the SUPER DISK in drive zero and typing [LOADM "NOCOPY"]. NOCOPY will load and execute. The screen prompts you to place a formatted disk in drive zero and press enter. We put our disk in drive zero, press enter, and in a flash it's done ready to do another.

(WARNING: DO NOT USE NOCOPY ON DISKS THAT CONTAIN DATA OR
PROGRAMS ON TRACK ZERO. TRACK ZERO IS DESTROYED BY NOCOPY.)

Now we have a disk that can't be copied by the normal BACKUP command. It will generate an I/O ERROR message. COPY will only copy the loader program, not the real program. And, if we were smart, there is data on a hidden track that a checker routine in the real program looks for just in case. That's four levels of protection.

For the users sake, such a program should be delivered on a "flippy" diskette (one with 2 sides) and the program should be on both sides. That way, he has two working copies. But they are on the same diskette. A performance guarantee is in order also. Something like, free replacement for one year if both sides fail to work properly and replacement for a small fee after one year sounds more than reasonable to me.

Press BREAK to exit NOCOPY.

ANALYZE

Here's one for your Disk Detective's Toolkit. While you are studying copy protection you'll find this module handy. ANALYZE will very quickly try to find tracks 0-39 on any disk placed in drive zero by looking for sector 1 of each track. What we are really looking for are tracks formatted in a non-standard way. In the world of copy protection, it's what you can't see not what you can.

To run this module, place the SUPER DISK in drive zero and type [LOADM "ANALYZE" ENTER]. ANALYZE will load and execute. You will be prompted to place the disk to analyze in drive zero and press enter.

After you press enter, the module will attempt to read sector 1 of each track from 0 to 39. If the sector is found, the message TRACK x   FOUND appears. If it cannot be found, the message TRACK x NOT FOUND appears.

This is a simple module that produces very basic information. But what it does is produce it very fast. Imagine how long it would take you to do the same thing 40 times.

What can you do with this information? Tracks that are found in the range of 0 to 34 are probably adhering to the standard DOS format. Tracks in that range that are not found are very suspect. Tracks in the 35-39 range that are found need to be examined. Likewise, tracks in that range that are not found need to be looked for with READTRAK. Very few copy protected programs have only 35 tracks. You can be assured that there is at least one extra track there.

What ANALYZE does is tell you where to look for secret data by pointing out what is accessible by the normal operating system. You should use it before you use READTRAK or the SNOOPER. It will save you a lot of time.

After you run ANALYZE the first time and get it's report, you can analyze another disk or quit. To analyze another disk, press the (A) key at the prompt. To quit and return to disk basic, press the (Q) key at the prompt. You can also exit the module by pressing the (BREAK) key.

BASIC PROTECTION

If you write most of your software in BASIC you might think that copy protection cannot be used. Not so. All of the techniques used by ML programmers can be used with BASIC. All you need to do is make you program secure and generally inaccessible.

Before any copy protection is added though, the BASIC program must be error free. Any error that causes the computer to stop executing program instructions and display an error message will make the code accessible. Your code must be error free.

Here are the steps necessary to create a well protected BASIC program on disk;

1. Confirm that the program is error free.

2. Use FORMAT and DSKIMODI (see separate instructions) to create a specially formatted disk. For our example let's say we make a disk with 36 tracks. On track 35, in a sector we've numbered 30, all 256 bytes in the data field contain the number 1.

3. Add the checker routine to the basic program. Below is a checker routine in BASIC that will look for track 35 (tracks are numbered 0-39, so track 35 is actually the 36th track). If there is no track 35 (a disk formatted with the DSKINI command will NOT have track 35) the checker will not run the program. If track 35 is found, the checker will attempt to load sector 30 (DSKINI formatted disks will only have sector numbers 1-18). If sector 30 is not found, the checker will not run the program. Finally, if sector 30 is loaded, the checker will add the values of all of the bytes in the sector and compare the result to the number 256. (on a disk formatted by DSKINI, the total would be 65280) The correct number should be 256, since we formatted the disk sector with all 1s. 256 x 1=256. If the total is anything other than 256, the checker will again refuse to run the program.

