WARNING: This book must NOT be used for software piracy.

Hardcore COMPUTIST's

Book Of Softkeys Volume III

SoftKey Publishing

HOW TO REMOVE COPY-PROTECTION
FROM UNCOPYABLE COMMERCIAL SOFTWARE
for the Apple II, Apple II Plus, and Apple //e.
in concise, easy-to-follow steps.

The Book Of Softkeys

Volume III

SoftKey Publishing
PO Box 110846-BK
Tacoma, WA 98411

Other Publications by SoftKey Publishing:

The Book Of Softkeys Volume II.......160 pages

shows you how to deprotect (softkey):

*Apple Cider Spider— Apple Logo— Arcade Machine— The Artist— Bank Street Writer— Cannonball Blitz— Canyon Climber— Caverns of Freitag— Crush, Crumble & Chomp— Data Factory 5.Ø— DB Master— The Dic*tion*ary— Essential Data Duplicator I & III— Gold Rush— Krell Logo— Legacy of Llylgamyn— Mask Of The Sun— Minit Man— Mouskattack— Music Construction Set— Oil's Well— Pandora's Box— Robotron— Sammy Lightfoot— Screenwriter II v2.2— Sensible Speller 4.Ø, 4.Øc, 4.1c— the Spy Strikes Back— Time Zone v1.1— Visible Computer: 65Ø2— Visidex— Visiterm— Zaxxon— Hayden Software— Sierra Online Software—* PLUS the complete listing of the ultimate cracking program...Super IOB 1.5— and more!

The Book Of Softkeys Volume I........160 pages

shows you how to deprotect (softkey):

Akalabeth— Ampermagic— Apple Galaxian— Aztec— Bag of Tricks— Bill Budge's Trilogy— Buzzard Bait— Cannonball Blitz— Casino— Data Reporter— Deadline— Disk Organizer II— Egbert II Communications Disk— Hard Hat Mack— Home Accountant— Homeword— Lancaster— Magic Window II— Multi-disk Catalog— Multiplan— Pest Patrol— Prisoner II— Sammy Lightfoot— Screen Writer II— Sneakers— Spy's Demise— Starcross— Suspended— Ultima II— Visifile— Visiplot— Visitrend— Witness— Wizardry— Zork I— Zork II— Zork III— PLUS how-to articles and program listings of need-to-have programs used to make unprotected backups.

COMPUTIST magazine

The Magazine For Serious Apple][users

The monthly magazine that shows you, step-by-step, how to remove copy-protection from commercial software for the Apple II series of computers.

Table of Contents

Introduction

Welcome to the **Book Of Softkeys Volume III**, a compilation of special articles from Hardcore COMPUTIST magazine (issues 11 through 15) that explain **how to remove copy-protection** from specific commercially-sold, locked-up and uncopyable software for Apple][computer systems.

Hardcore COMPUTIST (now called **COMPUTIST**) is a monthly publication devoted to the serious user of Apple][computers and compatibles. Hardcore COMPUTIST magazine contains information you are not likely to find in any of the other major journals dedicated to the Apple market.

Our editorial policy is that we do NOT condone software piracy, but we do believe that honest users are entitled to backup commercial disks they have purchased. In addition to the security of a backup disk, the removal of copy-protection gives the user the option of modifying application programs to meet his or her needs.

Furthermore, the copyright laws guarantee your right to such a deprotected backup copy:

> **..."It is not an infringement for the owner of a copy of a computer program to make or authorize the making of another copy or adaptation of that computer program provided:**
>
> **1) that such a new copy or adaptation is created as an essential step in the utilization of the computer program in conjunction with a machine and that it is used in no other manner, or**
>
> **2) that such new copy or adaptation is for archival purposes only and that all archival copies are destroyed in the event that continued possession of the computer program should cease to be rightful.**
>
> **Any exact copies prepared in accordance with the provisions of this section may be leased, sold, or otherwise transferred, along with the copy from which such copies were prepared, only as part of the lease, sale, or other transfer of all rights in the program. Adaptations so prepared may be transferred only with the authorization of the copyright owner."**
>
> United States Code title 17, §117 (17 USC 117)

Those of you who are not already familiar with Hardcore COMPUTIST are advised to read the **How To Use This Book** article in order to avoid frustration when attempting to follow a softkey article or when typing in the programs printed in this book.

How To Use This Book

Commands And Controls

Commands which a reader is required to perform are set apart from normal text by being indented and bold. An example is:

PR#6

The RETURN key must be pressed at the end of every such command unless otherwise specified. Control characters are shown as a single symbol. For example:

6 CTRL P

To complete this command, you must first type the number **6** and then hold the CTRL key while you press the **P** key. Be sure to enter the command into the computer by finally pressing the RETURN key.

Requirements

Most of the programs and softkeys which appear in this book require one of the Apple][series of computers and at least one disk drive with *DOS 3.3*. Occasionally, some programs and procedures have special requirements. The prerequisites for deprotection techniques or programs will always be listed at the beginning of the article under the '**Requirements**:' heading.

Software Recommendations

The following programs are strongly recommended for readers who wish to obtain the most benefit from our articles:

An Applesoft Program Editor
such as Call A.P.P.L.E's *Global Program Line Editor (GPLE)*.

A Sector Editor
such as SoftKey's *DiskEdit*,
or *ZAP* from Quality Software's *Bag of Tricks*,
or *Tricky Dick* from Golden Delicious' *The CIA*.

A Disk Search Utility
such as *The Inspector*,
The Tracer from *The CIA*,
or *The CORE Disk Searcher* (in this volume).

An Assembler
such as the *S-C Assembler*,
or *Merlin/Big Mac*.

A Bit Copier
such as *Copy II Plus*,
Locksmith,
or *Essential Data Duplicator (EDD)*.

A Text Editor
(able to producing normal sequential text files) such as:
Applewriter II,
Magic Window II,
or *Screenwriter II*.

You will also find *COPYA*, *FID* and *MUFFIN* from the *DOS 3.3 System Master Disk* useful.

Super IOB and controllers

Several softkey procedures will make use of a *Super IOB* controller, a small program that must be typed into the middle of *Super IOB*. The controller changes *Super IOB* so that it can copy different disks. See the ***Super IOB 1.5*** article and program in the **The Book Of Softkeys Volume II**. Before using any *Super IOB* controllers, read the ***Csaver*** article in this volume.

Reset Into The Monitor

Some softkey procedures require that the user be able to enter the Apple's System Monitor (henceforth called the Monitor) during the execution of a copy-protected program. Check the following list to see what hardware you will need to obtain this ability.

Apple][Plus - Apple //e - Apple compatibles:

1) Place an Integer BASIC ROM card in one of the Apple slots.
2) Use a non-maskable interrupt (NMI) card such as *Replay* or *Wildcard*.

Apple][Plus - Apple compatibles:

Install an F8 ROM with a modified reset vector on the motherboard as detailed in the ***Modified ROMs*** article in this volume.

Apple //e - Apple //c:

Install a modified CD ROM on the computer's motherboard. Cutting Edge Ent. (Box 43234 Ren Cen Station-HC; Detroit, MI 48243) sells a hardware device that will give you this ability. Making this modification to an Apple //c will void its warranty but the increased ability to remove copy-protection may justify it.

Recommended Literature

Apple][Reference Manual

DOS 3.3 manual

Beneath Apple DOS
by Don Worth and Pieter Lechner; Quality Software

Assembly Language For The Applesoft Programmer
by Roy Meyers and C.W. Finley; Addison Wesley

What's Where In The Apple
by William Lubert; Micro Ink

Typing In Applesoft Programs

BASIC programs are printed in this Book Of Softkeys in a format that is designed to minimize errors for readers who key in these programs. To understand this format, you must first understand the formatted **LIST** feature of Applesoft.

An illustration- If you strike these keys:

```
10 HOME:REMCLEAR  SCREEN
```

a program will be stored in the computer's memory. Strangely, this program will **not** have a LIST that is exactly as you typed it. Instead, the LIST will look like this:

```
10  HOME : REM CLEAR SCREEN
```

Programs don't usually LIST the same as they were keyed in because Applesoft inserts spaces into a program listing before and after every command word or mathematical operator. These spaces usually don't pose a problem except in line numbers which contain REM or DATA command words. The space inserted after these command words can be misleading. For example, if you want a program to have a list like this:

```
10  DATA  67,45,54,52
```

you would have to omit the space directly after the DATA command word. If you were to key in the space directly after the DATA command word, the LIST of the program would look like this:

```
    1Ø  DATA    67,45,54,52
```

This LIST is different from the LIST you wanted. The number of spaces you type after DATA and REM command words is very important.

All of this brings us to the Hardcore COMPUTIST LISTing format.

In a BASIC LISTing, there are two types of spaces: spaces that don't matter whether they are keyed or not, and spaces that **MUST** be keyed.

The latter spaces are printed here as delta characters (▵). For example:

```
1Ø  NOTE$ = "NOTE▵THAT▵THESE▵SPACES▵MUST▵BE▵ENTERED"
2Ø  VTAB 1Ø: HTAB 2Ø: PRINT NOTE$;: REM THE SPACES BETWEEN COMMANDS
     NEED NOT BE ENTERED.
```

As you see from this example, other spaces in our BASIC LISTing are put there for easier reading and it won't matter whether you type them or not.

Keying In Hexdumps

Machine language programs are printed here as both source code and hexdumps. Only one of these formats need be keyed in to get a machine language program. Hexdumps are the shortest and easiest format to type in. To key in hexdumps, you must first enter the Monitor with **CALL -151** RETURN.

Now key in the hexdump exactly as it appears. If you hear a beep, you will know that you have typed something that the Monitor didn't understand and you must retype that line.

When finished, return to BASIC by typing **EØØ3G** RETURN. Remember to BSAVE the program with the correct filename, address and length parameters as given in the article.

Keying In Source Code

The source code portion of a machine language program is provided only to better explain the program's operation. If you wish to key it in, you will need the *S-C Assembler*. Without this assembler, you will have to convert the *S-C Assembler* directives (printed in Hardcore COMPUTIST # 17) to similar directives used by your assembler.

The Softkeys

The softkeys listed below require ***Super IOB***:

Computer Preparation SAT
Demolition Division
Electronic Arts software
Hi-Res Computer Golf II
Master Type v1.7
MatheMagic
Penguin Software
Rocky's Boots
Seadragon
Snooper Troops II
Tic Tac Show
Ultima III Exodus

What is a softkey?

Softkey **is a term which we've coined to describe a procedure that removes, or at least circumvents, any copy-protection on a particular disk. Once a softkey procedure has been performed, the resulting disk can usually be copied by the use of Apple's COPYA program (on the DOS 3.3 System Master Disk) and is said to be 'COPYAable.'**

Alien Munchies

Gentry Software

Putting The Byte On Alien Munchies
by Tom Phelps
(Hardcore COMPUTIST # 15, page 7)

Requirements:

Apple][with 48K
Means of resetting into the Monitor
One slave disk with a null *HELLO* program

Alien Munchies by Gentry Software is your common, everyday, run of the mill "Fry the aliens on your barbecue grill!" arcade game. (Sure, everyday huh?) This game is slow at the beginning and patient playing is needed to get to the much more exciting 2nd and 3rd type of aliens (10,000 and 20,000 points respectively).

The method used to deprotect this game illustrates a very useful move routine you can use in your own cracks. The problem of running out of men before reaching these more challenging stages is solved with an example of the art of Advanced Playing Techniques (APTs).

Step-By-Step

1 Boot *Alien Munchies*.

2 RESET into the Monitor after the picture has come onto the screen.

3 Move page eight out of the way for a boot:

```
2000<800.8FFM
```

4 Boot a slave disk with no *HELLO* program.

5 Enter the Monitor:

```
CALL -151
```

6 Move page eight back:

```
800<2000.20FFM
```

At this point, the *Alien Munchies* program is in memory and in its proper memory locations. Rather than save a huge chunk of memory, let's save some disk space with a move routine. Furthermore, to bring the game under control a little better, let's include an APT routine.

7 Compact the code:

```
3000<6000.6FFFM
```

8 Enter the space-saving, relocatable move routine:

```
2000: A0 00 A9 00 85 00 85 02
2008: A9 30 85 01 A9 60 85 03
2010: B1 00 91 02 E6 02 E6 00
2018: D0 F6 E6 03 E6 01 A5 01
2020: C9 40 D0 EC EA
```

If you disassemble this code, it should look like this:

```
2ØØØ-    AØ ØØ        LDY #$ØØ
2ØØ2-    A9 ØØ        LDA #$ØØ
2ØØ4-    85 ØØ        STA $ØØ
2ØØ6-    85 Ø2        STA $Ø2
2ØØ8-    A9 3Ø        LDA #$3Ø
2ØØA-    85 Ø1        STA $Ø1
2ØØC-    A9 6Ø        LDA #$6Ø
2ØØE-    85 Ø3        STA $Ø3
2Ø1Ø-    B1 ØØ        LDA ($ØØ),Y
2Ø12-    91 Ø2        STA ($Ø2),Y
2Ø14-    E6 Ø2        INC $Ø2
2Ø16-    E6 ØØ        INC $ØØ
2Ø18-    DØ F6        BNE $2Ø1Ø
2Ø1A-    E6 Ø3        INC $Ø3
2Ø1C-    E6 Ø1        INC $Ø1
2Ø1E-    A5 Ø1        LDA $Ø1
2Ø2Ø-    C9 4Ø        CMP #$4Ø
2Ø22-    DØ EC        BNE $2Ø1Ø
2Ø24-    EA           NOP
```

9 Type in the following APT routine:

```
2025: 20 2F FB
2028: 20 58 FC A2 0B 20 4A F9
2030: A2 00 BD 62 20 20 ED FD
2038: E8 E0 12 D0 F5 2C 10 C0
2040: AD 00 C0 C9 D9 F0 07 C9
2048: CE D0 F5 4C 00 08 A9 EA
2050: 8D C9 10 8D CA 10 8D CB
2058: 10 A9 01 8D FD 12 4C 00
2060: 08 EA C9 CE C6 C9 CE C9
2068: D4 C5 A0 CD C5 CE A0 A8
2070: D9 AF CE A9 00
```

A disassembly of this would show:

```
2025-   20 2F FB      JSR $FB2F
2028-   20 58 FC      JSR $FC58
202B-   A2 0B         LDX #$0B
202D-   20 4A F9      JSR $F94A
2030-   A2 00         LDX #$00
2032-   BD 62 20      LDA $2062,X
2035-   20 ED FD      JSR $FDED
2038-   E8            INX
2039-   E0 12         CPX #$12
203B-   D0 F5         BNE $2032
203D-   2C 10 C0      BIT $C010
2040-   AD 00 C0      LDA $C000
2043-   C9 D9         CMP #$D9
2045-   F0 07         BEQ $204E
2047-   C9 CE         CMP #$CE
2049-   D0 F5         BNE $2040
204B-   4C 00 08      JMP $0800
204E-   A9 EA         LDA #$EA
2050-   8D C9 10      STA $10C9
2053-   8D CA 10      STA $10CA
2056-   8D CB 10      STA $10CB
2059-   A9 01         LDA #$01
205B-   8D FD 12      STA $12FD
205E-   4C 00 08      JMP $0800
2061-   EA            NOP
2062-   C9 CE         CMP #$CE
2064-   C6 C9         DEC $C9
2066-   CE C9 D4      DEC $D4C9
2069-   C5 A0         CMP $A0
206B-   CD C5 CE      CMP $CEC5
206E-   A0 A8         LDY #$A8
2070-   D9 AF CE      CMP $CEAF,Y
2073-   A9 00         LDA #$00
```

Unlike the *Sammy Lightfoot* APT in **The Book Of Softkeys Volume II**, let's make ours optional so you can play normal-style or with **infinite men** and a counter in your number of barbecues to tell you what level you're on.

10 Finally, add a JMP to our special routines before the actual program starts:

7FD:4C 00 20

11 Save the new-and-improved *Alien Munchies*:

BSAVE ALIEN MUNCHIES, A$7FD,L$5803

Computer Preparation: SAT

Harcourt, Brace and Jovanovich, Inc.

Deprotecting Computer Preparation: SAT
by Eddie Fang
(Hardcore COMPUTIST # 14, page 6)

Requirements:
48K Apple][, Apple][Plus, or Apple //e
One disk drive with DOS 3.3
Four blank disks
Super IOB and the swap controller
(Optional: File transfer program)

This computer preparation package by HBJ ranks up at the top with Barron's computer preparation course for the SAT. HBJ's package has a lot of the major words found on the SAT and a lot of similar mathematical and verbal problems, too.

My problem, however, is that I would like to change the words and problems for a younger brother or change the problems that I have finished studying to some newer ones. The frustration of copying this disk is immense. After trying all of the major bit copiers to no avail, I decided to try *Super IOB*. Here is how I did it!

First, get the disk that is labeled *A*. All the disks follow the same process, except you will have to modify different programs on different disks to get each totally broken. Follow these steps carefully:

1 Boot the first protected disk. When you hear the drive head access the first track, hit RESET.

2 Get into the Monitor:

```
CALL -151
```

3 We must move the RWTS down to a safe location for *Super IOB* to use:

```
1900<B800.BFFFM
```

4 When you get the Monitor prompt again, insert your *Super IOB* slave disk and boot it:

```
C6ØØG
```

5 Save the RWTS:

```
BSAVE RWTS.SAT,A$19ØØ,L$8ØØ
```

6 Install the controller at the end of this article (a modified version of the swap controller) into *Super IOB* and execute *Super IOB*:

```
RUN
```

7 When asked if you wish to format the backup first, reply with a Y. This will put DOS on the disk and set the bootup program to HELLO.

8 Copy the other three sides in the same manner (Steps six and seven).

Almost Finished

You now have a COPYAable version, but if you want to modify it, you will have to change a few line numbers of some of the BASIC programs. Several of the main programs have a line number Ø which looks something like this:

```
Ø REM CTRL H CTRL H CTRL H CTRL H CTRL H CTRL H CTRL H CTRL H CTRL M CTRL D FP
```

Whenever you try to list a line like this, DOS sees the CTRL M CTRL D FP and clears the program in memory. In other words, the program self-destructs when LISTed. This situation is easily circumvented by merely eliminating line zero from these programs. The general format for doing this is:

```
LOAD filename
DEL Ø,1
SAVE filename
```

Remove line zero from these programs on your **duplicate** disk:

Math Item Bank Display on the mathematical side.
Vocabulary Flashcards on the vocabulary flash cards side.
Verbal Item Bank Display on the verbal side.
Frank (I don't know why they called it that) on the diagnostic and testing side.

Closing Comments

You should now have four bootable, completely deprotected and modifiable *Computer Preparation: SAT* disks. I hope this helps you as much as it has helped me.

```
41Ø GOSUB 8Ø : HOME :A$ = "FORMATTING" : FLASH : GOSUB 45Ø : NORMAL
    : PRINT : PRINT CHR$ (4 ) "INITHELLO,S" S2 ",D" D2 ",V" VL :VL =
    Ø : RETURN
1ØØØ REM SWAP CONTROLLER
1Ø1Ø TK = 3 :ST = Ø :LT = 35 :CD = WR
1Ø2Ø T1 = TK : GOSUB 49Ø : GOSUB 36Ø : ONERR GOTO 55Ø1Ø3Ø GOSUB 43Ø :
    GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø3Ø
1Ø4Ø IF BF THEN 1Ø6Ø
1Ø5Ø ST = Ø :TK = TK + 1 : IF TK < LT THEN 1Ø3Ø
1Ø6Ø GOSUB 49Ø :TK = T1 :ST = Ø : GOSUB 36Ø
1Ø7Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø7Ø
1Ø8Ø ST = Ø :TK = TK + 1 : IF BF = Ø AND TK < LT THEN 1Ø7Ø
1Ø9Ø IF TK < LT THEN 1Ø2Ø
11ØØ HOME : PRINT "COPY^DONE" : END
1ØØ1Ø IF PEEK (64ØØ ) < > 162 THEN PRINT CHR$ (4 ) "BLOAD^
    RWTS.SAT,A$19ØØ"
```

DLM Software*

(Development Learning Materials)

Softkey For DLM Software
by Chris Chenault & Ray Darrah
(Hardcore COMPUTIST # 13, page 7)

*Specifically for: *Demolition Division, Alligator Mix, Alien Addition,* and *Minus Mission.*

Requirements:
48K Apple][or Apple //e
Super IOB
A blank disk

Demolition Division

Demolition Division is a drill that makes division fun for youngsters. In this game, you use your correct answers to shoot enemy tanks. You have a choice of speed, difficulty and paddle or keyboard control.

Alligator Mix

Alligator Mix is a math drill that consists of a mixture of addition and subtraction problems. In this game, you can only feed the alligator if the answer to the addition or subtraction problem matches the answer he has on his tummy.

Alien Addition

Alien Addition is an addition drill made into an arcade game. The game gives you the ability to change speed and difficulty levels and is one of many great education games that offer children fun while they learn.

Minus Mission

In *Minus Mission*, falling blobs of subtraction problems threaten to overcome a robot with green slime. Arcade skill is required as well as subtraction skill to save the robot.

The Softkey To Them All

For all of these great programs, I have developed a *Super IOB* controller that deprotects them nicely. I think it may work on other DLM releases as well as the four mentioned above.

In any case, here are the steps to follow when deprotecting any of these educational masterpieces:

1 Type in the *Super IOB* controller at the end of this article and save it.

2 Next, boot a normal DOS diskette and initialize a blank disk with the filename the same as the disk you wish to copy.

INIT *program name*

3 Execute the *Super IOB* program with the DLM controller installed.

4 When *Super IOB* is finished, try to boot the copy. If everything comes up OK then you're finished. If you get a *FILE NOT FOUND*, CATALOG the disk and RUN the first file you see. This should have been the program name you used in step 2. You may go back to step 2 and try again if you like.

controller

```
1ØØØ REM DLM SOFTWARE CONTROLLER
1Ø1Ø TK = 3 :ST = Ø :LT = 35 :CD = WR :MB = 13Ø :DOS = 13
1Ø15 GOSUB 36Ø : GOSUB 27Ø : GOSUB 36Ø
1Ø2Ø T1 = TK : GOSUB 49Ø : GOSUB 36Ø : ONERR GOTO 55Ø
1Ø3Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø3Ø
1Ø4Ø IF BF THEN 1Ø6Ø
1Ø5Ø ST = Ø :TK = TK + 1 : IF TK < LT THEN 1Ø3Ø
1Ø6Ø GOSUB 49Ø :TK = T1 :ST = Ø : GOSUB 36Ø
1Ø7Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø7Ø
1Ø8Ø ST = Ø :TK = TK + 1 : IF BF = Ø AND TK < LT THEN 1Ø7Ø
1Ø9Ø IF TK < LT THEN 1Ø2Ø
11ØØ HOME : PRINT "EVERYTHING^O.K.^NO^DOS^ON^COPY" : END
1ØØ1Ø PRINT CHR$ (4 ) "BLOAD^RWTS.13,A$19ØØ"
```

Electronic Arts Software*

Deprotecting Electronic Arts
by Pete Levinthal
(Hardcore COMPUTIST # 13, page 26)

*Softkeys for: *Cut And Paste, The Last Gladiator*, and *One on One*.

Requirements:
48K Apple][, Apple][Plus, or Apple //e
DOS 3.3 disk drive
Super IOB
A sector editor
A blank disk

Electronic Arts' recent releases have all used very nearly the same protection. Their protection scheme has been to change the data field prologue bytes from a normal **D5 AA AD** to a modified **D5 BB CF** on tracks $Ø3 to $2Ø. Tracks $ØØ to $Ø2 contain the modified RWTS to read the new data field bytes and hi-res title page. These three tracks are unprotected normal DOS 3.3. In addition, these recent releases have a nibble count on track $22, and track $21 is unused.

Remember that the prologue data field bytes tell DOS where the data starts on a sector. This is usually identified by the unique sequence of bytes **D5 AA AD**. Electronic Arts has modified these to **D5 BB CF** so normal DOS cannot tell where the data starts and hence, an I/O error occurs when copying with *COPYA* (or some bit copiers for that matter).

So, to deprotect the new Electronic Arts releases we must:

1 Read the original disk with data field prologue bytes of **D5 BB CF**.

2 Write to a normal DOS 3.3 disk with normal data field prologue bytes of **D5 AA AD**.

3 Change the modified Electronic Arts' RWTS to read normal **D5 AA AD** data field prologue bytes.

4 Disable the nibble count.

Easy enough, right? We can use *Super IOB* to do most of the work and then use a sector editor to complete the process. Here is the procedure:

1 Load *Super IOB* and type in the controller at the end of this article.

2 With the controller installed, run *Super IOB*.

Cut and Paste

When *Super IOB* is finished, it will have made these sector alterations:

Track	Sector	Byte	From	To
01	0F	68	20	18
01	0F	69	A2	60
01	0F	6A	A1	EB
02	03	47	BB	AA
02	03	51	CF	AD

If you have copied *Cut and Paste*, then that's all. Enjoy your backup.

Pull Out Your Sector Editor

If you have copied *The Last Gladiator* or *One on One*, then you will need to pull out your sector editor and make the following modifications (to the copied COPYAable disk):

The Last Gladiator

Track	Sector	Byte	From	To
1F	0E	05	A0	18
1F	0E	06	20	60
1F	0E	68	20	18
1F	0E	69	A2	60
1F	0F	05	A0	18
1F	0F	06	20	60
1F	0F	68	20	18
1F	0F	69	A2	60

One On One

Track	Sector	Byte	From	To
Ø9	Ø2	1F	Ø1	FD
ØC	Ø4	Ø5	AØ	18
ØC	Ø4	Ø6	18	6Ø
ØC	Ø4	Ø7	88	C8
ØC	Ø4	DC	AØ	18
ØC	Ø4	DD	FF	6Ø

Now you're all done! Don't forget to write the sectors back out to your *COPYA*able copy as you change them.

Electronic Arts controller

```
1ØØØ REM ELECTRONIC ARTS
1Ø1Ø TK = Ø :ST = Ø :LT = 33 :CD = WR
1Ø2Ø T1 = TK : GOSUB 49Ø : IF TK > 3 THEN RESTORE : GOSUB 21Ø
1Ø3Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø3Ø
1Ø35 IF TK = 2 THEN GOSUB 21Ø
1Ø4Ø IF BF THEN 1Ø6Ø
1Ø5Ø ST = Ø :TK = TK + 1 : IF TK < LT THEN 1Ø3Ø
1Ø6Ø GOSUB 49Ø :TK = T1 :ST = Ø : GOSUB 23Ø : IF TK = Ø THEN GOSUB
     111Ø
1Ø7Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø7Ø
1Ø8Ø ST = Ø :TK = TK + 1 : IF BF = Ø AND TK < LT THEN 1Ø7Ø
1Ø9Ø IF TK < LT THEN 1Ø2Ø
11ØØ HOME : PRINT "DONE^WITH^COPY" : END
111Ø POKE 19Ø15 ,17Ø : POKE 19Ø25 ,173 : POKE 18Ø24 ,24
112Ø POKE 18Ø25 ,96 : POKE 18Ø26 ,235 : RETURN
62Ø1Ø DATA 213 ,187 ,2Ø7
```

The Einstein Compiler v5.3

The Einstein Corporation

Softkey For The Einstein Compiler Version 5.3
by Marco Hunter
(Hardcore COMPUTIST # 11, page 6)

Requirements:
48K Apple][Plus or equivalent
One disk drive with DOS 3.3
A blank disk
COPYA from the *DOS 3.3 System Master*
A sector editing program

Although it has a few shortcomings, the *Einstein Compiler* is probably the best of its kind available. The compiler produces a file which is usually saved as a huge Applesoft program and tricks DOS into thinking it is loading an Applesoft file, when it is actually loading a compiled program as well as a library of subroutines. But that is for a review, and this is a softkey. Suffice it to say that *Einstein* is most likely your best bet in Applesoft compilers.

Einstein is a program which resides on an essentially normal disk. The infamous nibble-count technique is used to ensure that an original disk is being used. The procedure for locating a nibble count usually involves much time, knowledge of Assembly language and Assembly language tricks. A little luck can also come in handy. Fortunately, all nibble counts involve some method of accessing the disk. Since a disk controller card can reside in any of the Apple's slots, the nibble count must adjust itself to the slot being used. The most popular method is to load the X- or Y-register with the slot number (technically, the slot number * 16) and then to access the location C08C + X or C08E + X. In Assembly language, it would look like one of these.

```
     LDA $C08C,X                              LDA $C08E,X
     LDA $C08C,Y                              LDA $C08E,Y
```

Note: Other options include LDY $C08E,X and LDX $C08C,Y.

Since both the register loaded and the index used can change, it is best to search for C08C or C08E. Because two-byte addresses are always reversed in machine language, the bytes to search for are 8C C0 and 8E C0. After locating these bytes on a disk, preferably with a disk search utility such as *Bag of Tricks* or *The CIA*, you should disassemble the code to find out if it is truly a nibble count. Try to avoid searching tracks 0-2 because these generally contain DOS or some type of RWTS which are usually full of 8CC0's, but not nibble counts. Many times the code around a nibble count will contain many PLA's and PHA's. The software companies hope that by playing around with the stack they can fool most people.

Once you have located a nibble count, you can eliminate it in several ways. You could either put an RTS (return from subroutine) at the beginning, NOP (no-operation does nothing) the entire routine, jump out of it, or avoid jumping into it in the first place. Keep in mind, however, that some companies also protect their disks by including checksum routines which can detect the presence of altered code. Sierra On-Line is one company known to do this (See ***SOL: Sierra OnLine Software*** article on page 108 of **The Book Of Softkeys Volume II**).

Of course, some knowledge of Assembly language is an invaluable aid when tracking down protection schemes. If you don't happen to know any Assembly language yet, it would be well worth your time to pick up a good book on it. Assembly language is not as mysterious and hard to learn as many people would have you believe. If you can learn the hexadecimal number system and condition yourself to think in it when necessary, half the battle of learning Assembly language will be behind you. Even if you never write a single Assembly language program, just being able to follow code written by others is a valuable skill.

Well, enough talk. On to the process!

I located the bytes on *Einstein* and figured out a way to jump around the routine, so that everything went as normal.

1 Make a copy of *Einstein* with COPYA.

2 Use your sector editor to modify:

Track	Sector	Byte	From	To
08	04	2A	BD	4C
08	04	2B	8C	E2
08	04	2C	C0	91

Escape From Rungistan

Sirius Software

Softkey For Escape From Rungistan
by Chris Chenault
(Hardcore COMPUTIST # 15, page 6)

Requirements:
48K Apple][or Apple //e
One disk drive with DOS 3.3
A sector editor

Escape from Rungistan is a unique adventure featuring graphics, sound and animation. At the beginning of the game, you awake in a foreign prison and hear the guard say that you are to be shot at sunrise. There is no other choice but to attempt to break out. As the game progresses, you battle bears, snakes and test your skiing ability.

This adventure is written with an 'Old West' flavor and the author shows a great sense of humor. To aid your efforts at playing this game, the author added a clue file to keep you from getting too flustered.

Escape With Diskview

The locking procedure on this disk consists of a combination of many fairly simple procedures, but together they give you endless trouble. I used *DiskView* to discover that this is a DOS 3.3 disk with the last byte of the address prologue marker changed to $F7 from $96 on tracks $03-$22.

Using this pre-analysis, it seems to be a disk that a slightly modified *COPYA* would deprotect. Unfortunately, because some of the last few tracks have been damaged for protection, this won't work.

I finally came up with the following method after discovering that it had a normal CATALOG on the correct track and a greeting program called *START*. I discovered this with the help of *DiskEdit*. Now for the procedure.

Step-By-Step

1 Boot the *DOS 3.3 System Master* disk.

2 Insert a blank disk and format it with *START* as the boot (greeting) file:

```
INIT START
```

3 Re-insert *DOS 3.3 System Master* and load *FID*:

```
BLOAD FID
```

4 Write-protect your *Escape From Rungistan* disk.

5 Drop into the Monitor and modify DOS so that the last byte of the address prologue is ignored:

```
CALL —151
B969:29 ØØ
```

6 Start *FID* going:

```
8Ø3G
```

7 Copy all the files (using the = wildcard feature of *FID*) to the disk you INITed in step 2.

8 When *FID* informs you that a file named *START* already exists, hit RETURN so that we will get the file from the *Rungistan* disk.

The copy at this point won't work because the DOS on the original disk has strange DOS commands. This is how they have been altered:

Normal Command	Rungistan Command
RUN	ARC
CLOSE	SVRTT
READ	DNRT
OPEN	CBSE
MAXFILES	FILMAXES
BSAVE	AVESB
BLOAD	ODABL
ALL OTHERS	*rubbed out*

9 Use your sector editor to read track $Ø1, sector $Ø7 from the original disk and write it to your new unlocked disk. This changes the commands to *Rungistan* commands).

You're done! To CATALOG your new disk, study the code and do APTs, simply boot a normal disk before starting. Your copy of *Escape From Rungistan* is COPYAable.

Financial Cookbook

Electronic Arts

Deprotecting The Financial Cookbook
by Pete Levinthal
(Hardcore COMPUTIST # 15, page 6)

Requirements:

Apple][
A *Replay* card, *Wildcard* or other 48K copy card
A blank disk
COPYA from the *DOS 3.3 System Master*
A sector editor

Here is a quick-and-dirty method to copying the *Financial Cookbook* from Electronic Arts. This method requires a copy card such as the *Replay* or *Wildcard.*

Since the *Financial Cookbook* requires 64K of memory to run, it is no big deal to use a copy card to load the program back in. The trick to this method is to copy memory at just the right time. The object is to escape the protection but catch the program before it starts loading into the RAM card (thus preventing us from having to copy 64K of memory).

The *Cookbook* disk is completely copyable with *COPYA* except for track 6, which is the nibble count track. The boot proceeds like any other Electronic Arts release: it loads a title page and then does a nibble count and loads the program. Listen to the program load and you will hear the nibble count (sounds weird, eh?). We want to copy memory after the nibble count but before the RAM card is activated. Just after the nibble count, the drive will stop and the text page will flutter for a second. Press the copy card switch at this instant. You will only have a second or two to do it. You may have to practice a few times.

After this process, use your copy card utility disk to make binary files of the memory.

Now copy tracks $12 to $14 of the original *Financial Cookbook* to a blank initialized disk using a slightly modified *COPYA* then run a sector editor and change bytes $80 to 00 00, bytes $84 to

00 00, bytes $88 to 00 00, bytes $3C to 00 00 and bytes $40 to 00 00 to allocate tracks $12 to $14 and the DOS tracks as used in the VTOC. Finally, copy the binary (B) files of memory to this disk using *FID* or some other utility.

You're all done! Just BRUN the memory files and, if all went correctly, the program will restart, read in some data from tracks $12 to $14 and all is fine.

Step-By-Step

1 Boot the *Financial Cookbook* and after the title page and the nibble count, and just after the drive stops (for just a second!), hit the copy card switch.

2 Copy all 48K of memory.

3 Process the copied memory using your copy card utility disk to create normal binary files from it.

4 Start *COPYA* going, then break it at the title page:

```
RUN COPYA
CTRL C
```

5 Enter the Monitor:

```
CALL —151
```

6 Make *COPYA* so that it will only copy specific tracks:

```
302:16
35F:16
2DE:20 B0 02
2B0:A9 0F 8D D1 02 8D D2 02 60
3D0G
70
RUN
```

7 When the copy is done, get out your sector editor and make the following changes to your freshly created disk:

Track	Sector	Bytes	To
$11	$00	$80, $81	$00, $00
$11	$00	$84, $85	$00, $00
$11	$00	$88, $89	$00, $00
$11	$00	$3C, $3D	$00, $00
$11	$00	$40, $41	$00, $00

8 Finally, transfer the memory files to this disk using *FID*. Now you're all done! Just BRUN the memory files to restart the program.

Flip Out

Sirius Software

Deprotecting Flip Out
by Clay Harrell
(Hardcore COMPUTIST # 12, page 11)

Requirements:
48K Apple][or Apple][Plus with old-style F8 Monitor ROM
One blank initialized DOS 3.3 disk

Flip Out is a hi-res strategy game from Sirius Software (R.I.P.) which requires you to send all ten of your marbles through the playing course before your opponent does. Each player starts with ten of his opponent's marbles and then takes turns dropping these marbles into the *Flip Out* playing field (of which there are many variations). Your first goal is to trap your opponent's marbles in a spot where they will be difficult to recover. Each marble dropped may cause a chain reaction, so some strategy is required. After the players have dropped all ten of their opponent's marbles, they begin to drop their own marbles through the course. This continues until one of players wins by getting all his marbles through the course.

Who Stole the ROMS?

Flip Out is well done and challenging! But after I bought it and was done playing a game, I hit RESET and did not see the usual Monitor prompt (I have an old-style F8 Monitor ROM on the motherboard). My computer rebooted as if I had a new-style F8 Monitor ROM! This intrigued me into investigating this strange phenomenon.

What I discovered was that the main program (not the boot code) will copy an image of the new-style F8 Monitor ROM into a slot 0 RAM card if one is found. Of course, I had a RAM card and they had it turned ON instead of the motherboard ROMs. It is easy to understand what they are doing if you just remember what memory the RAM card occupies.

The RAM card occupies memory from $D000 to $FFFF. This may seem strange since the motherboard ROMs (Applesoft and the monitor) also occupy $DØØØ to $FFFF. However, there are a set of soft switches that can turn ON motherboard ROMs or turn ON the RAM card. An example of this occurs when you boot your 48K DOS 3.3 system master and it loads Integer BASIC into the RAM card. Now you have two languages available that occupy the same logical memory space, $DØØØ to $FFFF. You can switch between the two with INT and FP. When the INT command is typed in, the soft switches are thrown so that the language card is read-enabled. Likewise, the FP command read-enables the motherboard ROMs.

If you read your RAM card manual you can see that the softswitch at memory location $CØ8Ø (assuming your RAM card is in slot zero) will allow you to look at your RAM card's memory. But have you ever tried typing $CØ8Ø from the Monitor? It will lock up your Apple, requiring you to power OFF and then back ON again to recover.

Recovering the ROMS

This phenomenon occurs because you have switched to the RAM card's memory $DØØØ—$FFFF and turned OFF the motherboard's ROMs. When you do this, the computer loses control since there is no longer a Monitor ROM available from $F8ØØ—$FFFF to oversee your Apple's operations! You cannot recover from this condition, even with a RESET.

To get around this problem you must first type:

```
CØ81 CØ81 N F8ØØ<F8ØØ.FFFFM
```

from the Monitor. This reads the motherboard's ROM, but allows you to write to bank 2 of the RAM card. It moves the F8 motherboard ROM into $F8ØØ—FFFF in bank 2 of the RAM card! Now you may type CØ8Ø to turn ON the RAM card and look at its memory, since a copy of the $F8 monitor ROM is in the RAM card from $F8ØØ to $FFFF.

This is what Sirius and other software publishers will sometimes do to prevent people from utilizing their RAM cards for deprotection purposes. With a RAM card in slot Ø, no matter what $F8 ROM you have in the motherboard, your Apple will only look at the new style $F8 ROM image in your RAM card. Thus, the Apple clears memory and reboots when RESET is pushed! This is an easy problem to fix, now that we have identified it. Just take your RAM card out of your computer and you may RESET into the Monitor as usual. As demonstrated in the softkey for *Sensible Speller* (see article on page 82 of this volume of **The Book Of Softkeys**), you can sometimes get away with moving your RAM card to a slot other than Ø where the program does not expect to find a RAM card.

Flip Out is a single-load program. To deprotect single-load programs, there are 3 basic things which we need to determine: **1.** What memory is used by the program. **2.** The starting address of the program. 3. How to get the memory saved to a normal DOS 3.3 disk and reloaded back into memory in the proper place(s).

Keep these three items in mind as we snoop through *Flip Out* or any other single load game.

Sirius has changed its protection schemes a lot in the last few years. The height of their protection mania was demonstrated in games like *Bandits* and *Fly-wars*. The problem with hi-tech protection is that a program might not boot on a Rana drive, or on an Apple //e or some other flavor of Apple. Sophisticated protection schemes are also costly and drive up the retail price of software. In light of this, Sirius chose a much simpler, but still effective, copy-protection scheme for *Flip Out*. Just try and copy it with *Nibbles Away* or another bit copier and you'll see just how effective it is!

A Bit of Boot Tracing

Keeping in mind the three things we must figure out to deprotect a single-load game, the first thing I generally do is to find what memory is required to run the program. To do this we can flip through memory (remember that shape tables, etc. don't disassemble into meaningful code), or we can trace the boot and see where the program gets loaded to. I prefer to trace the boot when it is fairly simple, which it is on *Flip Out*.

So, boot up normal DOS 3.3 (so we can save a piece of code for later examination, if you wish) and enter the Monitor with:

```
CALL —151
```

Now we must copy the code in the disk controller ROM down to RAM so we can modify it to our liking. Do this by typing:

```
8600<C600.C6FFM
```

from the Monitor. Now we have the disk controller ROM code where we can modify it, and start to trace the boot.

If you did not know already, the disk controller ROM reads track zero, sector zero into memory from $800 to $8FF. Then it JMPs to $801 and starts executing the code (which continues by loading in a little more code, which then loads in more code, which then.....well, you get the idea).

At the end of the code at $8600 we see a JMP $0801. We must change this to JMP $FF59, which will exit us in the Monitor after it is done loading track zero, sector zero into $8000 to $8FF. Put *Flip Out* in drive one and type:

```
86F9:59 FF N 8600G
```

The drive will recalibrate and, a second later, beep into the Monitor just like we told it. To turn OFF the drive motor, type:

```
C0E8
```

from the Monitor prompt. If you want to, you can save this hunk of code to your normal DOS 3.3 disk with:

BSAVE BOOT0,A$800,L$100

(since this process did not disturb DOS which lives from $9600 to $BFFF). Now type:

801L

to flip through the code just loaded. Here's what you should find:

```
801-    A5 2B       LDA $2B
803-    AA          TAX
804-    85 FB       STA $FB
806-    4A          LSR
807-    4A          LSR
808-    4A          LSR
809-    4A          LSR
80A-    09 C0       ORA #$C0
80C-    8D 00 30    STA $3000
80F-    A0 00       LDY #$00         ---------------------
811-    84 00       STY $00
813-    A9 D0       LDA #$D0
815-    85 01       STA $01          Put the
817-    A2 30       LDX #$30         code from
819-    AD 81 C0    LDA $C081        ROM into
81C-    AD 81 C0    LDA $C081        the slot
81F-    B1 00       LDA ($00),Y      zero RAM
821-    91 00       STA ($00),Y      card.
823-    C8          INY
824-    D0 F9       BNE $081F
826-    E6 01       INC $01
828-    CA          DEX
829-    D0 F4       BNE $081F        ---------------------
82B-    A6 FB       LDX $FB          Load code
82D-    84 F7       STY $F7          into RAM
82F-    A9 04       LDA #$04         starting
831-    85 F8       STA $F8          at page
833-    85 FA       STA $FA          $04
835-    BD 8C C0    LDA $C08C,X      ---------------------
838-    10 FB       BPL $0835
83A-    C9 AD       CMP #$AD         Look for
83C-    D0 F7       BNE $0835        a data
83E-    BD 8C C0    LDA $C08C,X      field
841-    10 FB       BPL $083E        epilogue
843-    C9 DA       CMP #$DA         of AD,
845-    D0 F3       BNE $083A        DA, DD
847-    BD 8C C0    LDA $C08C,X
84A-    10 FB       BPL $0847
84C-    C9 DD       CMP #$DD
84E-    D0 EA       BNE $083A        ---------------------
88A-    4C 29 04    JMP $0429        JMP BOOT2
```

This code loads in the final loader *BOOT2* over the text screen memory ($4ØØ to $7FF) and JMPs to $429. Now we need to examine *BOOT2* (the game loader) to see where it actually loads in the game.

You might notice that this is slightly difficult since *BOOT2* gets loaded over the text page, and when we hit RESET this memory pretty much hits the bit-bucket. But *BOOT1* (the code which we are now looking at in $8Ø1—$88C) can be changed to load *BOOT2* somewhere else and gracefully RESET into Monitor. To do this we can change the load byte from page $Ø4 to page $14, and change the JMP $429 to jump into the Monitor. Then we can examine the *BOOT2* loader. To do this, from the Monitor, enter:

```
83Ø:14 N 88B:59 FF
```

The next thing we must change is the disk controller ROM code at $86ØØ. We need it to execute, but not write over, the modified code at $8Ø1. To do this, we can tell it to load track zero sector zero at $6ØØØ (instead of $8ØØ) and jump to our modified code at $8Ø1. Of course our code will load *BOOT2* into $14ØØ so we can look at it. If you are, go back and don't return until you understand what is going on.

OK, put *Flip Out* in drive one and type:

```
8659:6Ø N 86F9:Ø1 Ø8 N 86ØØG
```

The drive will recalibrate and boot zero will read track zero, sector zero into $6ØØØ (thus, not overwriting our code at $8Ø1). It will then jump to $8Ø1 (*BOOT1*) and load *BOOT2* into $14ØØ to $17FF. Unfortunately, it will keep reading *BOOT2* into $14ØØ to $17FF because we haven't changed enough code. So, after a few seconds, hit RESET.

Now if you want to, you can put your normal DOS 3.3 disk in a drive and save *BOOT2* with **BSAVE BOOT2,A$14ØØ,L$4ØØ**.

Next type **1429L** and examine *BOOT2*. You will notice that memory from $8ØØ to $BFF gets wiped clean and that a reset error routine gets moved to $8FØØ—$8F8Ø. This is a good indication that *Flip Out* lives from $CØØ to $8F8Ø! I'll let you sort through the *BOOT2* code to find out for sure, or you can take my word for it.

The last tidbit of information that the *BOOT2* loader reveals is the starting location of *Flip Out*. Look at the code at $17CC—$17E4 and you'll see how it wipes out memory from $8ØØ to $BFF and then JMPs to $78ØØ to start the game.

Now we have filled requirements one and two. All that is left is to save the memory from $CØØ to $8F8Ø on a normal DOS 3.3 disk. This is easy since a 48K slave disk does not destroy memory from $9ØØ to $96FF. So just boot *Flip Out* and, when the drive stops, RESET into the Monitor. Now boot your 48K slave disk and save *Flip Out* to disk! In cook-book fashion, here are the steps necessary to get a BRUNable version of *Flip Out*.

The Steps

0 Turn your Apple **OFF** and remove your RAM card.

1 Boot *Flip Out* then after the drive stops and *Flip Out* is loaded into memory, hit RESET to enter the Monitor.

2 Boot a 48K slave disk:

```
PR#6
```

3 Enter the Monitor by typing:

```
CALL-151
```

4 So that the program will execute when BRUN, enter some code which JMP's to $7800 to start up the game:

```
BFD:4C 00 78
```

5 BSAVE *Flip Out* by typing:

```
A964:FF
BSAVE FLIP-OUT,A$BFD,L$8383
```

If you want the title page displayed, you will also have to perform the following steps:

6 Reboot the *Flip Out* disk and RESET into the Monitor when you see the title page.

7 Boot your 48K slave disk and BSAVE the picture by typing:

```
PR#6
BSAVE PIC,A$2000,L$1FFB
```

8 BLOAD the *Flip Out* file and the picture file:

```
BLOAD FLIP-OUT
BLOAD PIC,A$2000
```

9 Enter the Monitor and type in the following code which will display hi-res page 1 and wait for a key to be pressed before it JMP's to the start of the game:

```
CALL -151
BE0:AD 10 C0 AD 50 C0 AD 54
BE8:C0 AD 57 C0 AD 52 C0 AD
BF0:00 C0 10 FB 4C 00 78
```

10 BSAVE the file by typing:

```
A964:FF
BSAVE FLIP-OUT,A$BE0,L$83A0
```

Hi-Res Computer Golf II

Avante-Garde Creations

Using Super IOB to Copy Hi-Res Computer Golf II
by Jeff Rivett
(Hardcore COMPUTIST # 12, page 6)

Requirements:
Apple][Plus or equivalent
Super IOB and the swap controller
Two blank disks

Using the following method will help you to remove the copy protection from a lot of older software. Although simple, it's a great solution when your program is constantly accessing the disk and the DOS is reasonably close to DOS 3.3. It's well worth trying in many cases, because it is so easy.

How it works

First the original disk is booted to get the copy-protected DOS into the machine. This is the only tricky part because you have to have some means of stopping the execution of the program so that the Apple's Monitor can be entered. In Avant-Garde's *Golf II*, the Reset vector is set to re-boot the disk. However, if you lift the drive door prior to hitting RESET twice, the disk drive will shut OFF and the protected DOS will still be in memory. The RWTS (Read/Write Tracks & Sectors) of the protected DOS can then be moved to a safe location prior to booting up with a normal DOS disk. The protected RWTS can then be saved and utilized by *Super IOB*, with the swap controller installed, to read the protected disk. For Golf, you only have to leave the disk drive door open to capture the RWTS, but for most other programs you will need an old-style monitor F8 ROM or one of the Non-Maskable Interrupt (NMI) copy cards to do the trick.

When *Super IOB* has the swap controller installed, it will load in a protected disk's RWTS at $1900-$2100. When it comes time to read the protected disk, this RWTS is moved into $B800—$BFFF. For writing, the normal RWTS is moved back into $B800—$BFFF.

In other words, the disk is deprotected by reading it with a protected RWTS and writing the copy with a standard RWTS.

1 Boot the *Golf II* disk and when the *DO YOU WANT TO SEE THE INTRO?* prompt appears, open the disk drive door and hit RESET. Hit RESET a second time to turn OFF the drive. The *Golf II* DOS will still be in memory.

2 Enter the Monitor and move the *Golf II* RWTS to $1900:

```
CALL -151
1900<B800.BFFFM
```

3 Boot up with a slave DOS 3.3 disk which has a small or null *HELLO* program:

```
C600G
```

4 Insert the disk which has *Super IOB* on it and BSAVE the *Golf* RWTS:

```
BSAVE GOLF.RWTS,A$1900,L$800
```

5 Load in the *Super IOB* program:

```
LOAD SUPER IOB
```

6 Type in or EXEC in the swap controller and make sure that the file BLOADed in line 10010 is called *GOLF.RWTS*):

7 Run *Super IOB* and copy both sides of the *Golf* disk. The *Golf* DOS should **NOT** be copied.

Golf II has a strange bug that can be fixed by adding a line to the Applesoft file called *SWING.PRACTICE.FP*. To fix this bug, LOAD the file and add this line:

```
2 POKE 16611 ,1
```

If you use the technique I have just outlined on another disk and it seems to work (except for the fact that the copied program will not run), you may be able to figure out what is going on by tracing the now readily accessible code and making modifications where necessary. This is something that cannot be taught. Sometimes the modifications will be quite simple, like adding a POKE statement to an Applesoft program or omitting portions of an Applesoft or machine language file.

If you happen to have a *Replay II* card, you can make a copy of the program at the title page and another copy with normal DOS 3.3 in the machine. You can then use the 'Compare' option on the *Replay II* utility disk to compare the copy-protected DOS with normal DOS 3.3. This is often very helpful in making patches that allow the new copy to run properly.

Knoware

Knoware, Inc.

Softkey for Knoware
by Doni G. Grande
(Hardcore COMPUTIST # 14, page 6)

Requirements:
Apple][Plus with 64K RAM
A way to RESET into the Monitor
COPYA
A sector editor
Three blank disks

Knoware is a very interesting game - simulation - educational program. The player starts out in a company working in the mailroom. The objective is to become chairman of the board. In order to accomplish this, you must use computer application programs for spreadsheet analysis, wordprocessing, and database use. All of the application programs may be used on their own, so once you make chairman of the board, you have learned how to use all of the programs!

First of all, only the first (out of three) disk is protected. Secondly, *Knoware* checks the disk in the drive to be sure it is an original. However, it is fairly easy to short-circuit this check. The procedure to make a *COPYA*able version of *Knoware* is as follows:

1 Use *COPYA* and duplicate the original Disk #1.

2 Use a sector editor to make the following changes:

Track	Sector	Byte	From	To
$17	$01	$14	$4C	$A9
$17	$01	$15	$8D	$00
$17	$01	$16	$20	$EA

This changes the file *ONESHOT.OVR* (A$2000,L$FBA), and prevent a call to a protection-check subroutine by replacing the JSR $208D with a LDA #00 and a NOP, which tricks the program into thinking there is an original disk in the drive.

3 Now use *COPYA* to copy Disks 2 and 3, and you have the complete set.

One modification that you might want to make is to allow a fast DOS to be used. There is a check in the file *ONESHOT.OVR* to see if the I/O hooks have been changed. Some fast DOS versions do change these. The following change will defeat this check:

Track	Sector	Byte	From	To
$10	$0F	$01	$D0	$EA
$10	$0F	$02	$0C	$EA
$10	$0F	$05	$C9	$A9
$10	$0F	$06	$9E	$00

An Alternate Procedure

This procedure may not work on all versions of *Knoware*. This is an alternate way to unlock *Knoware*.

1 Use *COPYA* and copy the original disk #1.

2 Insert the **copy** of disk #1 into drive 1 and type the following:

```
CALL -151
BLOAD ONESHOT.OVR
2010:A9 00 EA
BSAVE ONESHOT.OVR, A$2000,L$FBA
BLOAD BOOTKW
9FD:EA EA
A01:A9 00
BSAVE BOOTKW,A$800,L$852
```

3 use *COPYA* to make copies of disks #2 and #3.

Laf Pak

Sierra On Line

Normalizing Laf Pack
by Ferrel Wheeler
(Hardcore COMPUTIST # 13, page 9)

Requirements:

Apple][
1 blank disk
1 initialized disk (preferably with a fast DOS)
COPYA from the *DOS 3.3 System Master*
Any sector editor program such as *DiskEdit* or *ZAP*

Laf Pak is a collection of four small but very enjoyable and playable games. Number one in the pak is **Creepy Corridors**, a maze type game that is by far the most fun, especially with the hilarious sound effects. **Apple Zap** is a four way shoot'em up game. **Space Race** is a game in which you race an opponent or the computer through a multitude of little bird-like aliens. Finally, **Mine Sweep** is an almost impossible game in which you try to clear a mine field in a given amount of time.