This gives us 3 levels of protection. The checker must find track 35, sector 30, and the total of the byte values in that sector must equal 256. If the disk in the drive fails any one of these tests here's what will happen. The checker will display a message that says "THIS IS A BACKUP. USE IT TO RESTORE THE ORIGINAL IF NECESSARY. PRESS ANY KEY." When the user presses a key, the system will do a cold start.

4. Before the checker routine is added to the program though, a coulpe of other safeguards need to be inserted. The BREAK key must not be allowed to stop execution of the program. Fortunately, in the COCO 3 this is easy. Just include at the beginning of the program an ON BREAK GOTO statement. (See your BASIC programming manual if you are not familiar with this command. You could have the destination of this command do a cold start.

The RESET button must not do a warm start. If the user presses the reset button during execution, normally a warm start is executed and the user ends up in direct mode with the program still in memory and intact. If you POKE $71,0 in the beginning of the program, pressing the reset button will force a cold start and the program cannot be listed.

These are not the only methods you can use. The BREAK key can be made not to function at all. The RESET button can also be disabled. Commands like LIST can be made not to function too. Do whatever you like. The point is, you don't want the user to be able to BREAK out of the program and list the code and remove the checker routine.

THE CHECKER ROUTINE

```
0100   POKE &HEA,02        : this is the read sector op code
0110   POKE &HEB,00        : select drive zero
                             disk to check must be in drive 0
0120   POKE &HEC,35        : select track #35
0130   POKE &HED,30        : select sector #30
0140   POKE &HEE,&HDF      : sector will be loaded into memory
0150   POKE &HEF,&H00      : in the area occupied by the DOS cmd
0160   EXEC &HD75F         : go read in the sector (DOS 1.1)
                           : DOS 1.0 is EXEC &HD66C
0170   X=PEEK(&HF0)        : get the DSKCON status flag
0180   IF X<>0 THEN 500    : track or sector not found if $F0 is
                           : not equal to zero
0190   Y=0                 : clear Y
0200   FOR L=0 TO 255      : loop 256 times
0210   X=PEEK(&HDF00+L)    : get value at $DF00 + counter
0220   Y=Y+X               : add it to Y
0230   NEXT L              : get the next byte
0240   IF Y<>256 THEN 500: if total isn't 256 go to message

(((  line 250 is the beginning of the regular program. )))

0500   WIDTH 32:CLS        : goto text screen & clear it
0510   PRINT "THIS IS A BACKUP"
0520   PRINT "USE IT TO RESTORE ORIGINAL"
0530   PRINT "IF NECESSARY."
0540   PRINT "PRESS ANY KEY."
0560   K$=INKEY$:IF K$="" THEN 560 : wait for key press
0570   POKE &H71,0         : set flag for cold start
0580   EXEC &H8C1B         : do a reset (cold start)
```

You may want to type this program in and save it on a disk formatted by DSKINI. Then run it. You should get the message after the checker can't find track 35 because $F0 will contain an error code, not zero. Then, just to be sure it really works, format a disk with FORMAT and DSKIMODI and save this program on it. Run it again and verify that it works.

Once you become proficient with these tools you can design your own format and modify the checker to suit it. Protect your work.

5. Now we're ready to use BASSAVE/BIN. Load your BASIC program, complete with all protection in place. That includes the checker routine, break key and reset routines. Now put the SUPER DISK in drive zero and type [LOADM "BASSAVE"]. The routine will be loaded at $DF00, well out of the way of the memory where your basic program resides.

Remove the SUPER DISK and put your specially formatted disk in drive zero. Type [EXEC]. BASSAVE will save the basic program to the disk in drive zero and name it BASXXXXX/BIN.

You must now determine the start and end addresses of the basic program. To find the start address, type;

```
[ PRINT HEX$(PEEK(&H19));HEX$(PEEK(&H1A)) ]
```

Write down the 4 characters returned by this peek. This is the start address in hexadecimal.

To get the end address, type;

```
[ PRINT HEX$(PEEK(&H1B));HEX$(PEEK(&H1C)) ]
```

Write these 4 hex characters down too. This is the end address of the basic program +1.

6. Now decide on a name to call this binary file. This file is the basic program saved as a binary file. Whatever you decide to call it just be sure the name has eight characters (some of them can be spaces. The extension must be BIN. For our example, let's call it MYPROGRA/BIN.