In trying to backup this disk, I first noted that the publisher was Sierra On-Line. Many companies use a standard copy-protection method on most of their products and I figured *Laf Pak* was protected, like most of their other products, with a nibble count on track 0. This turned out to be true. A check is made at the beginning of each of the individual games. It is always helpful to know of a company's past methods of copy-protection when trying to unlock any of their software. For example, the original *Zork* softkey in **The Book Of Softkeys Volume I**, page 156, still works for all of Infocom's adventures, almost all of Automated Simulations software can be *DEMUFFIN*-ed, and the boot-code-trace for *Hard Hat Mack* in **The Book Of Softkeys Volume I**, page 101, can be used on some other Electronic Arts games.

To disable the the disk checking routines for *Laf Pak*, I simply had to find the first JSR (a subroutine call) at the beginning of each of the games and replace it with NOP's (no operation). For example:

Creepy Corridors starts at $800 and at $808 there is a JSR $1B33 which is the call to the disk verification scheme. In machine code this code is 20 33 1B, so I simply used a disk searcher to find these three bytes on the *Laf Pak* disk. With *Laf Pak* this can easily be done since the disk is written in standard DOS 3.3 format. I found similar calls in the other three games and recorded the locations of all the bytes that needed to be changed to EA (op code for NOP). So, to back up *Laf Pak* simply use *COPYA* to copy the entire disk (always write-protect your original first!). Then use a sector editing program to make the following changes to the backup.

Track:	4	4	4	14	14	14	9	9	9	C	C	C
Sector:	0	0	0	0	0	0	2	2	2	7	7	7
Byte:	8	9	A	D	E	F	F	10	11	F	10	11
From:	20	33	1B	20	0D	72	20	7C	13	20	33	17
To:	EA	EA	EA	EA	EA	EA	EA	EA	EA	EA	EA	EA

You now have a completely unprotected backup of *Laf Pak* which can be copied with any standard copier. If this is all you want you can stop here, but I wasn't satisfied because using a full disk for a backup of a game is usually a waste of disk space. I wanted each game normalized, i.e. BRUNable and *F/D*able. I also wanted the *Laf Pak* machine to work as normal from a standard binary file.

Normalizing each game will illustrate the method of using memory moves to compact binary files and will also illustrate how to pass commands to DOS from within machine language programs. When we are done there will be plenty of room on the disk for any other games you want to store with *Laf Pak*.

The first thing to do, while the sector editor is still handy, is to change the name of the file that is run on boot up. It is named *LAF PAK*, but is preceded with seven CTRL H s. Since we will want to load this file in from hand, we need to get rid of the control characters. Use your sector editor to read in track $11, sector $F and, beginning with byte $0E, type in **LP** ($CC $D0) and enough spaces ($A0) to cover the seven CTRL H s ($88) and the ASCII codes for *LAF PAK*. Write this sector back to disk. Now when you CATALOG your backup *Laf Pak* disk, the first file should be a 4-sector binary file named *LP*.

Before we actually begin you need to boot your initialized disk, type in the following code at $800 and save it as *PAGE MOVER*.

```
CALL -151
800:A2  00  A0  FF  84  3E  C8  84
808:3C  84  42  BD  25  08  85  43
810:E8  BD  25  08  85  3D  E8  BD
818:25  08  85  3F  E8  20  2C  FE
820:E0  06  D0  DE  60  00  00  00
BSAVE PAGE MOVER,A$800,L$28
```

In Assembly language this looks like:

```
0800-  A2 00     LDX #$00
0802-  A0 FF     LDY #$FF
0804-  84 3E     STY $3E
0806-  C8        INY
0807-  84 3C     STY $3C
0809-  84 42     STY $42
080B-  BD 25 08  LDA $0825,X
080E-  85 43     STA $43
0810-  E8        INX
0811-  BD 25 08  LDA $0825,X
0814-  85 3D     STA $3D
0816-  E8        INX
0817-  BD 25 08  LDA $0825,X
081A-  85 3F     STA $3F
081C-  E8        INX
081D-  20 2C FE  JSR $FE2C
0820-  E0 06     CPX #$06
0822-  D0 DE     BNE $0802
0824-  60        RTS
```

This routine uses the Monitor's move subroutine at $FE2C (note the Y-register must be 0 on entry) to perform block page moves where a page is considered to be 256 bytes starting at an address of the form *XX00*, i.e. the low order byte is 00. If $800 is considered the first byte of the routine, then the 'table' begins at byte $825. A table entry is 3 bytes designating which pages to move and where. For example, if I wanted the Monitor to perform the following move: 1100<2200.33FFM, the table entry would be: 825:11 22 33 . You can have as many as 85 table entries but you must set byte $21 to 3 times the number of memory moves to be performed (in other words, set it to the total number of bytes in the table). As long as you load this code on a page boundary, then bytes $0D, $13 and $19 are the only bytes you have to change to relocate the code. These locations should hold the hi-byte of the page to which you relocate the code. If you are using an assembler, you can make all of these changes in the source code by simply changing the origin and the contents of the table.

Our goal is to have 5 binary files, the 4 games and the menu or *FUN MACHINE* file which we will name *LAF PAK*. We will change the menu file to simply BRUN the selected game instead of loading it in directly from disk as it does on the original disk.

On the original, when you are playing any of the games and you hit ESC, the game will BRUN a file called *BOOT.OBJ* which in turn loads in the menu. We will simply change each game to BRUN *LAF PAK* instead of *BOOT.OBJ*.

The first thing we need to do is to save each of the games. I will illustrate all of the steps in detail for the first game, *Creepy Corridors*.

Creepy Corridors

1 Boot the initialized disk on which you saved *PAGE MOVER* (preferably a disk with a fast DOS on it). Insert your *Laf Pak* backup, load the boot-up file *LP* and get into the Monitor.

```
BLOAD LP
CALL -151
```

2 *LP* starts at $1400 where it first reads in the I/O routine at $220. Whatever the accumulator is holding on entry to $220 determines which game (or the menu) is to be loaded. A **0** means *Creepy Corridors* and a **4** means the menu. At $1438, the accumulator is loaded with **4** (indicating that the menu is to be loaded) and a jump is made to $220. We can change this JMP instruction ($4C) to a $60 (RTS) so that the I/O routine will be loaded, but not executed. To do this type:

```
143A:60 N 1400G
```

Upon examination of the I/O routine at $220, you can see that each game and the menu are loaded in three seperate chunks. The destination addresses and the number of pages to be loaded in each chunk are given in the tables at $2FA and $2EB, respectively. With this information and a little bit of hexadecimal addition, the following addresses can be determined:

Creepy Corridors:	$800—1CFF	$4000—5FFF	$7000—94FF
Space Race:	$800—15FF	$6000—6EFF	$8000—93FF
Mine Sweep:	$800—1EFF	$4000—5FFF	$8700—94FF
Apple Zap:	$4000—5FFF	$6000—7EFF	$7D00—8CFF
Fun Machine:	$800—EFF	$4000—60FF	$9000—94FF

3 We are going to let the *Laf Pak* routine read in each game, then stop it before it jumps to the beginning of each game. This is done by changing the JMP at $28A to a RTS:

```
28A:60
```

4 Now we just need to load the accumulator with **0** for *Creepy Corridors* and jump to $220. This is done by changing the code at $1438 so that the accumulator will contain a **0** when the I/O routine executes:

```
1439:00 4C N 1438G
```

5 The code is now in, so let's change *BOOT.OBJ* to *LAF PAK* so the menu will be loaded if ESC is pressed:

```
14C5:CC C1 C6 A0 D0 C1 CB A0
```

6 We next can compact the 3 chunks of code into one contiguous block with the following memory moves:

```
1D00<4000.5FFFM
3D00<7000.94FFM
```

7 Since *PAGE MOVER* will live at the end of the block (at $6200), we need to call this subroutine before the main program is run. So we put a JSR $6200 at the beginning of the code:

```
7FD:20 00 62
```

8 Now insert the initialized disk with *PAGE MOVER* on it, load it in and change it so that it will run at this new location:

```
BLOAD PAGE MOVER,A$6200
620D:62 N 6213:62 N 6219:62
```

9 Note that the inverse of the moves made in step 6 are 7000<3D00.61FFM & 4000<1D00.3CFFM, which give us the table entries for the move code:

```
6225:70 3D 61 40 1D 3C
```

10 Save the game:

```
BSAVE CREEPY CORRIDORS,A$7FD,L$5A80
```

You now have a completely normalized *Creepy Corridors* game which can be BRUN to your heart's content. Before going on, you should first test it by clearing memory and then running it from disk. An easy way to clear memory without turning your Apple OFF is by typing:

```
800:00 N 80<800.9500M
```

If something goes wrong, BLOAD the file and check the code at $6200 for any typos.

*In case you got lost along the way, here are the basic steps we performed in order to capture a compacted version of **Creepy Corridors**.*

A Load the game into memory.

B Compact the game code into one contiguous area of memory.

C Append the *PAGE MOVER* routine onto the end of the compacted code and fix it so it will run properly there.

D Put the proper values into the *PAGE MOVER* table.

E BSAVE the game after inserting some code at $7FD that calls the *PAGE MOVER* routine before the game tries to execute.

When the game is BRUN, the *PAGE MOVER* routine de-compacts the code.

*For the other three games, the steps are very similar to those we performed for **Creepy Corridors**.*

Space Race

1 — 3 see *Creepy Corridors*

4 The accumulator should hold a **1** so that *Space Race* will be loaded:

```
1439:01 4C N 1438G
```

5 Change *BOOT.OBJ* to *LAF-PAK*:

```
962:8D 84 C2 D2 D5 CE A0 CC
96A:C1 C6 A0 D0 C1 CB
```

6 Compact the game's code with the following memory moves:

```
1600<6000.6EFFM
2500<8000.93FFM
```

7 Insert the code which calls the PAGE MOVER routine:

```
7FD:20 00 39
```

8 Insert the initialized disk, load the move routine and relocate it:

```
BLOAD PAGE MOVER,A$3900
390D:39 N 3913:39 N 3919:39
```

9 Put in the table for PAGE MOVER:

```
3925:80 25 38 60 16 24
```

10 Save the game:

```
BSAVE SPACE RACE,A$7FD,L$3180
```

Mine Sweep

1 — 3 see *Creepy Corridors.*

4 The accumulator should hold a **2** so that *Mine Sweep* is loaded in:

```
1439:02 4C N 1438G
```

5 Change *BOOT.OBJ* to *LAF PAK*:

```
96D:CC C1 C6 A0 D0 C1 CB A0
```

6 Compact the code with the following memory moves:

```
2900<800.1EFFM
6000<8700.94FFM
```

7 Insert some code which calls the PAGE MOVER routine and jumps to the start of the game:

```
28FA:20 00 6E 4C 00 08
```

8 Insert the initialized disk, load PAGE MOVER and relocate it:

```
BLOAD PAGE MOVER,A$6E00
6E0D:6E N 6E13:6E N 6E19:6E
```

9 Install the memory move table at $6E25:

```
6E25:87 60 6D 08 29 3F
```

10 Finally, save the game:

```
BSAVE MINE SWEEP,A$28FA,L$4580
```

Apple Zap

1 — 3 see *Creepy Corridors.*

4 The accumulator should hold a 3 so that *Apple Zap* is loaded:

```
1439:03 4C N 1438G
```

5 Change *BOOT.OBJ* to *LAF PAK*:

```
63BD:CC C1 C6 A0 D0 C1 CB A0
```

6 No moves are necessary since the code is already contiguous.

7 Insert the code which jumps to the beginning of the game:

```
3FFD:4C 00 60
```

PAGE MOVER is not used with this game,so...

8 Insert the initialized disk and save the program:

```
BSAVE APPLE ZAP,A$3FFD,L$4D03
```

Before we move on to the menu program, I would like to note that *Mine Sweep*, *Apple Zap* and the menu all store standard hi-res pictures at $4000—$5FFF. Those of you who are ambitious and wish to save about 50 more disk sectors can pack these pictures and insert the unpacking routine with them to make these files even more compact. If you just want to be able to BRUN each of the games and do not care about the 'Fun Machine' menu, then you have completed this task and you can stop here. But if you would like to use the menu program on occasion, then read on.

The Fun Machine

Capturing the menu program is similar to the above except that we must intercept the jump to the I/O routine to load in a game and replace it with our own BRUN routine. At $862, the number of the game that is chosen is utilized to get an address from an address table at $D62. This address is stored in $ØØ.$Ø1 which is used as a vector to the correct entry into the I/O routine. We will place our own address table at $D62 so that $ØØ.$Ø1 becomes a pointer to the correct game that should be run. We will then print out BRUN with the code:

```
86C-   LDY #$Ø6
86E-   LDA $Ø879,Y
871-   JSR $FDED
874-   DEY
875-   BNE $Ø86E
877-   JMP $ØE3Ø
87A-   N U R B CTRL D
```

At $E3Ø, we will place a routine that uses the address in $ØØ.$Ø1 to print the name of the game. The code to do this looks like:

```
E3Ø-   LDY #$ØØ
E32-   LDA ($ØØ),Y
E34-   JSR $FDED
E37-   INY
E38-   BNE $ØE32
```

The result is that DOS will BRUN the desired game from the Fun Machine menu. All of the steps are summarized as follows:

1 — **3** see *Creepy Corridors*.

4 The accumulator should hold a **4** so that the menu is loaded:

1439:Ø4 4C N 1438G

5A Type in the code below which will print out the DOS command BRUN (preceded by a RETURN and a CTRL D):

86C:AØ Ø6 B9 79 Ø8 2Ø ED FD 88 DØ
876:F7 4C 3Ø ØE CE D5 D2 C2 84 8D

5B Type in this code which will print out the name of the game to be run:

E3Ø:AØ ØØ B1 ØØ 2Ø ED FD C8 DØ F8

5C Next, install in memory a table which contains the names of the four games (each followed by RETURN):

```
E40:C3 D2 C5 C5 D0 D9 A0 C3
E48:CF D2 D2 C9 C4 CF D2 D3
E50:8D C1 D0 D0 CC C5 A0 DA
E58:C1 D0 8D D3 D0 C1 C3 C5
E60:A0 D2 C1 C3 C5 8D CD C9
E68:CE C5 A0 D3 D7 C5 C5 D0
E70:8D
```

5D Now, put the addresses of the names in the address table:

```
D62:40 0E 51 0E 5B 0E 66 0E
```

6 Compact the menu code with the following memory moves:

```
F00<4000.60FFM
3000<9000.94FFM
```

7 Insert the code which calls the PAGE MOVER routine:

```
7FD:20 00 0E
```

8 Insert the initialized disk, load the PAGE MOVER and relocate it:

```
BLOAD PAGE MOVER,A$E00
E0D:0E N E13:0E N E19:0E
```

9 Install the PAGE MOVER table:

```
E25:40 0F 2F 90 30 34
```

10 Of course, we also have to save the compacted code for the menu:

```
BSAVE LAF PAK,A$7FD,L$2D03
```

Now you have a completely normalized *Laf Pak* that you can put on any one of your game disks. I hope the techniques I have illustrated will come handy when you are trying to normalize other copy-protected games.

Learning With Leeper

Sierra On-Line Inc.

Softkey For Learning With Leeper
by Marco Hunter
(Hardcore COMPUTIST # 13, page 8)

Requirements:
48K Apple][or compatible
One disk drive and DOS 3.3
COPYA from the *DOS 3.3 System Master*
A sector editor
A blank disk

The three things you can count on in this life are death, taxes, and Sierra On-Line's nibble counts. With this in mind, I tackled *Learning with Leeper*, a recent educational release from Sierra. As usual, the disk is normal DOS 3.3, easily *COPYA*-able but it will not work correctly). I discovered that the nibble count checks track $0. This was also standard Sierra.

Tracks $00, $03, and $1F are popular tracks for Sierra On-Line nibble counts. I decided to go all-out and remove the nibble count entirely. But when I finally found the nibble count routine, I decided that the easiest thing to do was simply avoid jumping into it, rather than trying to modify it. Let's deprotect *Learning With Leeper*:

1 Boot the *3.3 System Master* as usual (**PR#6** RETURN).

2 Use *COPYA* to make a backup of *Learning with Leeper*.

3 Use your sector editor to modify these bytes on the new copy eliminating the nibble count routine:

Track	Sector	Byte	From	To
$03	$0F	$2C	$20	$EA
$03	$0F	$2D	$00	$EA
$03	$0F	$2E	$12	$EA

Lion's Share

Davka Corporation

Softkey For Lion's Share
by Jan Eugenides
(Hardcore COMPUTIST # 12, page 14)

Requirements:
Apple][Plus or equivalent
One blank disk
Super IOB

The Lion's Share is a pretty good adventure, with some nice graphics. I enjoyed it, but even more, I enjoyed figuring out the protection scheme. It was much more of a challenge than getting the sword out of the snake pit! First, for those of you who just want to backup the game, here's the softkey:

1 Boot your *DOS 3.3 System Master* and clear any program in memory.

```
PR#6
FP
```

2 Enter the Monitor and make some modifications to DOS

```
CALL -151
BA69:60
BA6A<BA69.BA91M
BCDF:60
BCE0<BCDF.BCF8M
```

3 Put in a blank disk and put this modified DOS on it:

```
INIT A
```

4 Turn the disk over and put this DOS on the back side, too. You will have to cut a notch in the side of the disk to allow you to use the back:

```
INIT A
```

5 Put side one of *The Lion's Share* in the drive, start the drive ***with the drive door open*** then press the BREAK key:

```
PR#6
CTRL C
```

6 Close the door, wait for the break message and then clear the program in memory:

```
FP
```

7 Enter the Monitor and move the RWTS to a safe place for booting:

```
CALL -151
2000<B800.BFFFM
```

8 Put your *Super IOB* disk (or some other disk with a very short Hello program) in the drive, boot it and save the RWTS:.

```
6CTRL P
BSAVE RWTS.LIONS,A$2000,L$800
```

9 Load *Super IOB*, type in the controller at the end of this article and then RUN *Super IOB*.

10 Copy both sides of the original *Lion's Share* disk to both sides of the disks you formatted at the start of the procedure.

11 You're done! Boot your copy and enjoy!

Beneath The Softkey

When I booted the game, I noticed a prompt at the bottom of the screen. This usually means that some form of DOS is being used. Going on this assumption, I decided to have a look at tracks $0—2 of the disk, to see if I could ferret out the system.

No dice. I couldn't read the disk at all, nor would it CATALOG. Using a Hardcore COMPUTIST program called *Diskview 1.1A*, I managed to peek at the raw nibble dump, and it was definitely nonstandard. So, I resorted to a little trickery.

First I removed the top 8 RAM chips from my machine. These are the ones nearest to the left rear of the Apple, inside the white box labeled "RAM." (Scary, isn't it?) Then I booted my *System Master*. (It has to be the master, and not a slave disk.) This had the effect of loading DOS 16K lower in memory than usual. (The higher memory was in my left hand!)

You see, the master disk checks to see how much memory is available and loads DOS in at the top of this available memory. A slave disk, on the other hand, always loads DOS into the same location, namely the location it was in when the slave was made.

I then initialized a slave diskette, using my now 32K Apple. I'll call this the 32K slave. Next, I replaced the RAM chips, and booted *The Lion's Share.*

Pressing RESET causes the disk to re-boot, since the reset vector has been changed, so I took advantage of this fact. I now removed the game disk, and put in my 32K slave. Pressing RESET caused the disk to boot, and regular old DOS 3.3 was loaded in at its 32K location. Because of this, Davka's DOS was still intact in the upper memory! Now it was a simple matter to save this Davka DOS with an address of $9D00 and a length of $22FF.

I now had Davka DOS on my disk as a binary file. You may be able to use this method on other protected disks. Just keep the 32K slave around for future use.

The next step was to boot up the *System Master* again, thus placing DOS 3.3 back in its normal location at $9D00. I now BLOADed Davka DOS at $2D00. (It can be anywhere, really, as long as it doesn't overwrite DOS 3.3)

Using the Monitor VERIFY command I made a print-out of all the locations which were different between the two DOSs. This is a nifty command which compares two ranges in memory and tells you if they are the same or not, and if not, what the differences are. Check the Apple reference manual for more information.

In this case, the command was 2D00<9D00.BFFFV. (Turn on your printer first with a PR#1, and set it to skip the perforations if you can.) This gave me a complete list of all the changes!

With my trusty copy of ***Beneath Apple DOS*** in hand, I proceeded to analyze the changes. I saw that the spelling of the SAVE and INIT commands had been altered but the rest of the commands were the same.

Most importantly, I discovered that both the read and write routines had been changed, as well as their translate tables. No wonder I couldn't read the disk; it was scrambled!

Then it was no problem. I just patched the DOS 3.3 read routine to match that used by Davka. Sure enough, I could now CATALOG the game disk. Of course, the DOS 3.3 write routine was still standard, so all I had to do was *FID* all the files over to a normal disk, right? Not quite, but close!

Once I had copied all the files to a normal disk, I booted it and tried to run the game. It worked perfectly until I tried the SAVE GAME option, and then it bombed.

The Apple obligingly informed me that it had stopped at $BA78. Hmmm. Isn't that unused space in DOS? Yep, it sure is. But Davka put a routine there which changed the address and data markers back to standard, so that their game could read and write to standard disks. (Pretty sneaky, huh?)

I replaced this whole section with 60's (RTS), and also the section at $BCDF, which was part of the same routine.

After I had initially deprotected *The Lion's Share*, I discovered something which made the overall softkey procedure a little easier. What I found was that *The Lion's Share* boot program would BREAK (stop execution) if CTRL-C was typed as the disk booted. After the BREAK occurred, typing:

```
FP
CALL -151
2ØØØ<B8ØØ.BFFFM
```

allowed me to enter the Monitor and move the *Lion's Share* RWTS to a safe location. After booting with a slave disk, I just BSAVEd the RWTS.

To automate the copy process a little, I modified the SWAP Controller of *Super IOB* so that it would utilize the *Lion's Share* RWTS. All these changes were incorporated into the softkey procedure which I detailed at the beginning of this article.

Ta Daaa!

I haved played the whole game through on my copy, and it works perfectly. It works exactly like the original, only now I don't have to worry about damaging it because I can make backups. Also, I can peek at the files if I feel too frustrated, although this game isn't that hard. Not like *Zork III*, anyway! I can also check out the program to see how it works and learn some more. That's what it's all about, right?

controller

```
1ØØØ REM SWAP CONTROLLER (LIONS SHARE )
1Ø1Ø TK = 3 :ST = Ø :LT = 35 :CD = WR
1Ø2Ø T1 = TK : GOSUB 49Ø : GOSUB 36Ø : ONERR GOTO 55Ø
1Ø3Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø3Ø
1Ø4Ø IF BF THEN 1Ø6Ø
1Ø5Ø ST = Ø :TK = TK + 1 : IF TK < LT THEN 1Ø3Ø
1Ø6Ø GOSUB 49Ø :TK = T1 :ST = Ø : GOSUB 36Ø
1Ø7Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø7Ø
1Ø8Ø ST = Ø :TK = TK + 1 : IF BF = Ø AND TK < LT THEN 1Ø7Ø
1Ø9Ø IF TK < LT THEN 1Ø2Ø
11ØØ HOME : PRINT "REMEMBER^TO^COPY^BOTH^SIDES" : END
1ØØ1Ø PRINT CHR$ (4 ) "BLOAD^RWTS.LIONS,A$19ØØ"
```

Master Type v1.7

Lightning Software

Master Type Softkey
by Peter Rongays
(Hardcore COMPUTIST # 15, page 8)

Requirements:
48K Apple][Plus or equivalent, with
Old F8-ROM or copycard
Super IOB
One blank disk

Bruce Zweig's *Master Type* has been one of the best-selling pieces of software for the Apple][ever since it was released back in 1981. The program's most unfortunate drawback is a common one: it is copy-protected. Since the program is undoubtedly being used in a fair number of schools throughout the USA and elsewhere, there is a corresponding demand for information on how to backup this disk. The reluctance of an instructor to turn over the only copy of a $40 program to a group of precocious eight-year-olds is entirely understandable.

Luckily, as I found out, *Super IOB* has little difficulty in copying *Master Type*. This article will explain how to make the backup.

Judging from the *EDD III* parameter list for *Master Type*, it appears to have been protected by *Lock-It-Up*, one of those copy-protection utilities that you may have seen advertised as being able to produce an 'uncopyable' disk (Note: The 'uncopyable' disk is close kin to the 'unsinkable' Titanic). Indeed, *Master Type* is protected fairly well, as evidenced by its several months stay on the Hardcore **Most Wanted List** and the difficulty of backing it up with a bit copier.

Upon booting *Master Type*, the familiar Applesoft prompt will appear while the game loads, an indication that a disk can be copied by *Super IOB* with the Swap Controller installed, as long as its RWTS can be captured by some means. This is the case with *Master Type*, but the *Super IOB* copy made with the standard Swap Controller will **not** work because the text of some of the DOS commands have been altered. You can verify this by booting *Master Type*, halting it with an old Monitor RESET or NMI and then examiming memory from $A884 to $A9Ø8. About half of the normal DOS commands have been blanked out entirely and, of those that remain, nine are the same as with normal DOS and five have had their text changed:

DOS 3.3 Command	Master Type Command
INIT	SAVE
DELETE	KILLDE
CLOSE	CLOSE
READ	READ
EXEC	EXEC
WRITE	WRITE
OPEN	OPEN
CATALOG	CATNDOG
NOMON	NOMON
PR#	PR#
IN#	IN#
FP	FP
BLOAD	YZ123
BRUN	YZ23

Because the majority of *Master Type* is written in Assembly language, it would take a bit of work to track down the location from which these altered DOS commands are being executed. This would be required if you wanted to make the program totally compatible with DOS 3.3. Instead of all this work, I decided to modify the *Super IOB* Swap Controller a bit so that the entire *Master Type* disk, with the exception of the sectors that contain the RWTS, would be copied. A *Master Type* disk with a hybrid DOS is thus created. It will be normally formatted and completely COPYAable, but its DOS will still use the altered commands. If you examine the *Master Type* controller listed at the end of this article you will see that it is the Swap Controller with lines 1Ø1Ø, 1Ø6Ø and 1ØØ1Ø modified and line 1Ø65 added.

One other change that must be made to the copy of the *Master Type* disk is to fill in two of the free areas in the RWTS with hexadecimal $6Ø's. This is necessary because *Master Type* can store user-created lessons on normally formatted disks. On the original *Master Type* disk, this free space contains some code which modifies the RWTS depending upon whether the original or a data disk has to be read. This code won't be needed, or present, on the copy. Therefore, we will just replace it with a bunch of machine language RTS's.

With a little more effort, I'm sure *Master Type* could be made to work with a totally normal DOS, but it probably is not worth the effort to do so unless you have some modifications or enhancements you would like to add to it. I will leave that chore up to the more ambitious readers.

Making the Copy

1 Boot up the original *Master Type* disk and, after the program has been loaded, stop it with a RESET or NMI.

2 From the Monitor, move the *Master Type* RWTS to a 'safe' location:

```
1900<B800.BFFFM
```

3 Boot up a DOS 3.3 slave disk and then BSAVE the *Master Type* RWTS onto a disk which contains *Super IOB*:

```
BSAVE RWTS.MASTER TYPE, A$1900,L$800
```

4 Enter the Monitor and fill in two of the 'holes' in the RWTS with $6Ø's (RTS's) before initializing a blank disk:

```
CALL -151
BA69:60 N BA6A<BA69.BA94M
BCDF:60 N BCE0<BCDF.BCFEM
INIT HELLO
```

5 Load *Super IOB*, install the controller and then make a copy to the disk initialized in Step 4 above. Note: **Do not** reformat the disk.

controller

```
1ØØØ REM MASTER TYPE CONTROLLER
1Ø1Ø TK = Ø :ST = 1Ø :LT = 35 :CD = WR
1Ø2Ø T1 = TK : GOSUB 49Ø : GOSUB 36Ø : ONERR GOTO 55Ø
1Ø3Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø3Ø
1Ø4Ø IF BF THEN 1Ø6Ø
1Ø5Ø ST = Ø :TK = TK + 1 : IF TK < LT THEN 1Ø3Ø
1Ø6Ø GOSUB 49Ø :TK = T1 :ST = Ø : IF TK = Ø THEN ST = 1Ø
1Ø65 GOSUB 36Ø
1Ø7Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø7Ø
1Ø8Ø ST = Ø :TK = TK + 1 : IF BF = Ø AND TK < LT THEN 1Ø7Ø
1Ø9Ø IF TK < LT THEN 1Ø2Ø
11ØØ HOME : PRINT "EVERYTHING^O.K.^NO^DOS^ON^COPY" : END
1ØØ1Ø IF PEEK ( 64ØØ ) < > 162 THEN PRINT CHR$ (4 ) "BLOAD
    RWTS.MASTER^TYPE,A$19ØØ"
```

An Alternate Softkey *by Harry Noel*

I don't have an F8 Monitor ROM, so I had to find another way to load in *Master Type*'s RWTS. This is how you can do it:

1 Boot a normal DOS 3.3 disk and insert the *Master Type* disk.

2 Enter the Monitor with:

```
CALL -151
```

3 Move BOOT0 into RAM and modify it to jump to $8801:

```
8600<C600.C6FFM 86FA:88
```

4 Place a jump into the Monitor and start the boot:

```
8801:4C 59 FF
8600G
```

5 Stop the drive and move $800 to $8800:

```
C0E8
8800<800.8FFM
```

6 Set an indirect jump to the modified read routine:

```
880E:80
```

7 Return control after reading the RWTS and start the read:

```
884A:4C 59 FF
8600G
```

8 Stop the read and move the RWTS to a safe location:

```
C0E8
1900<B800.BFFFM
```

9 Boot the *Super IOB* disk:

```
6 CTRL P
```

10 And type:

```
BSAVE RWTS.MASTER TYPE,A$1900,L$800
```

Now continue at step 4 in Mr. Rongays' article and you will have a COPYAable version of *Master Type*.

If you want to make this disk totally DOS 3.3, you must go one step further.

First, copy all the files off the broken *Master Type* to a normal DOS 3.3 disk (I used *Copy II Plus v4.4C* to transfer files) and get ready to use your sector editor (I used *Tricky Dick*). Find all the occurences of these low-byte ASCII codes: **CATNDOG, KILLDE, SAVE,YZ123,YZ23**. Change them to **CATALOG, DELETE, INIT, BLOAD, BRUN** respectively. Then find these high-byte ASCII codes: **YZ123** and **YZ23** and change these to **BLOAD** and **BRUN**, respectively. You now have a normal copy of *Master Type*.

MatheMagic

International Software Marketing

Softkey For MatheMagic
by Doni G. Grande
(Hardcore COMPUTIST # 14, page 7)

Requirements:
Apple][with 48K
COPYA from the *DOS 3.3 System Master*
Super IOB
A blank disk

MatheMagic is a program which transforms your microcomputer into the ultimate calculator. With it, you can program your computer in the same way you would a programmable calculator.

The program disk itself is copy-protected, but it can use and initialize normal data disks. This is done by altering RWTS when access is needed to the original disk, and then changing it back to normal when a data disk is accessed. The only parts of RWTS I found changed were the following locations:

Location:	$B991	$BCAE	$BC60
From:	$DE	$DE	$FF
To:	$DF	$DF	$FE

The first two locations are the address epilog start bytes and the last location is the sync mark used in the protected DOS. The only problem with just changing these locations to their normal values is that the program keeps changing them back whenever it accesses the program disk, generating an I/O ERROR. Also, since the program is a compiled basic program, it is very difficult to discover the location in the program at which this change is made.

The answer to the problem is to let the program change DOS to access the program disk, but patch the RWTS subroutine to change those locations back to normal whenever it is called.

The main entry to RWTS is at $BD00, and a JuMP to the patch can be made there if the original instructions at this entry point are duplicated in the patch. After DOS is put back to normal, the patch jumps back to RWTS and all operates normally!

The steps required to make a normal copy of *MatheMagic* are:

1 Boot a normal DOS disk.

2 Enter the Monitor, move an image of the normal DOS 3.3 RWTS to $19ØØ and then modify it so it can be used by *Super IOB* to read the protected *MatheMagic* disk:

```
CALL -151
19ØØ<B8ØØ.BFFFM
1A91: DF
1DAE: DF
BSAVE RWTS, A$19ØØ, L$8ØØ
```

3 Run *Super IOB* with the Swap Controller installed to make a copy of *MatheMagic*. The Swap Controller will use the RWTS created in Step 2 to read the disk.

4 Type in the following short patch:

```
Ø36Ø:84 48 85 49 A9 DE 8D 91
Ø368:B9 8D AE BC A9 FF 8D 6Ø
Ø37Ø:BC 4C Ø4 BD
```

This is the patch which will be executed just before RWTS is entered. The Monitor listing is:

```
Ø36Ø- 84 48     STY $48     Store Y and A as RWTS would
Ø362- 85 49     STA $49
Ø364- A9 DE     LDA #$DE    Fix data and address marks
Ø366- 8D 91 B9  STA $B991
Ø369- 8D AE BC  STA $BCAE
Ø36C- A9 FF     LDA #$FF    Might as well fix the sync byte too
Ø36E- 8D 6Ø BC  STA $BC6Ø
Ø371- 4C Ø4 BD  JMP $BDØ4   Return control to the RWTS
```

5 Save this patch on the copy of the *MatheMagic* disk:

```
BSAVE DOSPATCH, A$36Ø, L$14
```

6 Load the file *MATH.HELLO* from the copy disk and add the following:

```
227 PRINT CHR$( 4) "BLOAD▵DOSPATCH"
228 POKE 48384, 76 :POKE 48385, 96 :POKE
     48386, 3 :POKE 48387, 234
```

This places a JMP to $36Ø and a NOP as the first instructions in RWTS so that *DOSPATCH* is executed before every call to RWTS.

7 Save *MATH.HELLO* back to the copy disk:

```
UNLOCK MATH.HELLO
SAVE MATH.HELLO
```

Millionaire

Blue Chip Software

Softkey For Millionaire
by Ryan Hodge
(Hardcore COMPUTIST # 12, page 7)

Requirements:
48K Apple][Plus equivalent
FID from the *DOS 3.3 System Master*
A blank disk
Integer Card or modified F8-ROM or NMI card

1 INIT a blank disk with *INITIAL* as the null hello program.

```
INIT INITIAL
```

2 Load *FID* from the *DOS 3.3 System Master* and modify DOS:

```
BLOAD FID
CALL -151
B925:18 60
B988:18 60
BE48:18
B8FB:29 00
```

3 Run *FID* and copy these files onto a normal DOS 3.3 disk: *CHAIN*, *COMMON*, *INDUST*, *STOCKS*, *MESDATA*, *PLAY*, *DESCRIP*, *PLAYER*, *RANDOM.DTA* (and *SAVE* if it's on your disk):

```
803G
```

4 Boot (PR#6 RETURN) the original *Millionaire* disk and when the screen 'asks' if you wish to resume an old game, hit RESET (or use your NMI card) to enter the Monitor, repeat the DOS changes in step 2 and finally, put the *Millionaire* hello program, *INITIAL*, onto your normal disk containing the other *Millionaire* files:

```
SAVE INITIAL
```

Music Construction Set

Electronic Arts

Backup And Modify Music Construction Set
by Dan Rosenberg
(Hardcore COMPUTIST # 12, page 27)

Requirements:
a separate list is given for each version

Many readers probably own a copy of *Music Construction Set (MCS)* which, in my opinion, is the best music program currently available for the Apple. However, I suspect that many owners of *MCS* are dismayed, like I was, by their inability to back up or modify the program. Because *MCS* is protected, it has an annoying habit of checking for an original program disk from time to time. For the user, this copy-protection measure means increased wear on the original *MCS* disk in addition to slowing down the overall process of composing music.

The *MCS* disk uses a version of DOS 3.3 called DOS 3.3P (the P stands for Protected.) Most copy programs will copy the *MCS* disk without errors, but the copy will not work because the program(s) check the disk for its originality.

Luckily it is not too difficult to defeat DOS 3.3P. Once the protection has been defeated it is possible to make some modifications to *MCS*.

The Copy

The first thing to do is determine which version of *MCS* you have. The different versions require entirely different copy methods. I know of two different versions, which I will call version 1 and version 2. Version 1 does not support the cassette output function (it won't ask you if you would like cassette output), and the files *A3*, *A4*, *P3*, *N* and *O* appear in the catalog. Version 2 allows you to use cassette port output, and *A3*, *A4* and *P3* do not appear in the CATALOG. Check for which version you have, and then use the appropriate method.

MCS version 1 copy

Requirements:
48K Apple][Plus or equivalent
COPYA from the *DOS 3.3 System Master* or bit copy program
A blank disk

1 Boot up with any DOS 3.3 disk:

```
PR#6
```

2 Copy your *MCS* disk with *COPYA* onto the blank disk. If *COPYA* has any trouble making a copy, get out your bit copy program and copy tracks $0—$22 with the standard parameters:

```
RUN COPYA
```

3 Insert the copy of *MCS* and load the binary file called *A4*:

```
BLOAD A4
```

4 Enter the Monitor by typing:

```
CALL -151
```

5 Make the following modifications to the image of *A4* in memory:

```
9131:60
913A:EA EA
4C00:60
```

6 Resave the file *A4* by typing:

```
BSAVE A4,A$4A00,L$B60
```

7 Boot up your copy of the *MCS* disk and start constructing (musically, that is).

MCS version 2 copy

Requirements:
Apple][Plus or equivalent
A blank disk
Text-Editing program or...
MAKE TEXT from the *DOS 3.3 System Master disk*

1 Boot up a DOS 3.3 disk and then initialize a blank disk with a 'null' *HELLO* program:

```
PR#6
FP
INIT HELLO
```

2 Boot up with the original *MCS* disk:

```
PR#6
```

3 When the title page saying "Will Harvey's Music Construction Set" comes up, hit RESET (CTRL RESET). Be sure to RESET as soon as you see the title page!

4 The Monitor prompt should now be showing, so place the disk you initialized in step 1 into the drive and type in the following commands:

```
BSAVE H,A$400,L$600
BSAVE A3,A$A00,L$4000
BSAVE A4,A$4A00,L$4B60
BSAVE N,A$7400,L$120
BSAVE P3,A$300,L$D0
```

5 Now, boot up with the disk which contains the *MCS* files. Since you can't do a **PR#6** RETURN from *MCS*, you will have to turn the power OFF (or CTRL RESET on the Apple //e).

6 Load in the file called *A4* and then enter the Monitor:

```
BLOAD A4
CALL -151
```

7 Next, type:

```
86D9.86DB
```

If you get **4C 00 C6** in response, then type:

```
86D9:EA EA EA
7F39:60
```

Note: If your new copy doesn't work, type in:

```
910D:60
```

If you didn't get **4C 00 C6**, then type:

```
9131:60
913A:EA EA
```

8 Save your changes by typing:

```
BSAVE A4,A$4000,L$4B60
```

The new copy will now work, but you'll want a program to start it off. Since part of *Music Construction Set* uses the normal BASIC memory and there is little room for a machine language program, we will use an EXEC file. You will either need a word processing program or you can use *MAKE TEXT* from the DOS 3.3 System Master to create the EXEC file. Note: if you use *MAKE TEXT*, don't make any typing mistakes because the backspace characters will be saved into your file.

9 Get out your word-processor or *MAKE TEXT* and create a text file which contains the following commands:

```
HGR
POKE -16301,0
BLOAD A3
BLOAD A4
BLOAD P3
BLOAD N
BLOAD H
POKE -16368 ,13
CALL 2156
```

10 Save this text file under the name *MCS.HELLO* on your copy of the *Music Construction Set*:

```
SAVE MCS.HELLO
```

11 Type in the following program and save it on the disk as the Hello program:

```
FP
10 PRINT : PRINT CHR$ (4 ) "EXEC^MCS.HELLO"
SAVE HELLO
```

12 To copy any of the music files from the original *MCS* disk to the copy, a file-transfer program like *FID* can be used. Both the music file and its *.OBJ* file have to be transferred. For instance, if you want the song ''Dixie'' on your copy of *MCS*, then the files *DIXIE* and *DIXIE.OBJ* will both have to be transferred.

13 You can now boot up your new *Music Construction Set* disk and use it normally. If you are using a DOS other than DOS 3.3 (*Pronto-DOS, Diversi-DOS*, etc.), you'll find the program will run much faster. I recommend you get one of these fast DOS's as they are well worth the money.

Alternate Method For MCS Version 2

Requirements:

48K Apple][Plus or equivalent
COPYA or a bit-copy program
Disk-editing program
A blank disk

1 Make a copy of the original *Music Construction Set* disk with *COPYA* (If *COPYA* won't copy it, use a bit-copier):

RUN COPYA

2 Use your disk-editor to make the following changes to the copy of the *MCS*:

Track	Sector	Byte	From	To
$B	$2	$D9	$4C	$EA
$B	$2	$DA	$ØØ	$EA
$B	$2	$DB	$C6	$EA
$B	$D	$29	$2Ø	$EA
$B	$D	$2A	$ØØ	$EA
$B	$D	$2B	$4C	$18

Don't forget to write the modified sectors back to the copy of *MCS*.

MCS Modifications

It is also possible to customize the *Music Construction Set*. For example, say that you have your *Mockingboard* in a slot other than 4. The slot that *MCS* expects to find the *Mockingboard* in can be easily changed with the program below. Note that there are two different DATA statements and you should use one or the other depending upon which version of *MCS* you own. Just type the program in (using the appropriate DATA statement) and save it as *MB SLOT CHANGER* by typing:

SAVE MB SLOT CHANGER

Just RUN the program when you want to have *MCS* utilize a *Mockingboard* in another slot. The program can also be used if *MCS* will not boot because you have a card in slot 4 that is not a *Mockingboard*. Just run the *MB SLOT CHANGER* program and then change the *Mockingboard* slot to one of the slots in your computer that is empty.

```
5 REM DATA FOR VERSION 1
1Ø DATA ^35946 ,35951 ,35956 ,35962 ,35967 ,35972 ,35982 ,35987
    ,35992 ,35998 ,36ØØ3 ,36ØØ8 ,36Ø19 ,36Ø22 ,36Ø27 ,36Ø3Ø
5 REM DATA FOR VERSION 2
1Ø DATA ^35932 ,35937 ,35942 ,35948 ,35953 ,35958 ,35968 ,35973
    ,35978 ,35984 ,35989 ,35994 ,36ØØ8 ,36ØØ5 ,36Ø13 ,36Ø16
15 REM TYPE IN ONLY 1 SET OF DATA!!
2Ø DIM A(16 ) : TEXT : HOME
3Ø FOR I = 1 TO 16 : READ A(I ) : NEXT
4Ø INVERSE : PRINT "MCS^MOCKINGBOARD^SLOT^CHANGER"
5Ø NORMAL : VTAB 5 : PRINT "WHAT^SLOT^WOULD^YOU^LIKE^TO^PUT^
    YOUR^MOCKINGBOARD^IN,^OR^HAVE^MUSIC^CONSTRUCTIO^SET^
    LOOK^FOR^A^MOCKINGBOARD?^(1-7):" ; : GET Z$
```

```
6Ø Z = VAL (Z$ ) : IF Z < 1 OR Z > 7 THEN HTAB 1 : GOTO 5Ø
7Ø PRINT "MAKE^SURE^YOUR^MUSIC^CONSTRUCTION^SET^^DISK^IS^IN^
    THE^DRIVE...AND^KEY" ; : GET Z$
8Ø PRINT : PRINT CHR$ (4 ) "BLOAD^A4"
9Ø FOR I = 1 TO 16 : POKE A(I ) ,Z + 192 : NEXT
1ØØ PRINT : PRINT CHR$ (4 ) "BSAVE^A4,A$4AØØ,L$4B6Ø"
11Ø PRINT : PRINT "DONE." : END
```

If you performed the alternate copy method (sector edit method) on version 2, then the above program will not work for you because your disk does not have a file called *A4* on it. However, you can still change the *Mockingboard* slot if you have a sector-editor. The bytes to modify are stored on track $B, sector $8.

Get your disk-editor running and read in track $B, sector $8 from the copy of *MCS* you made. The table below shows the bytes to modify. You will need to substitute the slot number you want your *Mockingboard* in for the ***n*** in the ***To*** column. For instance, if you want to use slot 2 for the *Mockingboard*, change the bytes listed in the table from $C4's to $C2's.

Byte	From	To	Byte	From	To
$5C	$C4	$Cn	$8A	$C4	$Cn
$61	$C4	$Cn	$9Ø	$C4	$Cn
$66	$C4	$Cn	$95	$C4	$Cn
$6C	$C4	$Cn	$9A	$C4	$Cn
$71	$C4	$Cn	$A5	$C4	$Cn
$76	$C4	$Cn	$A8	$C4	$Cn
$8Ø	$C4	$Cn	$AD	$C4	$Cn
$85	$C4	$Cn	$BØ	$C4	$Cn

After changing the bytes, don't forget to write the sector back to your disk.

Cassette Port Output

Version 1 of *Music Construction Set* does not allow you to use the cassette port for output as does version 2. Using the cassette port for output allows you to play the music through an external amplifier and speaker for improved sound quality (especially if you turn the treble all the way down on your amplifier). Adding cassette port output to version 1 of *MCS* is really quite simple if you have a little knowledge of the Apple's built-in I/O.

Sound can be output to the Apple's speaker by referencing address $CØ3Ø or to the cassette output by referencing address $CØ2Ø. The binary file called *A4* controls the output of *MCS*, and the changes necessary for cassette output need only be applied to this one file. The program listed below will allow you to pick the output path (speaker or cassette) and will make the necessary modifications to *A4*. Type in this program and SAVE it as *CASSETTE OUTPUT*.

CASSETTE OUTPUT

```
1Ø DATA^23,43,46,66
2Ø TEXT : HOME
3Ø FOR I = 1 TO 4 : READ A(I) : NEXT
4Ø INVERSE : PRINT "MCS^CASSETTE/APPLE^SPEAKER^OUTPUT^TOGGLE"
5Ø NORMAL : VTAB 5 : PRINT "DO^YOU^WANT^<C>ASSETTE^OR^<A>PPLE^
   SPEAKER^OUTPUT^(C^OR^A):";: GET Z$
6Ø IF Z$ <> "C" AND Z$ <> "A" THEN HTAB 1 : GOTO 5Ø
7Ø PRINT : PRINT CHR$ (4) "BLOAD^A4"
8Ø N = 33 : IF Z$ = "A" THEN N = 49
9Ø FOR I = 1 TO 4 : POKE A(I) + 218ØØ ,N : NEXT
1ØØ PRINT : PRINT CHR$ (4) "BSAVE^A4,A$4AØØ,L$4B6Ø"
11Ø PRINT "DONE." : END
```

When you want to switch from speaker to cassette output (or vice-versa), just:

RUN CASSETTE OUTPUT

Final Words

Since your new *Music Construction Set* is now on a normal DOS 3.3 disk, if a DOS error is generated (like trying to load a non-existent file), the program will leave you in BASIC (version 2 only). To return to *MCS* just type:

CALL 2156

Anything that you were working on at the time of the error will still be intact.

That about does it for my modifications to *Music Construction Set*. I am sure that if you poke around a bit, you can come up with some more enhancements for *MCS*. A good place to start investigating is the area around $7FØØ. Happy constructing!

PFS software*

Software Publishing Corp.

Deprotecting PFS Software
by Gary J. Wolfe
(Hardcore COMPUTIST # 14, page 6)

* *This softkey applies to: **PFS:File, PFS:File //e, PFS:Report, PFS:Report //e, PFS:Graph, PFS:Graph //e.***

Requirements:

Apple][Plus or equivalent
One disk drive
COPYA from your *DOS 3.3 System Master*
Sector-editor with search capability (*Zap, Inspector*, etc.)

Although most of PFS's software can be backed-up by using *Copy II Plus*' normal-copy (not bit-copy) utility, I prefer to **completely** remove the copy-protection from commercial software that I own. Even though the PFS series of programs are written in Pascal, the disk-protection code is written in Assembly language and can be fairly easily circumvented. To remove the protection from any of the PFS programs listed above, you will need some sort of disk-search utility. The entire disk has to be searched for a byte sequence of **DØ Ø4 88 98 FØ 27**. This code is found in a routine which checks the disk for the presence of extra bits in the sync fields, a protection scheme called the 'bit insertion technique'. If the extra bits are present, an $FF will be pushed on the stack, otherwise, a $ØØ will be pushed onto the stack before the routine returns to its caller. By changing the second byte of the search sequence from an $Ø4 to a $29, the protection code can modified so that it will always push an $FF onto the stack whether the extra bits are present or not.

1 First, run *COPYA* to make a copy of the PFS program.

2 Get out your disk-search utility and search the copy for the all occurrences of DØ Ø4 88 98 FØ 27 and change the $Ø4 in this sequence to a $29. Be sure to write the changes back to the disk.

3 Write-protect the disk before trying to boot it.

Penguin Software (PS)

Softkey For Transylvania & The Quest

by Thomas A. Phelps

(Hardcore COMPUTIST # 13, page 14)

NOTE: *As you are aware, both Penguin Software and Beagle Bros are spoken of highly for their unprotected applications disks. Even the Beagle game disk* ***Beagle Bag*** *bears no protection. Penguin game disks, on the other hand, are copy-protected, thus not allowing easy backup nor the art of APT (Advanced Playing Techniques).*

Arcade games in broken form do not especially lend themselves well to APT due to difficult-to-understand machine code, but adventures, especially ones written in BASIC, do. After performing the softkey for both ***Transylvania*** *and* ***The Quest****, you can examine the BASIC programs which are the core of these fine adventures, as well as each well-drawn picture, without the need of traveling there in the adventure itself.*

The protection for these adventures is more or less the same, and is easily broken with just a little effort. First, type in the Super IOB controller below. Since this controller works on all Penguin entertainment software I've tried (including Coveted Mirror), save it to disk as PENGUIN.CON.

After using the controller on a Penguin disk, you should add DOS (ideally a fast DOS).

controller

```
1ØØØ REM PENGUIN CONTROLLER
1Ø1Ø TK = 2 :ST = Ø :LT = 35 :CD = WR
1Ø2Ø T1 = TK : GOSUB 49Ø : GOSUB 111Ø
1Ø3Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1
: IF ST < DOS THEN 1Ø3Ø
1Ø4Ø IF BF THEN 1Ø6Ø
1Ø5Ø ST = Ø :TK = TK + 1 : GOSUB 111Ø : IF TK < LT THEN 1Ø3Ø
1Ø6Ø POKE 475Ø5 ,222 : POKE 47413 ,222 : GOSUB 23Ø : GOSUB 49Ø :TK =
     T1 :ST = Ø
1Ø7Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø7Ø
1Ø8Ø ST = Ø :TK = TK + 1 : IF BF = Ø AND TK < LT THEN 1Ø7Ø
1Ø9Ø IF TK < LT THEN 1Ø2Ø
11ØØ HOME : PRINT : PRINT "DONE^WITH^COPY" : END
111Ø POKE 475Ø5 ,218 : POKE 47413 ,218 : IF TK / 2 = INT (TK / 2 )
     THEN 23Ø
112Ø RESTORE : GOTO 19Ø
63Ø1Ø DATA 212 ,17Ø ,15Ø
```

Translyvania

Requirements:
Apple][, Apple][Plus, Apple //e or compatible
one blank disk
Super IOB

First, run *Super IOB* with the Penguin controller installed. Once the program has been copied, only one modification needs to be made for the program to function. The Binary file *TPAR* checks to see that $D6 is non-zero (causing the BASIC program to auto-run). If it is not non-zero, the program will exit to BASIC as soon as the player is ready to enter his first command. This problem can be fixed in either one of two ways. One way is to add **5 POKE 214,255** to *HELLO*, or, better yet, the file *TPAR* can be modified to ignore this check. To modify *TPAR*, follow these steps:

1 Since *TPAR* loads in an area normally used by DOS buffers, change MAXFILES:

```
MAXFILES1
```

2 Load in the file to be modified:

```
BLOAD TPAR
```

3 Enter the Monitor:

```
CALL -151
```

4 NOP the $D6 byte-check:

```
943D:EA EA EA
```

5 Save the new file:

```
BSAVE TPAR,A$9400,L$69D
```

For a complete, stand-alone copy, simply copy DOS onto the disk and use *HELLO* as the boot-up program.

According to the VTOC, the disk has no free sectors, but a VTOC rebuilder will show several free sectors which may be used to store saved games and the APT programs listed in this article (no extra disk needed!).

In addition to this bit of protection, *TPAR* contains a list of all the words the adventure understands. A utility that prints out disk files, or the following program will print a complete list of recognized words if *TPAR* is in memory (the periods just fill out commands requiring less than five letters).

APT: Print Commands

```
5 PR# 1 : PRINT
1Ø ST = 38281 :EN = 37888 + 1693
15 COL = 8Ø : REM # COLUMNS PRINTER SUPPORTS, SET TO FIVE FOR EACH
     RECOGNIZED WORD TO BE ON ITS OWN LINE
2Ø FOR I = ST TO EN : PRINT CHR$ ( PEEK (I ) );
3Ø NN = NN + 1 : IF NN = COL THEN PRINT CHR$ (13 ) :NN = 1
4Ø NEXT I : PRINT CHR$ (13 )
45 PR# Ø
```

Examining the BASIC program *TRANS* will give you some clues and show some interesting program techniques; it's worth looking at.

Another benefit of unlocking *Transylvania* is that the beautiful hi-res pictures can be enjoyed without the bother of traveling through the entire adventure. Owners of the *Graphics Magician* can load up the pictures, just as if they were any other picture created with the picture-editor, or those without can use the following program to take a look at the pictures (save to the *Transylvania* disk under *SEE PICTURES*).