Type [RENAME "BASXXXXX/BIN" TO "MYPROGRA/BIN"] and press ENTER.

7. Clear memory and put the SUPER DISK back in drive zero. Type [LOADM "BASRUN"] and press ENTER. BASRUN will load. We can now modify it to load and run our basic program that we just saved as a ML file. Put the protected disk with MYPROGRA/BIN on it back in drive zero.

To get BASRUN ready to use we need to tell it the name of the program to load and where to put it. First lets put the name in place. To do this we need to poke the 8 characters of the name into the proper location in BASRUN. The BIN extension is there.

We must poke the value of each letter of the name into memory starting at $DF29. MYPROGRA translates to;

```
77,89,80,82,79,71,82,65
```

You can use the utility called ASCCODE/BAS on the SUPER DISK to get these values if you like. Just load ASCCODE/BAS and change the letters in line 60 to the name you selected for your program and make a note of them on paper.

To poke these values in place type in and run this short program.

```
0010   FOR L=&HDF29 TO &HDF33         : locations to poke
0020   READ A                         : get a value
0030   POKE L,A                       : put it in BASRUN
0040   NEXT L                         : get the next value
0050   END                            : all done
0060   DATA 77,89,80,82,79,71,82,65   : values to poke in
```

Next, get the start and end addresses you wrote down earlier. These must be poked in also.

Poke the start address in at $DF1D and $DF1E. For example, if the numbers we have are &H2601,

type [POKE &HDF1D,&H26:POKE &HDF1E,&H01] and press ENTER

The end address we got goes at $DF22 and $DF23. If the end address was &H29E7;

type [POKE &HDF22,&H29:POKE &HDF23,&HE7] and press ENTER

Now we're ready to save the loader to our game disk. With the disk containing the game program in drive zero, decide on a name for the loader (let's use GO/BIN) and type;

```
SAVEM "GO/BIN", &HDF00,&HDF33,&HDF00
```

This puts the loader on the disk with the program it is to load and RUN.

To verify that all is done correctly, clear memory. Type DIR. You should see MYPROGRA/BIN and GO/BIN listed.

Type LOADM "GO" and press ENTER. Type EXEC after the drive stops. The loader (GO/BIN) should load the program called MYPROGRA/BIN and run it.

Check to see if what you wanted to happen happens when you press the BREAK key and RESET button. If it does, you should have a well protected BASIC program. If not, go back and check your work. If the loader fails to load the program, recheck your work on BASRUN/BIN. If the loader loads and runs the game program but you get the protection message, check your work on the checker routine and the format of the disk.

If you did each step carefully and verified that it functioned properly after you did it, you should have no problems.

The source code for BASSAVE/BIN and BASLOAD/BIN are listed in APPENDIX A at the end of this manual.

FORMAT AND DSKIMODI

These two programs are the heart of the copy protection system of the SUPER DISK. With them, you can create unlimited disk formats. As you experiment with them you will be able to devise your own unique protection schemes.

FORMAT

FORMAT is a basic program that modifies the disk formatting routine in RSDOS. You can format any number of tracks with 18 sectors each. To run this program, put the SUPER DISK in drive zero and type [RUN "FORMAT"] and press enter. The title screen will appear and you will be asked how many tracks you want to format. To simplify your getting acquainted with this module, we'll format a disk for use with the BASSAVE and BASRUN modules example. That is, we'll make a diskette that has 36 tracks. One of the sectors on track 36 will be numbered 30 and the data field will contain all ones. That way it will work with the checker routine in that example too.

The easiest way to have multiple formats on one disk is to format the same disk twice. We'll format it once with 36 tracks and then again with 35 tracks. The second format will be a standard RSDOS format and will not overwrite track 36 from the first format. Here we go.

Press the BREAK key so we can change the fill character for the data field from $FF (which RSDOS uses) to $01 which we want. The byte which contains the fill character is located at $D6E7 in DOS 1.1 or at $D5FA in DOS 1.0. Enter the appropriate poke for your DOS. We're using 1.1 below;

POKE &HD6E7,&H01 and press ENTER

Now type RUN and press ENTER. At the number of tracks prompt, type 36 and press ENTER. The program will end. Read the on screen message.