APT: See Pictures

```
1Ø IF PEEK (1Ø3 ) + PEEK (1Ø4 ) < > 65 THEN POKE 1Ø3 ,1 : POKE 1Ø4 ,64
     : POKE 16384 ,Ø : PRINT CHR$ (4 ) "RUN^SEE^PICTURES"
2Ø TEXT : HOME : HGR : PRINT CHR$ (4 ) "BLOADPICDRAW2"
3Ø HOME : VTAB 21 : INPUT "SEE^PICTURE^->^" ;A$ : IF LEFT$ (A$ ,3 ) <
     > "CAT" THEN 5Ø
4Ø TEXT : HOME : PRINT CHR$ (4 ) "CATALOG" : PRINT : PRINT "ANY^KEY^
     " ; : GET A$ : PRINT A$ : HGR : GOTO 3Ø
5Ø IF A$ = "" OR A$ = "END" THEN TEXT : HOME : END
6Ø PRINT CHR$ (4 ) "BLOAD^" A$ ",A46Ø8" : POKE 256Ø ,Ø : POKE 2561 ,18
     : CALL 26Ø8
7Ø GOTO 3Ø
```

The possibilities for modification are endless; you can even modify the adventure map!

The Quest

Requirements:
Apple][, Apple][Plus, Apple //e or compatible
Super IOB
two blank disks

The softkey for *The Quest* is less complex than the softkey for *Transylvania*. You need to use the Penguin controller on the boot side of the disk and, interestingly enough, only *COPYA* on the back side.

After that, a couple of modifications need to be made. First of all, the boot program needs to be changed so that a file called *AMP 2.8* can be BLOADed into the DOS buffers. Type the following:

```
FP
10 PRINT CHR$( 4)"MAXFILES1": PRINT CHR$( 4)"BRUN QUEST"
SAVE QA
```

Copy DOS onto the disk and use *QA* as the boot-up program.

The next modification concerns a check to see if Penguin DOS is still in the machine. The check occurs in line 9120 of a program called *QB* so

```
LOAD QB
```

Change the **Z = PEEK (47092)** in line 9120 to a **Z = PEEK (47093)** and then:

```
SAVE QB
```

Like *Transylvania*, the disk shows no free sectors, but a CATALOG rebuilder (such as *FIXCAT* from *Bag of Tricks*) will recover the unused sectors which can now be used for saved games.

APT For The Quest

The end of this adventure is much, much better than that of *Transylvania*. In fact, to see the end without toiling with the adventure, flip the disk to side two and **BRUN EQA**. Excellent work on the part of the authors and on the COPYAable side - interesting.

Again, all the hi-res pictures may be viewed but, disappointingly, many are duplicates. The BASIC program *MQ* can also be examined to aid in solving this adventure.

Minute Man and more...

As I mentioned earlier, *Super IOB* works on other Penguin game disks as well. For example, running *Super IOB* on *Minute Man* and simply changing line one to **1 HOME: HGR: HGR2: PRINT CHR$(4)"MAXFILES1"** will create an unlocked *Minute Man* which, of course, may be copied with *COPYA* or any other copier, even *FID*!

Well, that's it! Watch for the return of three mice in *The Quest*, and have fun!

Rocky's Boots

The Learning Company

Tracking down Rocky's Boots
by Jerry Caldwell
(Hardcore COMPUTIST # 14, page 22)

Requirements:
Apple][Plus or equivalent
Super IOB
Sector-editor
Blank disk

Rocky's Boots is an educational program designed to instruct students of any age in the basic concepts of electronic digital logic. The user progresses from learning about electricity to the point where he/she is able to construct 'machines' from wires, logic gates, clocks, sensors and other pieces of hardware that will 'kick' targets of a specific color and/or shape. Unfortunately (or fortunately for those of us who like a challenge), *Rocky's Boots* is copy-protected.

The Learning Company uses two techniques to copy-protect *Rocky's Boots*. First, every sector on the disk is marked as if it were on track 0. This prevents the use of *COPYA* or any other standard copier to make a backup. The second copy-protection measure involves the use of half-tracks with track arcing during the boot process of the disk. On *Rocky's Boots*, tracks 3.0, 3.5 and 4.0 each contain five sectors of data.

While the first technique is easy to circumvent with the use of a bit copier, the latter technique is more difficult to defeat, even with a bit-copier which can read half-tracks. If full tracks of data are written to a disk in half-track increments, the data on adjacent half-tracks will tend to be obliterated. However, the half-tracks on an original copy of *Rocky's Boots* are written in a special manner so that they are synchronized with one another, with each of the half-tracks containing only five sectors of data. This pattern is generally very hard to duplicate with a bit-copier but can be read with the proper software. Both of the copy-protection techniques I've described can be removed so that a backup which resides on a normally formatted disk can be produced.

There are two basic things which must be done to produce the backup. First, all of the data that is written to the original must be moved to a backup disk. The data that resides on the three half-tracks on the original disk will be written onto one full track on the backup disk. This transfer can be handled by *Super IOB* with the *Rocky's Boots* controller listed on this page installed.

Once *Rocky's Boots* has been moved to a normally formatted disk, a couple of changes need to be made to the disk so that it will run properly. This involves some changes to the disk I/O code so that no half-tracks will be accessed and I/O errors will not be generated when the program finds that the sectors are marked normally instead of as if they were all on track $Ø. One other change has to be made so that one can exit from *Rocky's Boots* without having it crash into the Monitor.

Making The Copy

Begin by typing in the controller listed below. Install it into *Super IOB* and RUN.

controller

```
1ØØØ REM ROCKY'S BOOTS CONTROLLER
1Ø1Ø TK = Ø :ST = Ø :LT = 34 :CD = WR : POKE 48573 ,128
1Ø2Ø GOSUB 49Ø :T1 = TK :TK = Ø :CD = Ø : GOSUB 1ØØ : GOSUB 8Ø :S =
     -128 : GOSUB 13Ø
1Ø25 CD = RD :S = T1 * 2 : GOSUB 13Ø : RESTORE : GOSUB 17Ø :S = 2
1Ø3Ø TK = PH / 2 : GOSUB 43Ø :TK = Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST <
     DOS THEN 1Ø3Ø
1Ø4Ø IF BF THEN 1Ø6Ø1Ø5Ø ST = Ø : GOSUB 13Ø : IF PH = 6 THEN GOSUB
     111Ø
1Ø55 IF PH < LT * 2 THEN 1Ø3Ø
1Ø6Ø GOSUB 31Ø : GOSUB 49Ø :TK = T1 :ST = Ø : GOSUB 23Ø
1Ø7Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø7Ø
1Ø8Ø ST = Ø :TK = TK + 1 + (TK = 3 ) : GOSUB 115Ø : IF TK = 3 THEN GOSUB
     116Ø
1Ø85 IF BF = Ø AND TK < LT THEN 1Ø7Ø
1Ø9Ø IF TK < LT THEN 1Ø2Ø
11ØØ HOME : PRINT "DONE^WITH^COPY" : END
111Ø S = 1 :ST = 1 : GOSUB 116Ø : GOSUB 1ØØ :ST = Ø
112Ø FOR A1 = 1 TO 5 :TK = 3 :ST = ST + 1 : GOSUB 43Ø :TK = Ø : GOSUB
     1ØØ : NEXT
113Ø GOSUB 13Ø : IF PH < 9 THEN 112Ø
114Ø GOSUB 13Ø :S = 2 :ST = Ø
115Ø POKE 48683 ,185 : POKE 48684 ,184 : POKE 48685 ,191 : RETURN
116Ø POKE 48683 ,234 : POKE 48684 ,234 : POKE 48685 ,234 : RETURN
5ØØØ DATA 255 ,255 ,255 ,255
5Ø1Ø DATA 1^CHANGES
5Ø2Ø DATA Ø ,7 ,43 ,231
```

Note: During the copy process, the disk head will recalibrate just before reading the original disk. If it should recalibrate in the middle of reading the original, then you will get a bad copy.

A Few Controller Words

Here is a list of the differences in this controller from the standard one which makes it successful.

1Ø1Ø —Set the last track to be copied at 33, set extended error retry mode.

1Ø2Ø —Tell DOS to position over track Ø, recalibrate.

1Ø25 —Fix command code, move to correct track via MOVE S PHASES, alter the ending marks (to FF FF, FF FF), fix step variable for whole tracks.

1Ø3Ø —Calculate TK so that PRINT TRACK & SECTOR # will display the correct track, restore TK to zero since that is what the sectors are marked as.

1Ø5Ø —If on track 3 then call the track arcing reader at 111Ø.

1Ø6Ø —Call the sector-editor to edit track Ø, sector 7, byte $2B to $E7, normalize the DOS ending marks.

1Ø8Ø —Skip track 4, tell DOS to write logical sectors unless on track three in which case, tell DOS to write physical sectors.

111Ø —Set the step variable for half-tracks, tell DOS to read physical sectors, read sector 1 as a dummy, fix sector variable.

112Ø —Read the next five physical sectors.

113Ø —Step forward a half-track, if not done with the track arcing, then read five more sectors.

114Ø —Move on to track five, fix step variable, start with sector zero of track five.

115Ø —Restore DOS so that it reads logical sectors.

116Ø —Alter DOS so that it reads physical sectors.

Once the *Super IOB* copy of *Rocky's Boots* has been made, there are a few changes that have to be made to the disk so that it will function properly on a normally formatted disk.

Rocky's Boot-up

During the boot-up of the disk, some code is written to page $Ø4 (on the text page) that is responsible for accessing the data on the half-tracks. After this has been accomplished, the code exits to $5ØØ to where the menu is read in. Putting code on the text page like this is also a form of copy-protection.

An interesting fact about *Rocky's Boots* is that it always uses the sector read routine at $C*x*5C (*x* being the slot number) to read the disk. That is, the original *Rocky's Boots* uses this ROM subroutine. The modification we will make moves this routine into RAM where it can be modified to suit our purposes.

This change modifies the code that is written to page $Ø4 so that it does not access the half-tracks, but instead, reads the data that was written on tracks 3.0, 3.5 and 4.0 all from track $03. After this has been done, the code will move the sector-read routine from the disk controller card to $400—$49E and then modify it so that the normally-marked sectors will not cause any errors. Our code will also make a modification at $508—$50A so that the code which was moved to $400—$49E will be used instead of the routine in the disk controller's ROM. This change has to be made to track $00, sector $07 (logical) on the *Super IOB* copy of *Rocky's Boots*.

To make this change, get out your sector-editor and read in track $00, sector $07 of the backup. Move the cursor to byte $AE of this sector and start entering the bytes listed in the hexdump below. Write the sector back to the disk when you have finished entering all of the bytes. For those who are interested, the source listing of this is included.

```
A2  0F  20  0C  04  20  2A  04
46  4A  A9  A4  85  76  A9  03
85  77  A9  00  85  78  A5  3F
85  79  A0  5C  B1  78  91  76
C8  D0  F9  A9  41  8D  43  04
A9  04  8D  0A  05  A9  00  8D
09  05  A6  2B  86  EF  4C  00
05  E6  41  20  2D  04  60
```

A Graceful Exit

The final modification is made to track $1, sector $07 of the backup so that the drive will reboot properly when the "END" option is chosen from the main menu. This modification is necessary because with the previous sector-edit we tricked the program into thinking the disk controller ROM was located on the text page (page $04). The program will just crash if it tries to reboot from there. So, to restore a graceful exit to *Rocky's Boots*, make the following change to the *Super IOB* copy.

Track	Sector	Byte	From	To
$01	$07	$03	$AC	$A4
$01	$07	$04	$0A	$3F
$01	$07	$05	$05	$EA

Once you have written the change back to the disk, you will have a fully functional backup of *Rocky's Boots*.

Those of you who would like to learn how *Rocky's Boots* works might be interested to know that each of the items on the menu (Rocky's Boots, Rocky's Challenge, etc) has an entry point of $A00. If you have some means of halting without a reboot (old F8 ROM or a NMI card, etc.) you can stop the program, snoop around memory to your heart's content and then restart the module in memory with a A00G. Just keep in mind that the code necessary for disk access resides on the text page and it will hit the proverbial bit bucket as soon as the Monitor is entered.

Once the program has been halted, you will have to reboot the disk in order use the main menu.

Other Learning Company Programs

Other Learning Company programs are copy-protected in a similar fashion to what I have described for *Rocky's Boots*. There are several ways to approach these programs.

One of the easiest approaches is to make a bit-copy of the original and boot it with the cover of the disk drive removed. The head will probably move inward until it tries to read from a track that should have adjacent half-tracks. You may then replace the copy with the original, close the door briefly and then re-open it. As the Apple reads the data in, you can determine which are the half-tracks. By alternating between the bit-copy and the original copy, you can determine where the half-tracks end and when full tracks are again being accessed. Once the half-tracks have been read, the bit-copy will probably function just as well as the original.

The sector responsible for reading in the half-tracks probably will be sector $Ø1 of track $ØØ on other Learning Company disks, just as it is on *Rocky's Boots*. You should be able to read this sector with the *Inspector* or *Tricky Dick* (set the end-of-address and data-marks to ØØØØØØ). Look for instructions like:

```
B9 81 CØ LDA CØ81,Y
B9 8Ø CØ LDA CØ8Ø,Y
```

These are instructions which control the head movement of the drive. Look also for instructions which call subroutine which are displaced three steps away in memory, such as:

```
8ØØ:  2Ø Ø3 Ø8  JSR $Ø8Ø3
8Ø3:  48        PHA
8Ø4:  98        TYA
8Ø5:  48        PHA
```

If a call to $8Ø3 causes the head to increment a half-track, then a call to $8ØØ will cause two half-track increments, in other words, a whole track (for an example of this, look at the code which starts at byte $2A of track $Ø, sector $7 on *Rocky's Boots*). Having found the appropriate sector, one still needs to recover the data from the half-track arcs. Knowing the location in memory to which the data goes, an Integer card could be used (as it could have in our example here) to recover that data. You would need to dissasemble the sector to trace that information. With other programs, it may not be easy to find a place in the RAM memory to place the disk controller ROM as we were able to do with *Rocky's Boots*. In such a case, you should plan on only modifying track Ø and the half-track arcs, and backup the remainder of the disk with a bit copier.

source code

```
* Patch this code to track $Ø, sector $7 of the 1OB copy of Rocky's Boot.
* The code will move the disk controller ROM Sector Read routine onto
* the text page where it will be used to read the normalized copy
                                     .OR $4AE
                                     .TA $8ØØ
                                     .TF ROCKY CODE
Ø4EA:  A2 ØF                         LDX #$ØF      START ON SECTOR $F
Ø4EC:  2Ø ØC Ø4                      JSR $Ø4ØC     READ SUBROUTINE
Ø4EF:  2Ø 2A Ø4                      JSR $Ø42A     MOVE A FULL TRACK
Ø4F2:  46 4A                         LSR $4A       RESET TO LOGICAL SECTORING
Ø4F4:  A9 A4                         LDA #$A4      SET UP THE ZERO
Ø4F6:  85 76                         STA $76       PAGE POINTERS
Ø4F8:  A9 Ø3                         LDA #$Ø3      TO MOVE THE ROM
Ø4FA:  85 77                         STA $77       SECTOR READ ROUTINE
Ø4FC:  A9 ØØ                         LDA #$ØØ      AT $CX5C TO $4ØØ
Ø4FE:  85 78                         STA $78
Ø5ØØ:  A5 3F                         LDA $3F       $3F HOLDS HI BYTE OF
Ø5Ø2:  85 79                         STA $79       CONTROLLER'S ROM ADDRESS
Ø5Ø4:  AØ 5C                         LDY #$5C      OFFSET INTO ROM
Ø5Ø6:  B1 78         MOVEIT          LDA ($78),Y   MOVE THE CODE
Ø5Ø8:  91 76                         STA ($76),Y
Ø5ØA:  C8                            INY
Ø5ØB:  DØ F9                         BNE MOVEIT
Ø5ØD:  A9 41                         LDA #$41      MAKE A MOD TO PREVENT
Ø5ØF:  8D 43 Ø4                      STA $Ø443     I/O ERRORS
Ø512:  A9 Ø4                         LDA #$Ø4      MODIFY THE CODE ON
Ø514:  8D ØA Ø5                      STA $Ø5ØA     PAGE $5 SO THAT STA $5Ø7
Ø517:  A9 ØØ                         LDA #$ØØ      THE CODE AT $4ØØ LDA #$ØØ
Ø519:  8D Ø9 Ø5                      STA $Ø5Ø9     IS USED TO READ DISK STA $5Ø6
Ø51C:  A6 2B                         LDX $2B       $2B HOLDS SLOT #
Ø51E:  86 EF                         STX $EF
Ø52Ø:  4C ØØ Ø5                      JMP $Ø5ØØ     EXIT TO $5ØØ
Ø523:  E6 41                         INC $41       PATCH THAT INCREMENTS
Ø525:  2Ø 2D Ø4                      JSR $Ø42D     THE TRACK #
Ø528:  6Ø                            RTS
```

***Note:** Some versions may require the change below:*

```
Ø514                                 STA $5Ø7
Ø517                                 LDA #$ØØ
Ø519                                 STA $5Ø6
```

Sabotage

On-Line Systems

Backup For Sabotage
by Clay Harrell
(Hardcore COMPUTIST # 12, page 7)

Requirements:
Apple][, Apple][Plus, or Apple //e
A blank initialized DOS 3.3 disk

Sabotage was about the first game I ever bought for my Apple way back when DOS 3.3 had just been released. Even though it is a somewhat simple game, it held my attention for quite some time and even now I occasionally still play it.

The last time I tried to play *Sabotage*, I noticed that the original disk was having a hard time loading the hi-res title page. No doubt, the disk had seen its day and needed to be backed-up. I figured I could still save it before it was too late.

The protection used by On-Line was somewhat simple when this program was released (back in the good ol' days when Sierra On-Line was just On-Line Systems). The disk is in a modified DOS 3.2 format which was another good reason for backing it up (this thing boots slooooowly!).

You can boot the disk and press CTRL C to interrupt the Hello program before it executes. You may now CATALOG the disk and examine the files if you wish.

The Hello program BLOADs the banner picture and just BRUNs the game (file: *SABOTAGE*). What we want to do is BLOAD SABOTAGE, then check the locations it was loaded at and its length:

```
BLOAD SABOTAGE
```

and then enter the Monitor with:

```
CALL -151
```

Then type:

```
AA60.AA73
```

The last two bytes listed will be the location *SABOTAGE* was loaded at, and the first two bytes will be the length of the file, in bassackward order, of course. If you list memory at $1DØØ, you will see that there are some memory moves to page $Ø1 and an RTS. If you type **$1D00G** RETURN, the drive starts up and makes sure you're using the original disk.

Obviously, we don't want this routine in our final production. Well, past that, the starting address is $1D1F and the file continues up to $54ØØ.

So, to backup *Sabotage* we just BLOAD the file *SABOTAGE*, boot a 48K slave disk and BSAVE the file.

In cookbook fashion, here are the steps to deprotection:

1 Boot the *Sabotage* disk.

2 Press CTRL C immediately, and after the Hello program is loaded, your Apple will beep and you will be in Applesoft BASIC.

3 Type:

BLOAD SABOTAGE

4 Boot a 48K DOS 3.3 slave disk by typing:

PR#6

5 Save the program to your disk by typing:

BSAVE SABOTAGE,A$1D1F,L$36E2

Seadragon

Adventure International

Softkey For Seadragon
by Jeff Rivett
(Hardcore COMPUTIST # 14, page 8)

Requirements:
Apple][
Super IOB

When I first encountered *Seadragon*, like most people, I was fairly impressed by its animation and sound routines. However, I didn't play it too often because I was not able to back it up, and I don't like using originals.

The program boots very much like a normal disk. You can get a good copy up to the title page with any bit-copier, but when you try to run the game, it will load and then start to do a strange thing. It sounds as if the disk drive is having a spasm, continually moving from track $Ø to track $22 until you hit RESET or turn the machine OFF. But the designers have nicely set the reset vector so that you cannot stop the reboot by repeatedly hitting RESET.

Count Nibble and Friends

The program is actually doing a funny kind of nibble-count. If you look at track $22 with a nibble-editor, you will see a repeating pattern of FF's and DD's. This pattern is what the copy-protection looks for, and it is fairly hard to copy. I found that *Copy II Plus* will do it on default parameters if you fiddle with your drive speed.

Once I was able to copy the disk, I started to really get into the game. There, I ran into another old enemy, the Arcade Mentality: the game is just too difficult. I felt I had been let down, and started thinking about how I could get some satisfaction from the game. That's when I thought about modifying it, which means first removing the copy-protection.

The protection on this disk comes in stages. First, the End-of-data marks have been changed from **DE AA** to **AA DE**. The DOS is very much like normal DOS, and to allow it to read the different marks, the read-error-routine has been turned off. Pretty clumsy, right? The only sector on the disk without these different marks is the one used to store high scores, track 3, sector Ø. In any case, I dealt with this problem by writing a *Super IOB* controller which fits these requirements.

The disk's DOS does not need to be changed to read the new, unprotected format, but I wanted to normalize the disk as much as possible, so I made the *Super IOB* controller edit track Ø, sector 3 as shown in the list below.

The next problem was the nibble-count. Since the disk was now easily read with a sector-editor, I reasoned that it should be possible to locate the nibble-count-routine and turn it off. I found this routine at track $19, sector $ØE.

Unscrambling

But the Protectors hadn't finished yet. After I turned off the nibble-count, I booted the copy. The game started to run and then some very funny things happened. The mines seemed to be all over the place. And when I got far enough into the cave, the game simply hung. There is an area that is used for game data at $95ØØ which seems to get scrambled when you turn the nibble-count off. But the routine that does this can be found at track $19, sector $Ø8.

Now the game seemed to run perfectly, and continued to do so just as long as I never used torpedoes. Well that certainly wasn't much fun. Another routine had been activated that was forcing an early link to the next level, that is, the end of the cave where you meet the *Seadragon*. When I pushed the torpedo button, this part of the game loaded and I was actually able to play it. You might want to try this just for fun if you've never seen the *Seadragon*. The routine that messes up this part of the game is at track $15, sector $ØE.

After I located and turned off these routines, I found that the game ran normally except that it took slightly longer to load. This was simply due to the fact that I had turned the read-error-routine on.

So to make a working, *COPYA*-able copy, load *Super IOB* and install the controller listed at the end of this article. Next, run it on *Seadragon*. The controller will make these sector changes:

Track:	$ØØ	$ØØ	$ØØ	$ØØ	$19	$19	$19
Sector:	$Ø3	$Ø3	$Ø3	$Ø3	$ØE	$Ø8	$ØE
Byte:	$36	$37	$3F	$4Ø	$ØØ	$25	$E7
From:	$EA	$EA	$ØØ	$DØ	$4C	$A2	$A9
To:	DØ	$ØA	$AA	$FØ	$6Ø	$6Ø	$6Ø

The Advanced Playing Techniques

There are lots of things out to get you in this game. Using the list I have compiled, you can now customize the game so that the more insidious ones just disappear. There are also a few irritating noises, which may now be turned on and off at will. I suggest, however, that you make a backup of your new copy before you get too carried away.

APT type	Track	Sector	Byte	From	To
Stop voice game start:	$04	$02	$B8	$AD	$60
Stop sub explosion sound:	$16	$0D	$DF	$AD	$EA
			$E0	$30	$EA
			$E1	$C0	$EA
Stop irritating noise:	$17	$07	$8F	$AD	$EA
			$90	$30	$EA
			$91	$C0	$EA
			$BD	$AD	$EA
			$BE	$30	$EA
			$BF	$C0	$EA
Mine speed ($00=off):	$14	$0C	$DE	$05	$00-$09
Eel speed ($FF=off):	$14	$08	$0B	$04	$00-$FF
Seaweed speed ($FF=off):	$14	$09	$59	$01	$00-$FF
Seaflea speed ($00=stop):	$14	$06	$15	$02	$00-$02
Stalactite speed ($0=stop):	$14	$07	$47	$04	$00-$09
Turn OFF force field:	$14	$0A	$33	$46	$00
	$15	$07	$62	$05	$00
Turn OFF shooters:	$14	$0A	$E3	$01	$FF
Unlimited damage:	$18	$0B	$6E	$8D	$EA
			$6F	$2E	$EA
			$70	$43	$EA
Unlimited air:	$18	$0B	$32	$01	$00
Free sonic disruptor:	$18	$0B	$59	$05	$00

Other Stuff

The main portion of the game is on tracks $12 to $19. After the sub blows up, the game restarts at $8868. The actual amount of air you have is stored at $432B—$432C. Damage is at $432E. There are many other interesting locations, including $4DB0 and $57F3.

How I Did It

Most of the detective work I had to do to de-protect this disk was done with the aid of my *Replay II* card and *Nibbles Away II*. *Replay* allowed me to enter the Monitor at any time and look at the code. When I found a suspicious looking routine, I searched the disk using the disk-scan utility in the sector-editor part of *Nibbles Away II*. After locating the routine, I was able to disassemble it from within the sector-editor, make a calculated change, and boot the copy to see what, if anything, had changed.

Closing Remarks

I found that being able to control the game really increased my enjoyment of it. For example, I was having a lot of trouble figuring out how to get past the Seafleas, so I turned everything else off so that I could concentrate on them alone. You may want to turn everything off just to try navigating the cave from beginning to end. Have fun.

controller

```
1ØØØ REM SEA DRAGON CONTROLLER
1Ø1Ø TK = Ø :ST = Ø :LT = 34 :CD = WR
1Ø2Ø T1 = TK : GOSUB 49Ø : RESTORE : GOSUB 17Ø
1Ø3Ø GOSUB 43Ø : IF ST = 1 AND TK = 3 THEN RESTORE : GOSUB 17Ø
1Ø35 GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø3Ø
1Ø4Ø IF BF THEN 1Ø6Ø
1Ø5Ø ST = Ø :TK = TK + 1 : IF TK = 3 THEN GOSUB 23Ø
1Ø55 IF TK < LT THEN 1Ø3Ø
1Ø6Ø GOSUB 23Ø : GOSUB 31Ø : GOSUB 49Ø :TK = T1 :ST = Ø
1Ø7Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø7Ø
1Ø8Ø ST = Ø :TK = TK + 1 : IF BF = Ø AND TK < LT THEN 1Ø7Ø
1Ø9Ø IF TK < LT THEN 1Ø2Ø
11ØØ HOME : PRINT "DONE^WITH^COPY" : END
5ØØØ DATA 222 ,17Ø ,17Ø ,222
5Ø1Ø DATA 7^CHANGES
5Ø2Ø DATA 25 ,14 ,Ø ,96 ,25 ,8 ,37 ,96
5Ø3Ø DATA 21 ,14 ,231 ,96
5Ø4Ø DATA Ø ,3 ,54 ,2Ø8 ,Ø ,3 ,55 ,1Ø
5Ø5Ø DATA Ø ,3 ,63 ,17Ø ,Ø ,3 ,64 ,24Ø
```

Sensible Speller IV

Sensible Software

Sensible Speller IV: UPDATE
by Doni G. Grande
(Hardcore COMPUTIST # 11, page 24)

Requirements:
Apple][or Apple][Plus
16K RAM card
(If you don't have a RAM card, see note at end of article)
Blank disk
Any DOS track-copy program

Editor's Note: The softkey that was published in **The Book Of Softkeys Volume II** *(Hardcore COMPUTIST # 9 for* ***Sensible Speller IV*** *was based upon revision 4.0d and, unfortunately, would only work correctly on that revision. Apparently there have been many minor revisions (over a dozen) to* ***Sensible Speller*** *and most of these revisions have changed the program to the extent that it no longer has the same startup point nor does it occupy the same memory range. If the former softkey is attempted on these revised versions, the results are usually disappointing: the copy will either hang up after loading the* ***Sensible Speller*** *logo or will give a* ***CHECKSUM ERROR*** *message when the main menu appears. Don't despair, though, because with some modifications to one of the programs (SPELLER.LOADER) presented in the original article the softkey* **can** *be performed successfully on any revision of* ***Sensible Speller IV*** *(at least up to revision 4.2b anyway).*

What It Does

Let us first delve a little deeper into what the original softkey does. This demonstrates an excellent use for a RAM card. A little-realized fact about RAM cards is that they completely ignore the RESET key. When power is first applied to the Apple, the RAM card's own RESET circuitry is set to certain power-up defaults. After that, whenever it is enabled by use of the soft switches (see the manual that came with the card if you do not know about soft-switches), it keeps control until it is specifically turned off. That is why it was possible to reset into the Monitor in the original softkey by just using the RAM card. This technique is normally possible only with an Integer card or modified F8-ROM! Those of you out there cursing your lack of an Integer card should note the use of the RAM card here.