DSKIMODI

Put the SUPER DISK back in drive zero if you removed it and type

LOADM "DSKIMODI" and press ENTER

DSKIMODI will load but not execute. The assembly source code listing for this module can be found in APPENDIX A. We need to change one of the sector numbers to 30. Type the following line.

POKE &HD921,30 and press ENTER

That will change the sector numbered 20 ($14) to 30. Now type [EXEC] and press ENTER. That will make all the changes to the format routine.

Remove the SUPER DISK and place a new unformatted disk in drive zero and type DSKINI0 and press ENTER. The disk will be formatted with 36 tracks and the sector numbers shown in the DSKIMODI except for the one we just changed.

After the disk is formatted, do a cold start and type DSKINI0 and press ENTER. This will format the tracks numbered 0-34 in the normal fashion. Track 35 will still be as we formatted it the first time. if you want to confirm this, you can use READTRAK and/or THE SNOOPER to examine the disk.

This disk is now ready to use with the example in BASSAVE and BASLOAD.

If you use your imagination, you will be able to come up with some pretty strange formats. Assembly language programmers can easily modify DSKIMODI to suit their needs.

Once you decide on a format you are comfortable with, we suggest you modify DSKIMODI and save it under another name so you don't have to repeat all the steps above every time you want to format a disk. Also you might consider modifying FORMAT to suit your needs and have it load and execute your modified version of DSKIMODI and then execute a DSKINI0 command. This will make mass producing protected diskettes very simple and painless.

It may take you some time to get comfortable with this process. But once you do, you'll feel a lot more secure about how many pirated copies of your software are being passed around.

DOSVARIA

DOSVARIA is included on the SUPER DISK as an educational tool. With it you can examine most of the variables used by the system that are related to disk input/output.

To run this module, just type [LOADM "DOSVARIA"] and press ENTER.

On the main screen is the menu. Press the number for the menu item you want. Each is explained briefly below.

1. I/O BUFFER #0 This is the primary buffer used in all disk I/O. As the data is read from or written to the disk it passes through this buffer for commands such as DIR. It is 256 bytes in size. If you choose it you will see the current contents of the buffer displayed.

2. I/O BUFFER #1 This buffer, which begins at $700 is also 256 bytes and is used by the system for such operations as comparing data when the VERIFY flag is set. Also, the first 18 bytes of this buffer contain the logical sector numbers to be used when a disk is formatted. Take a look at the DSKIMODI source listing in appendix A. Note how the sector numbers are placed in this buffer.

3. FILE ALLOCATION TABLES Items 3 through 6 will allow you to examine the 4 FAT tables stored in RAM. These are first read in when a drive is accessed and allow for faster disk I/O. The first table shows the FAT table for drive 0. Note that it shows exactly the same data in the table stored in sector 2 of the disk in drive 0. The same applies to drives 1-3.

7. DISK VARIABLES This selection will show you many of the ram variables used by the disk system, it's location in ram and a brief explanation as well as it's current value.

8. DFLBUF This area of ram is another 256 byte buffer that contains data for disk I/O operations. For example, when a track is formatted in ram ready to write to disk by the DSKINI routine in DOS, the track is written here byte by byte. It is then copied to the disk.

9. I/O AREA All of the variables shown here are used whenever any disk I/O operations are conducted DOS.

00. QUIT Enter 00 and press enter to return to DISK BASIC.

If you are not very familiar with disk I/O operations, we hope this module will be useful for you. by studying the information presented in it, you will become familiar with the variables used in I/O and the buffers. This is not meant to be a complete tutorial on disk operations. As suggested earlier, several publications are available from other sources that will be especially valuable to the novice.

MAKETRAK

MAKETRAK is a basic program that allows you to easily experiment with designing disk track formats. This module asks you to enter values for ALL of the variables used to construct a track to be used to format a disk using DOS's DSKINI command. ALL except the track numbers, side and sector numbers. These will remain tracks 0-34, side 0 and sectors 1-18. (Use FORMAT and DSKIMODI to experiment with these variables.) The main idea here is to allow you to see just what can and can't be changed.

Use this module to format disks any way you want. Then see if the system will recognize the format. Always do a cold start after you use this module because it modifies the system.

Why do this? If you are trying to design a disk format for copy protection purposes, you'll want to have two features for sure.

First you want your format to be accepted by the normal operating system. It must be able to read the disk and at least retrieve data from it if not write to it. It has to be able to find and get information from the disk.