To determine why the original softkey works (at least on 4.Ød), get out your copy of **The Book Of Softkeys Volume II** or Hardcore COMPUTIST # 9 and follow along.

First, the RAM card is moved to slot one, which is a non-standard slot. As mentioned in the article, most software only looks in slot zero for a RAM card, so it is effectively hidden by moving it to slot one. Also, as mentioned above, once the RAM card has control of the Apple, not even RESET will make it let go! So an active RAM card in slot one is not likely to be disabled by software and has the full power of a 'normal' (slot zero) RAM card.

Next, a disk with a somewhat modified DOS is made. This modification consists of a 'patch' in the $B6 page of DOS (A page of memory is 256 locations, which works out in hexadecimal to mean that the first two digits of the address do not change. Hence, page $Ø3 refers to $Ø3ØØ—$Ø3FF). Page $B6 ($B6ØØ—$B6FF) resides on the disk on track $Ø, sector $Ø and is the first sector to be read into memory when a disk is booted. It is read into memory page $Ø8 where, upon execution, it normally reads track $Ø, sectors $Ø through $9 into memory at $B6ØØ—$BFFF. This the portion of DOS containing the RWTS (Read/Write Track & Sector) which is responsible for all the gory details of disk operation. Track $Ø, sector $Ø is re-read into page $B6 so that RWTS has something to put on the disk so that it can boot. The patch made to the $B6ØØ area in the original softkey enables a RAM card in slot zero, copies the motherboard language into it, sets up a location on page $Ø2, and then jumps to another patch at $B7ØØ.

The patch at $B7ØØ is installed in step 7 of the softkey. This is the real core of the procedure since this patch loads *Sensible Speller* into memory from the disk when it is booted. The *SPELLER.SAVER* routine in the former article stores the menu and utilities on the disk in a certain way. The following table is a track/sector map of the first seven tracks and their respective memory locations.

Track:→	Ø	1	2	3	4	5	6	7
Sector:↓ ØØ	B6	ØB	x	x	1B	2B	4Ø	5Ø
Ø1	B7	ØC	x	x	1C	2C	41	51
Ø2	B8	ØD	x	x	1D	2D	42	52
Ø3	B9	ØE	x	x	1E	2E	43	53
Ø4	BA	ØF	x	x	1F	2F	44	54
Ø5	BB	1Ø	x	x	2Ø	3Ø	45	55
Ø6	BC	11	x	x	21	31	46	56
Ø7	BD	12	x	x	22	32	47	57
Ø8	BE	13	x	x	23	33	48	58
Ø9	BF	14	x	x	24	34	49	59
ØA	71	15	x	x	25	5	4A	5A

The hi-res *Sensible Speller* logo is stored on tracks $6 and $7. The first thing that the patch at $B7ØØ does is to display hi-res page two ($4ØØØ—$5FFF) and load this logo from the disk. Then, $Ø8ØØ—$3AØØ is loaded from tracks $Ø—$5 (in reverse order), skipping over tracks $2 and $3 which are not used by the copy routine. Page $3 is loaded from track $Ø, sector $ØC and then $77ØØ and $71ØØ are loaded from track $Ø, sectors $ØB and $ØA respectively. I assume that $77ØØ must be used in some way by *Sensible Speller 4.Ød*; later versions do not use it. Page $71 was used by the RESET trap routine to store page $Ø. All the important parts of memory are restored to their original contents when the disk is booted. The last task of the patch at $B7ØØ is to restore zero page and jump to the *Speller* program. This patch is stored to the disk on track $Ø, sector $1 in steps 8 through 1Ø. Then, $Ø8ØØ to $3AØØ is loaded from tracks $Ø, $1, $4 and $5 with the use of the normal DOS RWTS.

Getting back to the original procedure, next the unprotected tracks ($2—$3 and $6—$22) on the original disk are copied with a bit-copier or by *Super IOB*. Then, the *SPELLER.SAVER* routine is patched into the RAM card, the original disk is booted, and RESET is pressed. The patch entered in step 17 saves page $Ø at $71ØØ when RESET is pressed, allowing the *SPELLER.SAVER* routine to save a complete snapshot of memory to disk. When the *SPELLER.SAVER* program is run (DØØØG in step 22), it writes the important areas of memory to disk at the tracks and sectors shown in the preceeding table. When the disk is booted, *Sensible Speller* is loaded back into memory and started.

What Goes Wrong

I found that on later revisions of *Sensible Speller* the menu and utilities take up two or three more pages of memory than the original softkey saves to disk. If the entire menu is not present in memory, the copy of *Sensible Speller* will print a *CHECKSUM ERROR* message when the menu appears. The *SPELLER.SAVER* program saves the menu and utilities to the unprotected disk on tracks $Ø, $1, $4 and $5, but I found out that the same code also exists on tracks $8 through $B which are copied during step 11 of the softkey.

Another problem with the original softkey is that the entry point of $33D9 is correct only for 4.Ød. Each of the revisions seems to have a different entry point and if the entry point is not correct, the copy will usually just load in the *Sensible Speller* hi-res logo, switch to the text screen and then hang.

One thing to notice about an original copy of *Sensible Speller* is that the program will always return to the main menu if the RESET key is hit (at least with an autostart F8-ROM). This is a clue that the reset vector at $3F2 somehow points to the correct menu entry point. Because the original softkey saves page $Ø3 on track

$Ø, sector $ØC, the values in the RESET vector can be viewed with a sector-editor. The values in $3F2—$3F3 turn out to be **$F8 Ø3**, which points at $Ø3F8. At $3F8 the Assembly language instruction reads JMP ($ØØ4E) which is an indirect JMP to the values at $4E—$4F. Therefore, for all versions of *Sensible Speller* to work correctly, the last thing the *SPELLER.LOADER* program should do is to perform a JMP ($ØØ4E).

The modifications I have made to the *SPELLER.LOADER* program load the menu and utilities from tracks $8—$B into memory at $Ø8ØØ—$3FFF. Some of these sectors on track $B may or may not be needed but the program loads them in regardless so that the procedure will work with all of the different revisions. Another modification I made was to have the *SPELLER.LOADER* program lay track $2, sector $ØF over page $Ø9 in memory where the setup is contained. This ensures that the latest setup is used each time the disk is booted. Finally, my modified *SPELLER.LOADER* program performs the JMP ($ØØ4E) to enter the menu.

Note: *This assumes that you have already tried making a copy using the softkey from the previous Book Of Softkeys (Hardcore COMPUTIST # 9). This non-working copy will work if you:*

1 Boot the DOS 3.3 master disk.

2 Insert the copy disk made using the original procedure, then enter the Monitor and type in this hexdump:

```
CALL -151
B7ØØ: 2C 5Ø CØ 2C 57 CØ 2C 52
B7Ø8: CØ 2C 55 CØ A9 ØF 8D ED
B71Ø: B7 A9 Ø7 8D EC B7 A2 Ø1
B718: 8E EA B7 CA 8E FØ B7 A9
B72Ø: 5F 8D F1 B7 2Ø 7F B7 AD
B728: F1 B7 C9 4Ø BØ F6 A9 ØB
B73Ø: 8D EC B7 A9 Ø7 8D ED B7
B738: 2Ø 7F B7 AD F1 B7 C9 Ø8
B74Ø: BØ F6 A9 Ø3 8D F1 B7 A9
B748: ØØ 8D EC B7 A9 ØC 8D ED
B75Ø: B7 2Ø 7F B7 A9 77 8D F1
B758: B7 2Ø 7F B7 A9 71 8D F1
B76Ø: B7 2Ø 7F B7 A9 Ø2 8D EC
B768: B7 A9 ØF 8D ED B7 A9 Ø9
B77Ø: 8D F1 B7 2Ø 7F B7 AD 51
B778: CØ AD 54 CØ 4C 9C B7 A9
B78Ø: Ø1 8D F4 B7 A9 B7 AØ E8
B788: 2Ø B5 B7 CE ED B7 1Ø Ø8
B79Ø: A9 ØF 8D ED B7 CE EC B7
B798: CE F1 B7 6Ø A2 ØØ BD ØØ
B7AØ: 71 95 ØØ E8 DØ F8 6C 4E
B7A8: ØØ
```

If you want to save this program in the event of an error:

BSAVE SPELLER.LOADER.MOD,A$B7ØØ,L$A9

3 Use RWTS to write page $B7 to track $Ø, sector $Ø:

```
8Ø3:A9 B7 AØ E8 4C B5 B7
B7EB:ØØ ØØ Ø1
B7FØ:ØØ B7 ØØ ØØ Ø2
8Ø3G
```

The disk will now boot and work normally.

If you have not tried the softkey from **The Book Of Softkeys Volume II** (Hardcore COMPUTIST # 9), just use the *SPELLER.LOADER.MOD* from this article in place of the original *SPELLER.LOADER*. The rest of the steps in the original article can be followed without modification.

SPELLER.LOADER.MOD source code

```
11ØØ  ------------------------------------------------
111Ø  SPELLER.LOADER.MOD
112Ø  Loads what used to be part of Sensible Speller bootcode
113Ø  Modification to SPELLER.LOADER originally printed in
114Ø  Volume II of The Book Of Softkeys (Hardcore COMPUTIST 19)
115Ø  and works on Sensible Speller IV up to revision 4.2b.
116Ø
117Ø  Note that some code was rearranged to make room for the
118Ø  new code. The free space for patches is $B7ØØ-$B7B4.
119Ø  RWTS entry at $B7B5 must not be disturbed!
12ØØ  *-----------------------------------------------
121Ø                .OR $B7ØØ
122Ø                .TF SPELLER.LOADER.MOD
123Ø  *
124Ø  * DOS 3.3 RWTS Parmlist
125Ø  *
126Ø  DRIVE         .EQ $B7EA
127Ø  TRACK         .EQ $B7EC
128Ø  SECTOR        .EQ $B7ED
129Ø  BUFHI         .EQ $B7F1
13ØØ  COMMAND       .EQ $B7F4
131Ø  RWTS          .EQ $B7B5
132Ø  *
133Ø  * Display Hi-res page 2
134Ø  *
135Ø                BIT $CØ5Ø
136Ø                BIT $CØ57
137Ø                BIT $CØ52
138Ø                BIT $CØ55
```

```
1390  *
1400  * Load the SS logo
1410  * from tracks 6 & 7
1420  *
1430           LDA #$ØF
1440           STA SECTOR
1450           LDA #$Ø7
1460           STA TRACK
1470           LDX #$Ø1
1480           STX DRIVE
1490           DEX
1500           STX BUFHI-1
1510           LDA #$5F
1520           STA BUFHI
1530  LOOP 1   JSR READ
1540           LDA BUFHI
1550           CMP #$4Ø
1560           BCS LOOP1
1570  *
1580  * Load menu-utils from
1590  * trks $8-$B into mem
1600  * $Ø8ØØ-$3FFF.
1610  *
1620           LDA #$ØB
1630           STA TRACK
1640           LDA #$Ø7
1650           STA SECTOR
1660  LOOP 2   JSR READ
1670           LDA BUFHI
1680           CMP #$Ø8
1690           BCS LOOP2
1700  *
1710  * Load pg $Ø3 from
1720  * trk $Ø, sct $ØC.
1730  *
1740           LDA #$Ø3
1750           STA BUFHI
1760           LDA #$ØØ
1770           STA TRACK
1780           LDA #$ØC
1790           STA SECTOR
1800           JSR READ
1810  *
1820  * Get $77ØØ from trk $Ø
1830  * sct $B. This may not be
1840  * needed for your rev.
1850  *
1860           LDA #$77
1870           STA BUFHI
1880           JSR READ
1890  *
1900  * Get $71ØØ (SS page $Ø)
1910  * from trk $Ø, sct $A.
1920  *
1930           LDA #$ 71
1940           STA BUFHI
1950           JSR READ
1960  *
1970  * Overlays the setup
1980  * code over pg $Ø9
1990  * from trk $2, sct $F.
2000  *
2010           LDA #$Ø2
2020           STA TRACK
2030           LDA #$ØF
2040           STA SECTOR
2050           LDA #$Ø9
2060           STA BUFHI
2070           JSR READ
2080  *
2090  * Set the screen to text
2100  * page 1 and exit.
2110  *
2120           LDA $CØ51
2130           LDA $CØ54
2140           JMP EXIT
2150  *
2160  * READ modified to dec.
2170  * BUFHI after each read
2180  * to save code space
2190  *
2200  READ     LDA #$Ø1
2210           STA COMMAND
2220           LDA #$B7
2230           LDY #$E8
2240           JSR RWTS
2250           DEC SECTOR
2260           BPL RTS1
2270           LDA #$ØF
2280           STA SECTOR
2290           DEC TRACK
2300  RTS1     DEC BUFHI
2310           RTS
2320  *
2330  * Restore pg $Ø frm $71ØØ
2340  * and jump to SS IV entry
2350  *
2360  EXIT     LDX #$ØØ
2370  LOOP3    LDA $71ØØ,X
2380  STA $Ø,X
2390  INX
2400  BNE LOOP3
2410  *
2420  * JMP indirect to Menu
2430  *
2440           JMP ($ØØ4E)
```

No RAM card?

Readers who wish to make a back up of *Sensible Speller IV* but who do not have a removable RAM card should refer to page 82 of **The Book Of Softkeys Volume II** or Hardcore COMPUTIST # 10, page 6.

The procedure described in that article doesn't require a RAM card but does require that the entry point to the *Sensible Speller* menu be known for the particular revision being backed up.

Entry points for revisions 4.0c and 4.1c were given in the article and we have since learned the menu entry points for several other revisions. The table below gives the menu points for several different revisions currently in circulation.

The correct entry point should be used in line 1070 of the source code or at address $B791—$B792 of the object code.

If you don't have one of the versions listed try using an entry point of $0800.

Revision	Entry Point
4.0c:	$33B8
4.0d:	$33D9
4.0h:	$351A
4.0i:	$3538
4.0j:	$3522
4.1b:	$3514
4.1c:	$3517
4.2a:	$3584
4.2b:	$3586

Snooper Troops Case #2

Spinnaker Software

Deprotecting Snooper Troops (Case #2)
by Jim Mitchell
(Hardcore COMPUTIST # 13, page 7)

Requirements:

Apple][
One blank diskette
DOS 3.3 System Master
Super IOB

Snooper Troops Case #2 is an educational game by Spinnaker Software for ages 1Ø to adult. This game and *Snooper Troops I* are copy-protected in exactly the same way and cannot be backed-up using the usual copy programs. However, they can be transformed into *COPYA* form without too much difficulty.

The copy-protection consists of changing the address-field-header from **$D5 AA 96** to **$BB AA 96**, and leaving track $Ø9 empty (actually filled with $FF's). When *Snooper Troops* is copied with *Locksmith* or *Copy II Plus*, the disk will boot and send you off to track $Ø9 where you will stay until you boot another disk. To change this program into the friendlier *COPYA* form, all you need to do is use a *Super IOB* controller to read the address-field-header as **$BB AA 96**, and write the track back onto an empty disk with the normal **$D5 AA 96** address-field-header. When all the tracks have been transferred to the new disk, DOS 3.3 will be written onto the disk using the *MASTER CREATE* program from the System Master diskette. You should then have an unprotected copy of *Snooper Troops, Case #2*, which you may examine or modify at your leisure.

Procedure:

1 Type in and save the *Super IOB* controller at the end of this article.

2 Save the program in case of an error. I also suggest that you write-protect your original copy of *Snooper Troops* to protect it during this procedure, but be sure to remove it before running the program or it will not run properly.

3 Run *Super IOB* and follow the prompts that the program will display on your monitor. Note that your drive will make noise when reading tracks $ØØ and $Ø9. Do not interrupt the procedure. Just ignore the noise if possible; it will not affect the finished product. Also, note that tracks $ØØ—$Ø2 will not be copied.

4 When the copy is completed, remove the original copy of *Snooper Troops* and place it in a safe location.

5 Insert your System Master diskette and execute the *MASTER CREATE* program.

BRUN MASTER CREATE

6 When asked for the name of the greeting program, type HELLO.

7 Now insert your new copy of *Snooper Troops* that you just made with *Super IOB* and press RETURN. DOS 3.3 will now be written onto the disk and, when the disk is booted, the *HELLO* program will automatically run.

You may now CATALOG the disk, examine the program, and modify it if you like. You may also copy the new disk using *COPYA* or any other copy program.

Note: This program will also allow you to make a backup copy of *Snooper Troops, Case #1*. I have not attempted to use it on any other programs from Spinnaker Software, but I suspect that it might work on some of them. It will also work on *Piece of Cake*, an educational math program by Counter Point Software, which I found to have a nearly identical protection scheme. No changes to the program are necessary.

controller

```
1ØØØ REM SNOOPER TROOPS
1Ø1Ø TK = 3 :ST = Ø :LT = 35 :CD = WR
1Ø2Ø T1 = TK : GOSUB 49Ø : RESTORE : GOSUB 19Ø
1Ø3Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø3Ø
1Ø4Ø IF BF THEN 1Ø6Ø
1Ø5Ø ST = Ø :TK = TK + 1 : IF TK = 9 THEN TK = 1Ø
1Ø55 IF TK < LT THEN 1Ø3Ø
1Ø6Ø GOSUB 23Ø : GOSUB 49Ø :TK = T1 :ST = Ø
1Ø7Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø7Ø
1Ø8Ø ST = Ø :TK = TK + 1 : IF TK = 9 THEN TK = 1Ø
1Ø85 IF BF = Ø AND TK < LT THEN 1Ø7Ø
1Ø9Ø IF TK < LT THEN 1Ø2Ø
11ØØ HOME : PRINT : PRINT "DONE^WITH^COPY" : END
63Ø1Ø DATA 187 ,17Ø ,15Ø
```

SoftPorn Adventure

Sierra OnLine Systems

Softkey For SoftPorn Adventure
by Wes Felty
(Hardcore COMPUTIST # 11, page 6)

Requirements:
Apple][Plus
One blank disk
Copy II Plus or...
MUFFIN from your *DOS 3.3 System Master*

After several sojourns into the semi-sleazy world of Sierra On-Line's *SoftPorn Adventure*, I attempted to SAVE a game in progress, but was only rewarded with an *I/O ERROR*. I then decided that this disk needed to be moved to a normal DOS. The steps are quite simple and illustrate some of the basic steps for de-protecting many other programs.

First of all, this program is bootable under either DOS 3.2 or DOS 3.3. For many disks like this, all it takes to break the basic protection scheme is to *MUFFIN* the file over to a normal DOS 3.3 disk. On *SoftPorn Adventure*, however, there are some other fairly simple protection schemes which also have to be removed.

Conversion to Normal DOS

You will need a formatted DOS 3.3 disk to copy the *SoftPorn* files onto so you should INIT a blank disk if you don't already have one at hand. Then, either BRUN MUFFIN from the *DOS 3.3 System Master* or the normal copy mode of *COPY II Plus*. If you use *COPY II Plus*, set the source disk to DOS 3.2 and the destination to DOS 3.3. Copy (or *MUFFIN*) all files except *HELLO* to the DOS 3.3 disk using the wildcard character (=).

Unusual File Names

An old method of protecting files on a disk was to include control characters within the name or the use of characters that couldn't be entered from the keyboard.

The files are: *CHAIN*, *HELLO*, */////*, *////*, __, ____, and _____.

Modern copy programs such as *Copy II Plus* have no trouble displaying titles like this or copying the programs. The Apple //e allows direct entry of these characters, but on an Apple][or Apple][Plus about the only way to get control characters and characters not on the keyboard is to do a CATALOG and then use ESC, I, and → to trace over the file names. Of course, if you have a disk-editor you can use it to change the file names directly on the directory sectors, although I would not recommend doing this to an original disk. The use of the wildcard character when transferring files with *FID* eliminates the any problems caused by file names.

I recommend that you do NOT rename the files on the *SoftPorn Adventure* disk. If you wish they can be easily LOADed and LISTed by CATALOGing and using ESC, I and →. If you change the names then you will also have to change the names inside the programs:

A **/** shows up in a program as *LOG* and **_** shows up as *SIN*.

In _____, lines 55Ø, 56Ø, and 58Ø refer to *////*. Lines 52Ø, 53Ø, and 54Ø refer to ____ and line 61Ø refers to __.

In program __, line 2Ø11 refers to *///*. Line 51Ø in the program *///* refers to __.

Nibble Count

SoftPorn's first line of defense actually isn't its DOS, but the use of a nibble-count. Checking through the Hello program, I found that its only real function was to perform the nibble-count. Therefore, don't bother copying it or delete it after transfer. On your new disk:

```
RENAME ________, HELLO
LOAD HELLO
```

Now delete line 347 which again checks the nibble-count:

```
347
SAVE HELLO
```

The Reset Vector

The last step in the deprotection of this program is the disabling of the 'on reset, boot...' command. To do this:

```
LOAD HELLO
LISTØ,1
```

Delete the **POKE 1011,0** and the **POKE 1012,0** commands. These POKEs set up the reset vector so that it will boot when RESET is pressed.

```
SAVE HELLO
```

You have a fully broken *SoftPorn Adventure*.

Advanced Pornography Technique?

The deprotected *SoftPorn Adventure* disk works just like the original, except that you can now save a game in progress, even onto the game disk itself. Just ignore the prompts to switch disks.

You may also list the BASIC programs by LOADing them using the → to retype.

You can also read the Text files using *Copy II Plus*'s "VIEW FILES/TEXT" option for some clues for playing the game.

You can even RESET from within a game, give yourself a million dollars (type **M = 10000** RETURN), then type **CONT** RETURN to get back into the game.

One final clue for the game. As the serpent told Eve: "One man's garbage is another man's gain."

Recapping, here are the necessary steps for making a backup of *SoftPorn Adventure*.

1 Use *MUFFIN* from the DOS 3.3 Master disk to transfer all the files except *HELLO* from the original *SoftPorn Adventure* over to a formatted DOS 3.3 disk. Use the wildcard character (=), with prompting, when making the transfer:

```
BRUN MUFFIN
```

Alternatively, the normal file transfer utility from *Copy II Plus* can be used if the original DOS is set to 3.2 and the copy DOS is set to 3.3.

2 On the copied disk:

```
RENAME _ _ _ _ _,HELLO
```

If you have an Apple][or Apple][Plus you will need *Copy II Plus* or a disk-editor to change the file name. CATALOGing the disk and then using ESC, I, and ↓ will also work.

3 Removes the nibble count check from the *HELLO* program:

```
LOAD HELLO
347
0 POKE 34,0: POKE 35,24: CLEAR
SAVE HELLO.ECD
```

Stickybear series

Xerox Educational Software

Stickybear Softkey
by Jerry Caldwell
(Hardcore COMPUTIST # 15, page 10)

Requirements:
Apple][Plus or equivalent
Blank disks
Disk Search Utility (*Inspector, ZAP*, etc.)
COPYA from the *DOS 3.3 System Master*
Copycard or Integer card (optional)

For those who have looked into the excellent *Stickybear* series, the disks can be read by a normal sector editor or copied by *COPYA* with the exception of one sector. The protection scheme is based solely upon the data residing on this one sector. On the different programs in the series, the location of this protection will vary, but it is generally sector $ØF on track $Ø1 or $Ø2. This sector is checked at various intervals to validate the presence of the original *Stickybear* disk and, as many of you probably know, is very hard to duplicate even with the latest bit-copy programs.

Analysis of the booting process reveals that various tracks are loaded into memory, and then a jump is taken to a sequence of machine language instructions that are nearly identical on several of the *Stickybear* disks. In this series of instructions, an IOB (Input/Output Block, see pages 94-98 of the *DOS 3.3 Manual* or Chapter 6 of *Beneath Apple DOS*) is created for reading the protected sector. Next, a subroutine which alters the RWTS 'post-nibblization' routine (used to convert the 6-x-2-encoded nibbles read from the disk into 8-bit bytes) at $B8C2 is called. This change to the post-nibblization routine makes the first instruction of it a jump to a different post-nibblization routine located on hi-res page 2 (at $4E9F on *Stickybear Bop*). When the RWTS is called, the protected sector is read into memory. To restore the post-nibble routine to its original form, a second call is made to the routine which changed it in the first place.

The technique for producing a backup is as follows:

Using a modified *COPYA*, the original disk will be copied, ignoring the the read-error on the protected sector.

The *COPYA*-ed backup will then be searched with a disk-search utility for the code which reads the protected sector. Once this code has been found, the memory where the protected data is read into must be identified by examining the code which sets up the IOB.

The original *Stickybear* disk is then booted up, allowed to read the protected sector into memory and the program is then halted by an old Monitor RESET, an NMI card, or by modifying one of the copies of the disk. This translated data can then be recovered and written back to the proper sector on the *COPYA*-ed disk with a sector-editor.

Finally, because this sector on the backup will no longer be protected, the routine which alters the post-nibblization routine must be disabled.

Modifying COPYA

As pointed out in the documentation *for Bag of Tricks*, *COPYA* can be modified to permit disks with I/O errors to be copied (except for the sectors which are unreadable). This can be done by changing byte $E1 of file *COPY.OBJ0* to an $EA (NOP instruction). The procedure to do this is as follows:

First, insert the disk with *COPYA* on it and load the *COPY.OBJ* file:

```
BLOAD COPY.OBJ
```

Alter *COPY.OBJ* so that it will ignore unreadable sectors:

```
POKE 929 ,234
```

And resave it on another disk:

```
BSAVE COPY.OBJ0,A$2C0,L$10B
```

Note: *BSAVE this to another disk which contains the Applesoft program COPYA*, unless you want to have the modified version on your System Master.

Now, run the modified *COPYA* and, when prompted, copy the original disk to a blank. You should hear the drive recalibrate twice when it comes across the protected sector. If you do not have any means of resetting into the Monitor, make two copies of the *Stickybear* disk at this time.