Second, you need a format that the normal DOS cannot duplicate. You need to have something there for the checker routine you write to look for that isn't found on disks formatted by DSKINI.

After you enter the requested information, the program will end. Then, place a disk to format (don't use a disk with any programs on it you don't want to loose) in drive 0. Type [DSKINI0] and press ENTER. The modified DSKINI routine will try to format the disk as you have suggested. Sometimes it will. Sometimes you will get an error message. An error message means that something you changed was not acceptable.

If you've never modified a format before we suggest that you make only one change at a time. That way, if you get an error you'll know exactly what caused it. It is also a good idea to review the section on floppy disk formats in this manual.

As stated before, not all copy protection routines rely on the addition of extra tracks. Sometimes the protection consists entirely of a small modification to the data between the sectors of what appears to be an ordinary 35 track diskette. This module will make it easy for you to design a similar system. Have fun.

COPYPLUS

COPYPLUS is a combination format and copy utility. This module requires 2 drives to use. With it you can format and backup a disk in one step. This module will copy disks with up to 40 tracks and 18 sectors per track automatically. You do not need to know how many tracks the disk has on it.

To use this module, put the SUPER DISK in drive 0 and type [LOADM "COPYPLUS"] and press ENTER. The module will load and execute. Remove the SUPER DISK and place the disk you want to copy in drive 0 and a new unformatted disk in drive 1. When ready press ENTER.

The module will format the disk in drive 1 and then alternately read and write all of the sectors of each track. If an error is encountered either reading or writing the process will end.

This module is not designed to duplicate copy protected disks. If a track number larger than 39 or a sector number larger than 18 is encountered, the copy process is ended. It's purpose is to make creating copies a lot easier than the normal format and backup routine supplied with RSDOS.

FASTFORM

FASTFORM is another utility to make your life a little easier. This module will format diskettes with 35 tracks and 18 sectors per track at the press of a key. Insert the SUPER DISK in drive 0 and type [LOADM "FASTFORM"] and press ENTER. After the module loads, remove the SUPER DISK and place the disk you want to format in a drive. Type [EXEC] and press ENTER. Press the number of the drive you wish to format. The message " ****** FORMATTING *****" will appear and the disk will be formatted. On completion of the formatting process, the formatting message will disappear and a tone will sound. You can then format another dusk in either drive. You can format as many disks as you like with just a single key press.

To exit the module, press the BREAK key. You will be returned to DISK BASIC.

DISKCOPY

This module is to be used along with FASTFORM. It will BACKUP a disk in one drive to a FORMATTED disk in another drive. If you have to make several copies of a disk, use FASTFORM to quickly format the new diskettes. Then, use DISKCOPY to easily backup as many copies as you like with just a few key presses.

To use DISKCOPY, place the SUPER DISK in drive zero and type [LOADM "DISKCOPY"] and press ENTER. The module will load and execute. Answer the prompts for the drives to copy from and to AFTER you have the disks in the drives.

To exit the module, just answer NO to the copy again query.