The backups made with the altered *COPYA* will be **almost** identical to the original now, except that they will not work. The sector that is protected on the *Stickybear* original will be formatted, but will contain no data. We will rectify this situation shortly.

The next step in creating the backup is to locate the sector which contains the instructions for setting up the IOB to read the protected sector. These instructions are as follows:

```
A9 tt         LDA #$tt      Track with protected sector
8D EC B7      STA $B7EC     Store it in the IOB
A9 ss         LDA #$ss      Protected sector
8D ED B7      STA $B7ED     Put it in the IOB, too
```

You will need a disk-search utility like *ZAP* or *Inspector* to search the backup for the instruction STA $B7EC (8D EC B7). This code could be located almost anywhere on the disk, but tracks $1, $2 and $11 are likely candidates. Once you find this code, disassemble the sector and look for some more code which has the form:

```
A9 ll         LDA #$ll      Low byte of data buffer
8D FØ B7      STA $B7FØ     Store it in the IOB
A9 hh         LDA #$hh      High byte of the data buffer
8D F1 B7      STA $B7F1     Store it in the IOB
```

This set of instructions will indicate to you the location in memory that the protected sector will be read into. For example, if the code you find reads:

```
A9 ØØ         LDA #$ØØ
8D FØ B7      STA $B7FØ
A9 Ø3         LDA #$Ø3
8D F1 B7      STA $B7F1
```

then the protected sector will be read into memory starting at address $3ØØ. However, different *Stickybear* programs will store the data at different locations. Make a note of whatever address you decide that the protected sector is being loaded into.

Next, disassemble a little bit further and look for the next three JSR's. (Note: On some programs in the *Stickybear* series, the three JSR's will be on a sector adjacent to the one where the code which sets up the IOB is found). You should find a JSR $B7B5 which is sandwiched between two other JSR's to the same address. The call to $B7B5 is the main entry point to RWTS and the other two JSR's call the routine which alters the post-nibblization routine. The first call to the routine will alters the post-nibble routine before reading the protected sector and the second call to the routine restores it back to its normal form.

Just past the second call to the table-alteration routine you should see the instructions:

```
6Ø            RTS
A2 ØØ         LDX #$ØØ
```

The RTS marks the end of the routine which reads the protected sector and the LDX #$ØØ is the first instruction of the code responsible for altering the post-nibble routine. Later, we will negate this subroutine by storing a $6Ø in the place of the $A2 so that no alteration of the post-nibble routine will occur when it is called.

Recovering the Data

The data on the protected sector can be recovered in one of two ways. The first method requires an old Monitor F- ROM or NMI card so that the running *Stickybear* program can be interrupted. If you do not have any means of halting the program, then you will have to use the second, brute force method. I'll describe the easy method first.

The Easy Way

The RESET method of recovering the data is very straightforward. First, the original *Stickybear* program is booted up and allowed to read the protected sector. The program is then halted and the portion of memory that the data was read into is moved to a 'safe' location in memory before booting up a DOS 3.3 slave disk. The translated data from the protected sector can then be saved onto disk for later use. For instance, if you discover that the protected sector is being read into $300, boot up the original *Stickybear* disk, halt the program after it has started running and from the monitor type:

```
9000<300.3FFM
C600G (After inserting a DOS 3.3 slave disk)
BSAVE PROTECTED SECTOR,A$9000,L$100
```

The data from this sector will later be written to a backup copy of the *Stickybear* disk.

The Hard Way

Lacking an Integer or NMI card, we must alter one of the backups we have made to allow us to boot the backup, wait for us to insert the original, read the encoded sector and then exit to the monitor.

Earlier in the article the 'JSR sandwich' responsible for altering the post-nibblization-routine and reading the protected sector was described. What we will do is alter the first and third JSR's. The first JSR will be altered so that it calls a routine which we will write. This routine will wait for a keypress (while we insert the original disk) before calling the post-nibble-alteration routine. The second JSR will be changed to a JMP $FF65 so that the Apple's Monitor will be entered and the data from the protected sector can be recovered.

Due to of the number of different *Stickybear* programs, the location on the disk that these alterations have to be made will vary. As an example, I will show you how to make the changes to *Stickybear Bop*. For other programs in the series, the changes will be pretty much the same, except that they will have to be made to different sectors on the disk.

On *Stickybear Bop*, the code which reads the protected sector starts on track $11, sector $Ø2 and ends on track $11, sector $Ø1. A disassembly of this routine is printed below:

```
4DD8- A9 Ø2      LDA #$Ø2
4DDA- 8D EC B7   STA $B7EC
4DDD- A9 ØF      LDA #$ØF
4DDF- 8D ED B7   STA $B7ED
4DE2- A9 ØØ      LDA #$ØØ
4DE4- 8D FØ B7   STA $B7FØ
4DE7- A9 BØ      LDA #$BØ
4DE9- 8D F1 B7   STA $B7F1
4DEC- A9 ØØ      LDA #$ØØ
4DEE- 8D EB B7   STA $B7EB
4DF1- A9 Ø1      LDA #$Ø1
4DF3- 8D EA B7   STA $B7EA
4DF6- A9 B7      LDA #$B7
4DF8- AØ E8      LDY #$E8
4DFA- A2 ØØ      LDX #$ØØ
4DFC- 48         PHA
4DFD- 8A         TXA
4DFE- 48         PHA
4DFF- 98         TYA
4EØØ- 48         PHA
4EØ1- 2Ø 12 4E   JSR $4E12
4EØ4- 68         PLA
4EØ5- A8         TAY
4EØ6- 68         PLA
4EØ7- AA         TAX
4EØ8- 68         PLA
4EØ9- 2Ø B5 B7   JSR $B7B5
4EØC- Ø8         PHP
4EØD- 2Ø 12 4E   JSR $4E12
4E1Ø- 28         PLP
4E11- 6Ø         RTS
4E12- A2 ØØ      LDX #$ØØ
4E14- AØ ØØ      LDY #$ØØ
4E16- BD 3B 4E   LDA $4E3B,X
4E19- E8         INX
4E1A- 85 68      STA $68
4E1C- BD 3B 4E   LDA $4E3B,X
4E1F- E8         INX
4E2Ø- 85 69      STA $69
4E22- C5 68      CMP $68
4E24- DØ Ø5      BNE $4E2B
4E26- C9 ØØ      CMP #$ØØ
4E28- DØ Ø1      BNE $4E2B
4E2A- 6Ø         RTS
4E2B- BD 3B 4E   LDA $4E3B,X
4E2E- 48         PHA
4E2F- B1 68      LDA ($68),Y
4E31- 9D 3B 4E   STA $4E3B,X
4E34- E8         INX
4E35- 68         PLA
4E36- 91 68      STA ($68),Y
4E38- 4C 16 4E   JMP $4E16
4E3B- C2         ???
4E3C- B8         CLV
4E3D- 4C C3 B8   JMP $B8C3
4E4Ø- 9F         ???
4E41- C4 B8      CPY $B8
4E43- 4E DC B8   LSR $B8DC
4E46- 4C DD B8   JMP $B8DD
4E49- 4F         ???
4E4A- DE B8 4E   DEC $4EB8,X
4E4D- ØØ         BRK
```

As a result of examining this code you should be able to see that sector $F on track $Ø2 is the protected sector and that it is read into $BØØØ—$BØFF.

The JSR $4E12 instructions are the calls to the post-nibblization-alteration routine. The first of these calls will be changed to a JSR $BCDF. At $BCDF, which is the beginning of 33 free bytes within the RWTS, we will put the routine which waits for the keypress before reading the protected sector. This routine will look like this:

```
2Ø ØC FD    JSR $FDØC    Wait for a keypress
2Ø 12 4E    JSR $4E12    Alter Postnibble
6Ø          RTS          Return
```

The second JSR $4E12 will be changed to JSR $FF65 for entry into the Apple's Monitor. So, on *Stickybear Bop* make the following sector edits to one of your backup copies:

Track	Sector	Byte	From	To
00	06	DF	88	20
00	06	E0	A5	0C
00	06	E1	E8	FD
00	06	E2	91	20
00	06	E3	A0	12
00	06	E4	94	4E
00	06	E5	88	60
11	01	02	12	DF
11	01	03	4E	BC
11	01	0E	12	65
11	01	0F	4E	FF

After making the changes, boot this disk. The patch at $BCDF will be executed momentarily. The drive will continue to spin, but all movement of the head will stop because the code is waiting for a key to be pressed. At this time, remove the modified backup, insert your write-protected original disk and press any key. When you hear a beep, the protected sector will have been read into memory. Depending upon the current setting of the screen soft-switches, you may or may not be able to see what you type at the keyboard. But fear not. Your Apple will accept input just the same. So, place a DOS 3.3 slave disk in the drive and recover the data from the protected sector by **carefully** typing:

```
9000<B000.B0FFM
C600G
BSAVE PROTECTED SECTOR,A$9000,L$100
```

Remember, *Stickybear Bop* was used as an example here and you will have to modify the procedure depending upon which disk in the series you are using.

Once you have saved the data from the protected sector on disk, all you have to do is write this data back to the appropriate sector on the other COPYAed disk of the *Stickybear* disk and change the first byte of the routine that alters the post-nibble routine from a $A2 (LDX) to a $60 (RTS). A generalized step-by-step procedure for making backups of the *Stickybear* series follows below.

Stickybears in General

1 Make two copies of the *Stickybear* disk using the modified *COPYA* program which was described earlier.

2 Use a disk search utility, such as *ZAP* or *Inspector* to search the disk for the code which sets up the IOB for reading the protected sector. Search for a hex pattern of **8D EC B7**.
Note: On *Stickybear Bop*, it will be found on track $11, sector $2.

3 Examine the code on the sector where this pattern was found and identify the portion of memory that the protected sector is read into.
Note: On *Stickybear Bop*, it is read into $BØØØ—$BØFF from track $Ø2, sector $F.

4 Recover the data from the protected sector by using either the 'Easy Way' or the 'Hard Way' and save it to disk.

5 Write the data from the protected sector to the appropriate sector on the backup with a sector-editor.
Note: On *Stickybear Bop*, it should be written to track $Ø2, sector $F.

6 Change the first byte of the routine which alters the post-nibble-routine from a $A2 to a $6Ø and write it back to the disk.
Note: On *Stickybear Bop*, this is byte $12 on track $11, sector $1.

At this point, you should (hopefully) have a completely *COPYA*-able version of your *Stickybear* disk that boots and runs exactly likes the original. Congratulations, if you were successful. If your backup does not boot, then carefully go back over the procedure and make sure that you did not forget to do anything along the way.

Although I have not had an opportunity to test this procedure on anything but *Stickybear disks*, I would not be surprised if it also works on other programs put out by Xerox.

Albert Snopes'

More Stickybear softkeys

for: ***Fat City***
Basket Bounce
Numbers Or Shapes

1 Use the modified *COPYA* presented in the previous article.

2 Use the *TRACER* from *CIA* files to find the sector where the IOB is set up. Then search for a pattern of **8D EC B7** and determine the location on the disk of the protected sector and its destination in memory.

3 RESET into the Monitor after booting the game. Move the data from the protected sector to $9ØØØ where it was safe from a reboot. Booted a slave disk and BSAVED PROTECTED SECTOR, A$19ØØØ,L$1ØØ.

4 Use the *Inspector* and *Watson* to write the recovered data back to the *COPYA*-ed disk. Perform the sector edit that prevents the program from altering its RWTS.

I have found that this technique will work on four of the disks in the series without any alterations. Here is the data you will need:

Sector which reads protected sector: **track $02, sector $06**

Protected sector: **track $01, sector $0F**

Destination of protected sector: **$1F00**

Sector edit: **track $02, sector $06**, from **$3A** to **$60**

Randy Ramirez's

Still More Stickybear info

If you're a beginner and a little shaky about writing the protected sector back to the *COPYA* backup, here's another way to do it:

1 Instead of moving $300—$3FF to $9000, move it to 6000 (it's just a nice number to tinker with...):

```
6000<300.3FFM C600G
```

2 Boot a DOS 3.3 diskette without a Hello program.

3 Remember: You copied $300—$3FF into $6000—$60FF. Now, instead of BSAVEing the protected sector, lets just write it directly to the backup disk that you made, since it's already in memory. This short write-routine will do the job:

```
CALL -151
300:20 E3 03 sets up DOS' RWTS routine
303:4C D9 03
B7EB: 00 02 0F set track ($02) and sector ($0F)
B7F0:00 60 buffer (storage) in lo-hi order ($6000)
B7F2: 00 00 02 02 tells RWTS to write a track & sector
```

4 Now you're ready to write. Insert the backup you've made of the *Stickybear* and from the Monitor type:

```
300G
```

You have a perfectly COPYAable *Stickybear BOP* backup.

I used this technique of writing a sector because when you BSAVE the protected sector, it creates a third sector which makes it impossible to write into a single sector without a sector editor. The third sector is due to a binary files address and length information.

This technique writes a sector to disk as long as you specify the buffer address and the track and sector, of course.

Suicide

Piccadilly software

Deprotecting Suicide
by Clay Harrell
(Hardcore COMPUTIST # 12, page 6)

Requirements:
Apple][or Apple][Plus with old-style F8 Monitor ROM
A blank initialized DOS 3.3 disk

Suicide is a rather gruesome little arcade game in which you attempt to save some creatures from killing themselves on the sidewalk. The game has a few twists too: you should not save mutant creatures, but let them fall to their deaths. Although this game has been around for quite awhile, the technique used to remove its protection is applicable to a great deal of other protected software.

Piccadilly uses a lot of disk protection to keep you from making a nibble-copy of their disk. The primary protection is the use of half-tracks. Upon booting the disk, the disk drive head seeks track 32 and then proceeds to read in the program alternating between whole and half-tracks down to track 19. The program then JMP's to $1000 to start the game.

The first step in removing the copy-protection from a disk like this is to find what range of memory it occupies. The best way to do this is to use the Monitor memory-move command to fill most of memory with 00's, load in the game, and then to RESET into the Monitor what has been loaded over by the program. To fill memory from $800—$9600 with 00's, from the Monitor type:

```
800:00 N 801<800.95FFM
```

After this is done, boot the game and RESET into the Monitor and see what memory is used.

For *Suicide*, you should find that memory from $800 to $5FFF is used by the game. After booting the game and RESETing into Monitor, typing:

```
7000<800.A00M
```

will clear the way for a slave disk boot. We can now BSAVE *Suicide* to a DOS 3.3 slave disk.

If you try 1000G, you will find that the game starts up and everything is fine. When you RESET into the Monitor you will see some screen garbage over text page one but fear not. This is the program loader and not part of the game. It does not need to be saved.

With *Suicide*, it is important to press RESET at the right time. Make sure you press RESET right when the red drive light goes out (when the initial title page is showing). Stopping the program here serves two purposes. First, when you save the game you will get the nice title page, and second, it stops the program before it initializes the lower pages of memory.

So here's the method to deprotecting *Suicide*:

1 Boot *Suicide* and, just after the drive motor stops (and when the initial title page is showing), RESET into the Monitor.

2 Next, move pages $08 and $09 where they will be safe during the boot of a slave disk;

```
7000<800.9FFM
```

3 Boot a 48K slave disk with a short or null *HELLO* greeting program:

```
PR#6
```

4 Enter the Monitor and move pages $08 and $09 back to their original locations:

```
CALL -151
800<7000.71FFM
```

5 Next, enter some code which displays hi-res page 2 before the game starts:

```
FF4:AD 50 C0 AD 54 C0 AD 57 C0 AD 52 C0
```

6 Install a patch to the above code at $FF4:

```
7FD:4C F4 0F
```

7 Finally, BSAVE the game to a DOS 3.3 disk by typing:

```
BSAVE SUICIDE,A$7FD,L$5803
```

TellStar

Information Unlimited Software

Softkey For TellStar
by William Wingfield, Jr.
(Hardcore COMPUTIST # 13, page 8)

Requirements:
Apple][Plus or Apple //e
Inspector/Watson
MUFFIN from the *DOS 3.3 System Master*
Initialized DOS 3.3 disk

TellStar is an absolutely amazing program. It is great to have around for the people that look at the Apple and say, "So what does it do other than play games?" I have yet to see anyone who hasn't been impressed by *TellStar*'s information of the night sky. The program does have one slight problem, though: copy-protection. That problem can be remedied quite easily so that the program can be viewed and reduced in size from a whole disk to 326 sectors.

There are two parts to the protection scheme for this program. The first is an altered DOS 3.2 format. The second is a phony catalog on track $11; the real catalog is on $15. It's good to know that someone puts those Beagle Brothers tip books to use. We'll use *MUFFIN* as our DOS 3.2 code source and to transfer the files when we are ready.

The first thing to do is to fix the catalog so that *MUFFIN* can find the files. We'll be writing to the *TellStar* disk, so use a backup to be on the safe side.

1 Boot the Master disk, install *Inspector* (or *Watson*) and load *MUFFIN*:

BLOAD MUFFIN

2 Now we'll fix the 3.2 code to match that used by *TellStar*. You can make the changes from *Inspector* or from the Monitor. The last byte of the Address-marker changes from track to track so we'll make DOS ignore it:

```
1A8A:29 00
```

3 The last byte of the End-of-Data-marker has been changed to $D5 so we'll match that next.

```
1A60:D5
```

4 Since we need to write to the disk, we must also match the bytes in the section of DOS that does the writing. The last byte of the Address-marker on track $11 is $D6 (you may wish to verify the markers on your disk in the event that they aren't all the same) so we'll match it:

```
1FFF:D6
19E3:D5
```

5 Now to fix the CATALOG. Enter *Inspector* if you haven't already done so. *Inspector* uses $3DA—$3DB as a pointer for the RWTS it uses when accessing a disk. The RWTS for *MUFFIN* that we just altered is at $1E00 so edit these bytes to $00 and $1E respectively. Next, go to a clean buffer such as $50 and read in track $11, sector $C. Bytes $2 and $3 show $11 and $0C, the fake catalog. Edit these to $15 and $09, the real catalog location, and write the sector back to the disk. Return the pointer on page $3 to $B7B5 and exit *Inspector*.

6 Now to unlock the disk. Start up *MUFFIN*:

```
CALL 2051
```

7 Use = for *FILE NAME* and transfer all the files to the 3.3 disk. You can delete the file called *REAL CAT* (it just reserved space for the CATALOG on the protected disk). *TELLSTAR I* is the greeting program.

8 Exit from *MUFFIN* and load the greeting program:

```
LOAD TELLSTAR I
```

9 Eliminate the POKE 214 ,255 from line 100.

Now if you RESET from within the program, it won't try to run itself every time you enter a command. You're finished! Grab a drink and enjoy the feeling of having beaten **The Protectors** once again.

Tic Tac Show

Computer Advanced Idea

Boot Code Trace For Tic Tac Show
by Steve Fillipi
(Hardcore COMPUTIST # 15, page 23)

Requirements:
Apple][Plus or equivalent
Bit-copier or *Super IOB*
One blank disk

Tic Tac Show is a computerized rendition of the popular TV game show that bears a very similar name. .The game can be played by 1 or 2 players and is hosted by the lovely young lady, Carol. The players try to place their X's or O's on a Tic-tac-toe board by correctly answering questions from a particular subject category. The first player to get 3 adjacent squares on the board is declared the winner. The game is very nicely done and youngsters of all ages should enjoy playing it. Because the program allows you to create your own categories and questions, the game can be tailored for specific educational settings. Unfortunately, the potential of this program in an educational environment is somewhat limited by its copy-protection.

With a little investigation, I found the protection scheme on *Tic Tac Show* to be as follows:

1. The Address-epilogues and Data-epilogues on track $Ø have been modified. The Address-epilogues are changed to B5 AB EB and the Data-epilogues are AB AA EB.

2. The *Tic Tac Show* DOS resides on tracks $1.5 through $4.5.

3. Tracks $6—$22 are normally formatted, but 2 directories are present, one on track $11 and one on track $Ø6. The files in the directory on track $11 contain the questions and answers, and the files in the directory on track $6 hold the code for the game itself. The DOS utilizes the two directories by changing the value at $ACØ1 from a $11 to a $Ø6, or vice-versa.

From this description of the protection scheme, it appears that it would not be too difficult to backup this disk with a bit-copier by copying tracks 0, 1.5-4.5 and 6-22. Surprisingly, I could not get *Copy II Plus v4.4c* or *Essential Data Duplicator III* to make a bootable copy of *Tic Tac Show*. Instead of fooling around with parameter settings and disk drive speed adjustments, I decided to try and boot-code-trace the disk. For me, this proved to be a more successful and educational approach than using a bit copier. My report follows.

Boot Code Theory

Although the boot-code-tracing technique has been covered several times before in the pages of this magazine, I will quickly go over a little theory for the benefit of new readers.

The basis for boot-code-tracing lies in the fact that on any bootable disk, whether protected or not, track $0, sector $0 must be readable by the disk controller hardware. The first thing that happens when a disk boots is that track $0, sector $0 is read into memory at $800—$8FF. After this has occurred, the code in the disk controller's ROM jumps to $801 and the code there reads track $0 into memory (on normal disks). Protected disks may vary. The boot will continue from here, reading whatever tracks are necessary, until it is complete.

To boot-code-trace a disk, it is necessary to halt the boot after track $0, sector $0 has been read into memory, but before the code at $801 starts to excute. This can be done by moving the code from the disk-controller ROM down into RAM where it can be modified. Instead of jumping to $801 to continue the boot, it will now enter the Apple's Monitor. The code read in from the disk can then be examined and possibly modified so that another stage of the boot will take place before the Monitor is again entered. By stopping at various stages of the boot, the entire program can eventually be read into memory and transferred to a normal disk. The difficulty of boot-code-tracing copy-protected disks varies greatly, but whatever the difficulty, a good knowledge of 6502 machine language is an absolute necessity for trying this technique on your own.

Tic Tac Boot

When performing a boot-code-trace, first disconnect DOS and fill up memory from $800—$BFFF with $11's so we can tell where code gets loaded into. Do this by typing:

```
CALL -151
FE89G N FE93G
800:11 N 801<800.BFFFM
```

Next, move the code from the disk-controller ROM down into RAM where it can be modified:

```
9600<C600.C6FFM
96F8:A9 00 85 FC 85 FD 85 FE
9700:A9 60 85 FF A0 43 B1 FC
9710:91 FE 88 10 F9 AD E8 C0
9718:4C 69 FF
```

If you disassemble from $96F8, you should see the following code:

```
96F8- A9 00     LDA #$00          9706- B1 FC     LDA ($FC),Y
96FA- 85 FC     STA $FC           9708- 91 FE     STA ($FE),Y
96FC- 85 FD     STA $FD           970A- 88        DEY
96FE- 85 FE     STA $FE           970B- 10 F9     BPL $9706
9700- A9 60     LDA #$60          970D- AD E8 C0  LDA $C0E8
9702- 85 FF     STA $FF           9710- 4C 69 FF  JMP $FF69
9704- A0 43     LDY #$43
```

This code will save some necessary zero page locations to page $60 before turning OFF the drive motor and entering the Monitor.

Insert the original *Tic Tac Show* disk into the drive and boot it:

```
9600G
```

When the drive turns OFF, the *BOOT1* code will have been read into page $08. Examine and compare this code to *BOOT1* from a DOS 3.3 slave disk. They are identical except for the values at $84D—$85C (sector-skewing-table) and $8FE—$8FF. The *Tic Tac Show* boot reads the sectors from track $0 in physically ascending order rather than interleaving them as DOS normally does. This, in part, accounts for the slow boot of the disk. On a DOS 3.3 slave disk, the values at $8FE—$8FF are B6 09 which indicates that the data will be read into memory starting at $B600 and that 10 sectors (0—9) will be read in this stage of the boot. On the *Tic Tac Show* disk, these values are $3F $05. This means that 6 sectors of data will be read into memory starting at $3F00.

The *BOOT1* code exits at $84A with an indirect jump to the address stored in $8FD—$8FE. On a normal slave disk this will be $B700 but, on *Tic Tac Show*, it is $4000.

We now want to execute the *BOOT1* code and let it read the 6 sectors of data from track $0 into memory. Typing 801G RETURN would prove to be unsuccessful because the code at $801 expects the disk drive to be revolving and needs some data from zero page that was lost when the Monitor was entered. The necessary code from zero page was stored on page $60, so we must enter some code which will turn on the disk drive, wait for the drive to come up to speed (about 1 second) and restore locations $0—$43 from page $60 before jumping to $801. This code can be placed at $900:

```
900:8D E9 C0 A0 09 A9 C0 20
908:A8 FC 88 D0 F8 A0 43 B1
910:FE 91 FC 88 10 F9 4C 01
918:08
```

This should disassemble as:

```
0900-  8D E9 C0  STA $C0E9          090D-  A0 43     LDY #$43
0903-  A0 09     LDY #$09           090F-  B1 FE     LDA ($FE),Y
0905-  A9 C0     LDA #$C0           0911-  91 FC     STA ($FC),Y
0907-  20 A8 FC  JSR $FCA8          0913-  88        DEY
090A-  88        DEY                0914-  10 F9     BPL $090F
090B-  D0 F8     BNE $0905          0916-  4C 01 08  JMP $0801
```

This code uses the Monitor WAIT routine at $FCA8 to create the delay that pauses while the disk drive reaches its proper speed. Before you can execute this code we will also have to modify the instruction at $84A so that it jumps to the code which turns OFF the disk instead of continuing with the boot:

84A:4C 0D 97

and then execute the next stage of the boot:

900G

When the drive shuts off, the code responsible for accessing the half-tracks will have been placed at $4000—$44FF (track $0, sector $0 is re-read into $3F00—$3FFF). The start of this code should disassemble as:

```
4000-  86 2B     STX $2B            400B-  A9 00     LDA #$00
4002-  A2 FF     LDX #$FF           400D-  85 06     STA $06
4004-  9A        TXS                400F-  85 07     STA $07
4005-  2C 80 C0  BIT $C080          4011-  85 08     STA $08
4008-  2C 81 C0  BIT $C081          4013-  8D 78 04  STA $0478
```

Notice that the first thing this code does is to store the X-register into $2B, which is where the slot number of the drive controller is held (actually 16 * slot number). We have to make sure that, when we execute the next stage of the boot at $4000, the X-register contains a #$60.