APPENDIX A

```
----   PAGE  001 BASSAVE/BIN

                              NAM    BASSAVE/BIN
                        * purpose: save a BASIC program as a
                        *          ML file so it can be RUN
                        *          by executing BASRUN.
                        *************************************
DF00                          ORG    $DF00
                        ***** put name in buffer ******
DF00 31   8D 001E START   LEAY   PRONAM,PCR  name buffer
DF04 8E   094C            LDX    #$094C     file name buffer start
DF07 5F                   CLRB              clear counter
DF08 A6   A0      LOOP    LDA    ,Y+        get a character
DF0A A7   80              STA    ,X+        put it in buffer
DF0C 5C                   INCB              add 1 to counter
DF0D C1   0B              CMPB   #$0B       is it 11?
DF0F 27   02              BEQ    NEXT       yes, goto NEXT
DF11 20   F5              BRA    LOOP       no, keep going
                        ***** set start/end addr ******
DF13 DC   19      NEXT    LDD    $0019      start addr location
DF15 34   06              PSHS   D          save it on the stack
DF17 DC   1B              LDD    $001B      get the end addr
DF19 34   06              PSHS   D          save it too
DF1B DC   19              LDD    $0019      trans addr (start)
DF1D 34   06              PSHS   D          save it on the stack
                        ***** go save file ************
DF1F 7E   CF7E            JMP    $CF7E      do a savem
                        ***** name buffer *************
DF22 42                 PRONAM FCC   /BASXXXXBIN/
DF23 41
DF24 53
DF25 58
DF26 58
DF27 58
DF28 58
DF29 58
DF2A 42
DF2B 49
DF2C 4E
                        *******************************
     DF00                     END               START

TOTAL ERRORS 00000
```

```
----    PAGE  001 BASRUN/BIN

                              NAM    BASRUN/BIN
                       * purpose: load and run a BASIC program
                       *          that was saved to appear to
                       *          be a ML program.
                       ************************************
DF00                          ORG    $DF00
                       ***** put name in buffer ******
DF00 31   8D 0025 START LEAY  PRONAM,PCR  name buffer
DF04 8E   094C            LDX    #$94C    file name buffer start
DF07 5F                   CLRB            clear counter
DF08 A6   A0      LOOP    LDA    ,Y+      get a character
DF0A A7   80              STA    ,X+      put it in buffer
DF0C 5C                   INCB            add 1 to counter
DF0D C1   0B              CMPB   #$0B     is it 11?
DF0F 27   02              BEQ    DONE     yes, goto DONE
DF11 20   F5              BRA    LOOP     no, keep going
                       ***** define file type - ******
DF13 CC   00C8    DONE    LDD    #0200    file type =ml
DF16 FD   0957            STD    $957
                       ***** go load file ************
DF19 BD   CFC5            JSR    $CFC5    do a loadm
                       ***** set start/end addr ******
DF1C CC   2601            LDD    #9729    insert start addr here
DF1F DD   19              STD    $0019    textab
DF21 CC   29E7            LDD    #10727   program end +1
DF24 DD   1B              STD    $001B
                       ***** go run it ***************
DF26 7E   AE75            JMP    $AE75    run command
                       ***** name buffer *************
DF29 42           PRONAM FCC    /BASXXXXBIN/
DF2A 41
DF2B 53
DF2C 58
DF2D 58
DF2E 58
DF2F 58
DF30 58
DF31 42
DF32 49
DF33 4E
                       ********************************
        DF00                  END             START

TOTAL ERRORS 00000
```

```
----    PAGE  001 DSKIMODI/BIN

00010                              NAM   DSKIMODI/BIN
00020                       * purpose: change the logical sector #'s
00030                       *          used by RSDOS DSKINI routine
00040                       *          from 1-18 to user's choice
00050                       ****************************************
00060  1000                        ORG   $1000
00070  1000 86    BD        START  LDA   #$BD      jsr op code
00080  1002 B7    D5CE             STA   $D5CE     insert in DSKI routine
00090  1005 CC    D900             LDD   #$D900    address to branch to
00100  1008 FD    D5CF             STD   $D5CF     insert it
00110  100B CC    1212             LDD   #$1212    * two nops
00120  100E FD    D5D1             STD   $D5D1     * insert 2
00130  1011 FD    D5D3             STD   $D5D3     * and 2 more
00140  1014 39                     RTS             go back to BASIC
00150                       ***** routine to insert sector #'s ****
00160  D900                        ORG   $D900
00170  D900 31    8D 000D          LEAY  SECS,PCR
00180  D904 C6    12               LDB   #18
00190  D906 8E    0700             LDX   #$700
00200  D909 A6    A0        LOOP   LDA   ,Y+
00210  D90B A7    80               STA   ,X+
00220  D90D 5A                     DECB
00230  D90E 26    F9               BNE   LOOP
00240  D910 39                     RTS
00250                       ***** data used for sector #'s ********
00260  D911 80              SECS   FCB   $80,$80,$80,$80,$80,$80,$80,$80
       D912 80
       D913 80
       D914 80
       D915 80
       D916 80
       D917 80
       D918 80
00270  D919 80                     FCB   $80,$80,$80,$80,$80,$80,$80,$80
       D91A 80
       D91B 80
       D91C 80
       D91D 80
       D91E 80
       D91F 80
       D920 80
00280  D921 14                     FCB   $14,$81
       D922 81
00290       1000                   END             START

TOTAL ERRORS 00000
```