If you disassemble from $40D8, you should find:

```
40D8-  A9 06     LDA #$06           40EF-  4A        LSR
40DA-  8D 01 AC  STA $AC01          40F0-  4A        LSR
40DD-  A9 01     LDA #$01           40F1-  AA        TAX
40DF-  8D EA B7  STA $B7EA          40F2-  A9 00     LDA #$00
40E2-  8D F8 B7  STA $B7F8          40F4-  9D 78 04  STA $0478,X
40E5-  A5 2B     LDA $2B            40F7-  9D F8 04  STA $04F8,X
40E7-  8D E9 B7  STA $B7E9          40FA-  20 93 FE  JSR $FE93
40EA-  8D F7 B7  STA $B7F7          40FD-  20 89 FE  JSR $FE89
40ED-  4A        LSR                4100-  4C 84 9D  JMP $9D84
40EE-  4A        LSR
```

By the time this code executes, the *Tic Tac Show* DOS is in place, but not yet initialized. A value of #$06 is stored at $AC01 to set up the access of the directory on track $06, and the code goes on to set up some memory locations used by DOS and does the equivalent of an IN#0:PR#0. At $4100, the JMP $9D84 is the entry to the DOS cold-start routine. When this routine is executed, the DOS will be initialized and the disk's greeting file *MENU/IBC* will be loaded in and take over. We don't want this to happen. Instead, change the instruction at $4100 so that it jumps to the Monitor:

```
4101:69 FF
```

The code that we placed at $900 for turning ON the drive and setting up zero page is still intact and is again needed for executing the next stage of the boot. It has to be changed so that the X-register will hold a #$60 and the jump will be to $4000 instead of $801:

```
916:A2 60 4C 00 40
```

Then continue the boot with:

```
900G
```

When the drive shuts off, all the code we need to save will be in memory. I found the necessary code to be at $800—$1FFF and $8C00—$95FE, in addition to the DOS at $9600—$BFFF. To recover this code, it will have to be moved to a location in memory that is not overwritten by the boot of a slave disk ($900—$95FE is safe). To do this, perform the following memory moves:

```
6000<800.8FFM
2100<8C00.BFFFM
```

and then boot a DOS 3.3 slave disk:

```
C600G
```

Enter the Monitor and move page $08 back into place:

```
CALL -151
800<6000.60FFM
```

and then enter some code at $20DE that will put $8C00—$BFFF back into place before jumping to the DOS cold-start routine:

```
20DE:20 89 FE
20E1:20 93 FE A0 00 84 3C 84
20E9:42 88 84 3E A9 21 85 3D
20F1:A9 54 85 3F A9 8C C8 20
20F9:2C FE 4C 84 9D
```

This code should disassemble as:

```
20DE-  20 89 FE   JSR $FE89
20E1-  20 93 FE   JSR $FE93
20E4-  A0 00      LDY #$00
20E6-  84 3C      STY $3C
20E8-  84 42      STY $42
20EA-  88         DEY
20EB-  84 3E      STY $3E
20ED-  A9 21      LDA #$21
20EF-  85 3D      STA $3D
20F1-  A9 54      LDA #$54
20F3-  85 3F      STA $3F
20F5-  A9 3C      LDA #$3C
20F7-  85 43      STA $43
20F9-  C8         INY
20FA-  20 2C FE   JSR $FE2C
20FD-  4C 84 9D   JMP $9D84
```

Checked the new code, then save $800—$54FF to disk by typing:

```
BSAVE TIC TAC SHOW,A$800,L$4D13
```

About the only thing left to do is to copy tracks $6—$22 of the original *Tic Tac Show* disk to an initialized disk (preferably with a fast DOS) and create a *HELLO* program for the disk. The necessary tracks can be copied either with a bit copier or by *Super IOB*, and the standard controller with the variable TK in line 1010 changed from 0 to 6. The Applesoft *HELLO* program must relocate itself to a free area of memory ($800—$54FF will be occupied). Therefore, BLOAD in the file *TIC TAC SHOW* and then do a CALL 8414 ($20DE). This *HELLO* program is listed near the end of the article.

Tic Tac Show is definitely not one of the more difficult disks to boot-code-trace. As such, this program would be a good beginner project for those wishing to learn the technique. The principles covered here can definitely be applied to disks with tougher forms of copy-protection if desired. The steps necessary for boot-code-tracing *Tic Tac Show* are recapped below.

A Recap

1 INITialize a blank disk, preferably with a fast DOS. The volume number of this disk must be **1**:

```
INIT HELLO,V1
```

2 Use a bit copier or *Super IOB* with a modified standard controller to copy tracks $6—$22 from the original *Tic Tac Show* disk to the disk initialized in Step 1.

3 Boot up DOS 3.3, enter the Monitor, move the code from the disk controller ROM down to page $96 and modify it to save some necessary zero page locations before turning OFF the drive motor and entering the Monitor at $FF69:

```
CALL -151
9600<C600.C6FFM
96F8:A0 43 A9 00 85 FC 85 FD
9700:85 FE A9 60 85 FF B1 FC
9708:91 FE 88 10 F9 8D E8 C0
9710:4C 69 FF
```

4 Boot the original *Tic Tac Show*; insert disk and:

```
9600G
```

5 After the drive turns OFF and the Monitor prompt reappears, make the *BOOT1* exit jump to the code at $970D, where the drive will be turned OFF and the Monitor is entered:

```
84A:4C 0D 97
```

6 At $900, place a routine which will turn on the drive, wait about one second, restore the necessary zero page locations and then jump to the *BOOT1* code at $801:

```
900:8D E9 C0 A0 09 A9 C0 20
908:A8 FC 88 D0 F8 A0 43 B1
910:FE 91 FC 88 10 F9 4C 01
918:08
```

7 Read the code necessary for accessing the half-tracks into memory by typing:

```
900G
```

8 The next stage of the boot starts at $4000 and exits at $4100, where a jump to the DOS cold-start-routine is taken. Modify this jump to enter the Monitor instead of performing the cold-start:

```
4101:69 FF
```

9 The code we entered at $900 is still necessary for turning on the drive; however, it must be modified a bit so that a jump to $4000 is taken with the X-register set to #$60 (for the slot #). Modify this code and then execute it:

```
916:A2 60
918:4C 00 40 N 900G
```

10 After the drive has been on for about thirty seconds, the Monitor prompt should reappear and the drive will turn itself OFF. All the code that has to be recovered is now in memory. Insert the DOS 3.3 slave disk that was initialized in Step 1 into the drive. Before booting it, move the code to 'safe' areas of memory:

```
2100<8C00.BFFFM
6000<800.8FFM
C600G
```

11 After the DOS 3.3 slave disk has been booted, enter the Monitor and move page $60 back down to page $08 by typing:

```
800<6000.60FFM
```

12 At $20DE, enter the routine which will move code at $2100—$54FF back to its proper location ($8C00—$BFFF) before jumping to the DOS coldstart routine:

```
20DE:20 89 FE
20E1:20 93 FE A0 00 84 3C 84
20E9:32 88 84 3E A9 21 85 3D
20F1:A9 54 85 3F A9 8C 85 43
20F9:C8 20 2C FE 4C 84 9D
```

13 Next, save memory from $800—$54FF to the disk by typing:

BSAVE TIC TAC,A$800,L$4D13

14 Finally, type in the following *HELLO* program and save it to the disk:

HELLO

```
10 ON PEEK (104 ) = 96 GOTO 20 : POKE 104 ,96 : POKE 24576 ,0 : PRINT
    CHR$ (4 ) "RUN^HELLO"
20 PRINT CHR$ (4 ) "BLOAD^TIC^TAC^SHOW,A$800"
30 CALL 8414
```

SAVE HELLO

The resulting disk is completely *COPYA*-able and, in addition, will boot quite a bit faster than the original did.

Carol awaits you.

Time is Money

Turning Point Software

Deprotecting Time Is Money
by Rod Wideman
(Hardcore COMPUTIST # 12, page 7)

Requirements:
48K Apple][Plus equivalent
COPYA from *DOS 3.3 System Master*
A blank disk
A sector-editing program

Copy *Time Is Money* easily using the following instructions:

1 Make a copy of *Time Is Money* with *COPYA* from the *DOS 3.3 System Master*:

RUN COPYA

2 Use a sector-editor to make the following modifications to the copy of *Time Is Money*:

Track	Sector	Byte	From	To
$02	$0F	$19	$BD	$60
$02	$0F	$74	$38	$18
$05	$0F	$19	$BD	$60
$05	$0F	$74	$38	$18

3 Don't forget to write the modified sectors back out to the disk.

Type Attack

Sirius Software

Backing Up Type Attack
by Jerry Caldwell
(Hardcore COMPUTIST # 12, page 8)

Requirements:
48K Apple][Plus or equivalent
A nibble-copy program
A sector-editor
One blank disk

Type Attack from Sirius is one of those rare pieces of software that is both educational and fun to use. Unfortunately, like so many home and educational programs, *Type Attack* is copy-protected to the extent that none of the bit-copiers I own would make a working backup copy. Not wanting to hand my original *Type Attack* disk over to my young children, I set out to find a way to make a copy. Happily, I was successful. My report follows.

I found that on the original *Type Attack* disk, only tracks $Ø—$1Ø have any useful data on them, although track $22 is used to verify the presence of the original program disk. I assumed that if I could find and circumvent the routine which checks track $22, I would have a working bit-copy of *Type Attack*.

The majority of the *Type Attack* disk is 4-x-4-encoded (rather than 6-x-2-encoded) and cannot be read by a normal sector-editor. This makes it very difficult to find the location on the disk that the verfication routine is called from. However, track $Ø, sector $Ø can be read by any sector-editor and I found that it was possible to make some modifications there to prevent the disk-verification-routine from being called. The changes I made on track $Ø, sector $Ø modified the instruction at $A68 so that the disk-verification-routine is bypassed and also changed the reset vector so that the disk will not do a total reboot if RESET is hit.

On an original copy of *Type Attack* the final instruction of the code on track $Ø, sector $Ø reads:

```
JMP $9FØ
```

I modified this code to read:

```
LDA #$4C
STA $A68
LDA #$6E
STA $A69
LDA #$ØA
STA $A6A
LDA #$ØØ
STA $AØ1
LDA #$4Ø
STA $AØF
JMP $9FØ
```

Luckily, there is enough free space on the sector to fit in the extra code. Thus, the entire procedure for making a backup of *Type Attack* involves copying tracks $Ø—$1Ø with a bit-copier followed by the sector-edit to track $Ø, sector $Ø of the copy.

Attack On Type Attack

1 Use a bit-copy program to copy tracks $Ø—$1Ø of *Type Attack*. Set the Address-header parameters to **AD DA DD** (for *COPY II Plus*: E = AD, F = DA, 1Ø = DD).

2 Use your sector-editor to make the following changes to track $Ø, sector $Ø.

Byte	From	To	Byte	From	To
$93	$4C	$A9	$A1	$ØØ	$ØA
$94	$FØ	$4C	$A2	$ØØ	$A9
$95	$Ø9	$8D	$A3	$ØØ	$ØØ
$96	$ØØ	$68	$A4	$ØØ	$8D
$97	$ØØ	$ØA	$A5	$ØØ	$Ø1
$98	$ØØ	$A9	$A6	$ØØ	$ØA
$99	$ØØ	$6E	$A7	$ØØ	$A9
$9A	$ØØ	$8D	$A8	$ØØ	$4Ø
$9B	$ØØ	$69	$A9	$ØØ	$8D
$9C	$ØØ	$ØA	$AA	$ØØ	$ØF
$9D	$ØØ	$A9	$AB	$ØØ	$ØA
$9E	$ØØ	$ØA	$AC	$ØØ	$4C
$9F	$ØØ	$8D	$AD	$ØØ	$FØ
$AØ	$ØØ	$6A	$AE	$ØØ	$Ø9

3 Write the sector back to the disk.

Ultima III: Exodus

Origin Systems

Softkey For EXODUS: Ultima III
by Tim Schaap
(Hardcore COMPUTIST # 11, page 27)

Requirements:
Apple with 48K
One disk drive with DOS 3.3
Super IOB
One blank disk

Exodus: Ultima III, by Origin Systems, is a superior role-playing game. The author, Lord British, has added many enhancements to this, the third, *Ultima* scenario. Unfortunately, the program side of this third *Ultima* still doesn't allow the user to back it up. On the brighter side, there exists a method of unprotecting *Exodus*. Some boot code tracing is required to capture its RWTS, but once this has been done, *Super IOB*, with the proper controller installed, can be used to make a backup of *Exodus*.

Several things prevent making a duplicate of *Exodus* with a bit copier. The address and data marks on the disk are changed extensively throughout the disk. The only tracks that are used on the disk are $Ø—$1Ø, the rest of the tracks being unformatted. The disk is similar to normal DOS 3.3 in that it uses normal DOS 3.3 RWTS calls and an Input/Output Block (IOB). Even though the RWTS and IOB are at different locations than in normal DOS 3.3, *Exodus* is a prime target for deprotecting with *Super IOB*.

But How Do I Do It?

To make a backup of your original *Exodus* disk, first we have to capture the entire *Exodus* RWTS so that it can later be put into memory for utilization by *Super IOB*. In order to do this a little boot-code-tracing is required. Boot-code-tracing is not a process used for manufacturing footware, but is a technique for gradually loading pieces of code into memory from disk and halting the code before it can begin to execute. This method is based upon the fact that, even on highly protected Apple disks, track $Ø, sector $Ø, must be readable by the disk controller hardware.

1 Begin by entering the Monitor:

```
CALL -151
```

2 Move *BOOTØ* into RAM so we can control where it will go after reading in *BOOT1*:

```
86ØØ<C6ØØ.C6FFM
```

At this point, you may look at $86F8 and see that it jumps out to $Ø8Ø1 after reading in *BOOT1*. We want to modify this so it will jump to $88Ø1 instead. We make $86Ø1 jump to $FF59, the Monitor, so it will give us control after it has read in *BOOT1*.

3 Modify *BOOTØ* to jump to $88Ø1 and at $88Ø1 place a jump to the Monitor:

```
86FA:88
88Ø1:4C 59 FF
```

4 Everything is ready so we can start up our modified boot:

```
86ØØG
```

5 Stop the drive after it beeps:

```
CØE8
```

6 Move $Ø8ØØ (*BOOT1*) to $88ØØ so we can change how it works:

```
88ØØ<8ØØ.9ØØM
```

7 The next step changes $8811 to ORA with #$8Ø instead of #$CØ so it will set up the indirect jump to go to $865C (our modified-read routine, down in RAM), and not $C65C (the ROM-read routine). There is a branch out to $8846 that jumps out only after it has read in the necessary information. This step also makes $8846, the location where it branches out, jump to the Monitor in order to give us control after it has read in the RWTS.

```
8812:8Ø
8846:4C 59 FF
```

Note from Nathan Manlove: *On my version, a better choice for step 7 is $8812:8Ø, $8848:4C 59 FF. At this point it is interesting to note that the RWTS is loaded and set up. Therefore, I just skipped steps 9, 1Ø and 11 and went right to step 12. Everything went smoothly from there.*

8 Start it up again and stop it after the beep:

```
86ØØG
CØE8
```

9 It has just loaded in its RWTS, but we will not be able to use it at this point because it has not set itself up. Let's make it so it will set itself up and let us have control afterwards. Look at $0846, the original *BOOT1* location. It sets up the reset vector and the slot number where the disk drive resides ($0854 up to $0860). We will have to skip this portion of the code and start our next boot process at $0860. But first, we have to set up the reset vector at $03F2 to jump to the Monitor when we hit RESET after the demo has begun:

```
3F2:59 FF 5A
```

10 Remember, we don't want the code to set up its own values for the reset vector and other items. Therefore, we will begin the next boot at $0860.
***Note:** the drive may recalibrate, but it will read the rest of the program in afterwards.*

```
860G
```

11 As soon as the red light on the disk drive goes off, hit RESET. Looking through the RWTS starting at $B500, one would find that the IOB table begins at $B750 and that some locations go to $B610 to read or write. This location is the main RWTS call to go and read a sector.

12 Now let's move the RWTS down to $2400 where *Super IOB* will use it:

```
2400<B400.BFFFM
```

Insert the disk that has *Super IOB*. Make sure that the disk has a short *HELLO* program. For example, a program which merely CATALOGs the disk and gives control to the user. Boot up with this disk, and after the *HELLO* program gives you control of the Apple, type:

```
BSAVE RWTS.EXODUS,A$2400,L$C00
```

13 Type in the controller at the end of this article and save it by whatever means you usually use.

14 When RUNning *Super IOB* with the *Ultima III* controller installed, you must copy with disk drives that are in slot 6.

Press Y when the program asks if it should INITialize the blank disk. Give the disk a volume number of 2.

What Happened?

After it is finished, *Super IOB* makes some sector edits by changing all the bytes referred to on the small chart below from **B1** to **B2**. This tells *Exodus* that a non-protected disk is being used.

Track	Sector	Change these bytes
00	00	E7
00	0D	4 10 1C 28 34 40 4C 58 64 70 7C 88

In order for *Exodus* to have a protected boot side and a normal DOS player side, the RWTS has to differentiate between the two. The protection scheme relies upon the disk volume number to tell whether the disk is protected or not. A volume number of 1 tells the RWTS that the disk is protected and tells it to get the address and data marks from the table which begins at $B765. A volume number of 2 tells the RWTS that the disk is the player disk and that it should use normal address and data marks. All other volume numbers are rejected. The sector-edit performed on track $00, sectors $00 and $0D ensures that a normal RWTS is always accessing the disk.

Here is an explanation of some of the modifications to *Super IOB* that the controller makes.

60 — makes $1A00—$23FF Applesoft variable space, giving plenty of room for the program to work in.

360 — moves memory from $2400—$2FFF to $B400—$BFFF.

1010 — sets the last track to be copied at 16 and sets up page 3 to call the *Exodus* RWTS at $B610.

1020 — makes the volume to be accessed next a **1**, which indicates a protected disk.

1060 — makes the volume number a **2**, which indicates an unprotected disk. It also tests to see if track zero was read in in order to call the sector-edit-routine.

1110—1120 — performs the above mentioned sector edits so that upon booting, the *Exodus* RWTS will think it is reading the player disk.

62000—62010 — alters the *Exodus* RWTS so that it gets its IOB data from $030A instead of $B750.

Exodus with normal address and data marks is now ready to be backed-up and played. This same procedure, with a few changes, can be used to make *Caverns of Callisto* *COPYA*-able.

controller

```
60 LOMEM: 6656 : HIMEM: 9215 : GOTO 10010
360 POKE 253 ,36 : POKE 255 ,180 : POKE 224 ,12 : CALL 832 : RETURN
1000 REM ULTIMA III CONTROLLER
1010 TK = 0 :ST = 0 :LT = 17 :CD = WR : IO = 772 : GOSUB 360 : POKE 773
     ,16 : POKE 774 ,182 : GOSUB 62000
1020 VL = 1 :T1 = TK : GOSUB 490
1030 GOSUB 430 : GOSUB 100 :ST = ST + 1 : IF ST < DOS THEN 1030
```

```
1Ø4Ø IF BF THEN 1Ø6Ø
1Ø5Ø ST = Ø :TK = TK + 1 : IF TK < LT THEN 1Ø3Ø
1Ø6Ø VL = 2 : GOSUB 49Ø :TK = T1 :ST = Ø : IF TK = Ø THEN GOSUB 111Ø
1Ø7Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST < DOS THEN 1Ø7Ø
1Ø8Ø ST = Ø :TK = TK + 1 : IF BF = Ø AND TK < LT THEN 1Ø7Ø
1Ø9Ø IF TK < LT THEN 1Ø2Ø
11ØØ HOME : PRINT : PRINT "DONE^WITH^COPY" : GOSUB 36Ø : END
111Ø READ LOC : POKE LOC ,178 : IF LOC < > 13448 THEN 111Ø
112Ø RETURN
1ØØ1Ø PRINT CHR$ (4 ) "BLOAD^RWTS.EXODUS,A$24ØØ"
62ØØØ READ LOC : READ NUM : POKE LOC ,NUM : IF NUM < > 1Ø THEN 62ØØØ
62Ø1Ø RETURN
62Ø2Ø DATA 4661Ø ,13 ,46611 ,3 ,46621 ,14 ,46622 ,3 ,46625 ,14
      ,46626 ,3 ,467Ø8 ,3 ,4671Ø ,1Ø
62Ø3Ø DATA 1Ø215 ,13316 ,13328 ,1334Ø ,13352 ,13364 ,13376 ,13388
      ,134ØØ ,13412 ,13424 ,13436 ,13448
```

Zoom Graphix

Phoenix Software

Deprotecting Zoom Graphix
by Michael Decker
(Hardcore COMPUTIST # 12, page 9)

Requirements:
Apple][Plus or equivalent
COPYA and *FID* from *DOS 3.3 System Master*
Two blank disks

Zoom Grafix is a superb piece of programming and a delight to use, and has been a workhorse in my library for years. The recent acquisition of a hard-disk drive prompted me to deprotect it.

After investigating the *Zoom Grafix* disk a bit, I was surprised to find that the bulk of the programs on the disk are written in Applesoft. *Zoom Grafix* is protected against standard DOS 3.3 copy programs by the use of non-standard address marks. This can be easily circumvented by making the appropriate POKEs into DOS before *COPYA* is run. However, the *Zoom Grafix* programs also check the disk for an illegal volume number in addition to the usual techniques of setting the autorun flag ($D6) and messing up the DOS warm-start routine. Since the programs were written in Applesoft, it was fairly easy for me to remove these traps.

The two major steps to deprotecting *Zoom Grafix* involve:

1. Copying *Zoom Grafix* onto a disk which has normal DOS 3.3.
2. Studying the LISTable Applesoft programs and removing the instructions which serve to copy-protect the disk.

What To Do

This procedure is based upon the version of *Zoom Grafix* which is dated *9APR82*. The procedure will also work on the earlier version, but the programs are numbered differently and you will need to find the appropriate lines to modify on your own.

We will use a modified *COPYA* to copy the original disk to a temporary disk which retains the abnormal DOS. Then *FID* will be used to transfer all the files from the temporary disk to a normal

3.3 disk or one that has been initialized with a fast DOS. We also have to write a new boot program for the final deprotected *Zoom Grafix* and remove copy-protection traps that some of the Applesoft programs contain.

The overall procedure I use may seem rather roundabout, but there are no special requirements such as a non-autostart ROM or NMI card. All owners of *Zoom Grafix* should be able to perform this softkey. Let's get started.

1 Boot with a normal DOS 3.3 disk and then RUN the *COPYA* program:

```
PR#6
RUN COPYA
```

2 After *COPYA* is loaded and running, we need to halt it so that some modifications to DOS can be made. Line 7Ø of *COPYA* will also be deleted to eliminate the reLOADing of *COPY.OBJ*:

```
7Ø
CALL -151
B954:29 ØØ
B99Ø:29 ØØ
3DØG
RUN
```

The changes at $B954—$B955 and $B99Ø.$B991 cause DOS to ignore the first bytes of the address headers (normally $D5) and address trailers.

3 After you have made a copy of *Zoom Grafix* delete the file called *GRAFIX* from the copy disk and then restore the proper bytes to the addresses that were modified above. To do this, type:

```
FP
DELETE GRAFIX
CALL -151
B954:C9 D5
B99Ø:C9 DE
3DØG
```

4 Next type in the BASIC program listed below which will serve as the boot program for the final deprotected *Zoom Graphix* disk:

```
1Ø TEXT : HOME :D$ = CHR$ (13 ) + CHR$ (4 )
2Ø PRINT D$ "MAXFILES^1" : PRINT D$ "BLOAD^GRAFIX.OBJ"
3Ø HOME : PRINT "1)^^CONFIGURE^ZOOM^GRAFIX" : PRINT : PRINT "2)^^
    RUN^ZOOM^GRAFIX" : PRINT : PRINT "^^ENTER^YOUR^CHOICE^" ; :
    GET A$ : PRINT A$
4Ø IF VAL (A$ ) < 1 OR VAL (A$ ) > 2 THEN GOTO 3Ø
5Ø A = VAL (A$ ) : ON A GOTO 6Ø ,8Ø
```

```
6Ø POKE 1Ø3 ,1 : POKE 1Ø4 ,96 : POKE 24576 ,Ø
7Ø PRINT CHR$ (4 ) "RUN^GRAFIX^SET-UP" : END
8Ø POKE 1Ø3 ,1 : POKE 1Ø4 ,96 : POKE 24576 ,Ø
9Ø PRINT CHR$ (4 ) "RUN^GRAFIX^PART^ ] ["
```

5 Now, initialize a blank disk with this program in memory. If you like, a fast DOS can be used:

INIT ZOOM LOADER

6 We will use *FID* from the DOS 3.3 System Master to transfer the files from the temporary copy of *Zoom Grafix* to the disk we just initialized. Get *FID* up and running and use the wildcard character (=) to transfer all the files:

BRUN FID

You are now done with the temporary disk. When you CATALOG the final copy disk, you'll notice that the program files all seem to be ØØØ sectors long. This is wrong and we will correct their sizes by loading, deleting, and resaving them as we deprotect them. If you have an Applesoft program editor like *GPLE*, load it in now because it will make your job much easier.

7 The first file we will start up is called *GRAFIX SET-UP*, so load it into memory:

LOAD GRAFIX SET-UP

8 From the end of line 1Ø5, remove the following instruction which checks for a logo on the text screen. Be sure to remove just this one instruction and not the entire line:

```
: IF PEEK (1ØØ3Ø) > 153 THEN !
```

9 Insert line 1Ø7 (to defeat the volume number check) and modify lines 44Ø and 59Ø by typing in the lines listed below:

```
107 LM = Ø
44Ø VTAB 4 :POKE 47147 ,Ø :PRINT CHR$ (4)
       "RUN^GRAFIX^PART^II"
590 NORMAL :TEXT :HOME :PRINT CHR$ (4)"FP"
```

1Ø Check your work and then SAVE the modified *GRAFIX SET-UP* program:

DELETE GRAFIX SET-UP
SAVE GRAFIX SET-UP

11 The next file to be modified is called *GRAFIX PART] [*, so LOAD it in. You may have to CATALOG the disk and trace over the file name if you can't type the left bracket ([ ) from your keyboard:

LOAD GRAFIX PART][

GRAFIX PART][is the main program which contains a number of excellent routines.

12 Make the following modification to *GRAFIX PART][* before resaving it: Enter a line to kill the volume number check:

```
    195 LM = 0
```

Remove from the beginning of lines 990 and 1750 a command which will trash the DOS warm-start routine:

```
: POKE -25150,18:
```

Replace line 880 with the following line:

```
    880NORMAL:TEXT:HOME:PRINT CHR$(4);"FP"
```

13 Check your work and then resave the program. The brackets in the file name will be replaced with I's. Again, you may have to CATALOG and trace over the file name:

```
    DELETE GRAFIX PART ][
    SAVE GRAFIX PART II
```

14 The remainder of the files do not need to be modified, but we will LOAD, DELETE and reSAVE them all so that a CATALOG will show the proper file sizes:

```
    BLOAD GRAFIX.INFO
    DELETE GRAFIX.INFO
    BSAVE GRAFIX.INFO,A$800,L$4D5
```

The next binary file overwrites its own buffer at the default MAXFILES of 3, so:

```
    MAXFILES 1
    BLOAD GRAFIX.OBJ
    DELETE GRAFIX.OBJ
    BSAVE GRAFIX.OBJ,A$9000,L$94D
    FP
```

Final Comments

You now have a fully functioning, deprotected *Zoom Grafix* which you can list, examine, and modify.

GRAFIX PART][is the main program; it pokes in two short machine language routines from 926-935 and 936-973 (the latter switches hi-res screens).

GRAFIX.INFO contains set-up parameters as used by *GRAFIX SET-UP*. These parameters are ultimately passed to *GRAFIX.OBJ*. The latter performs the actual printing tasks, as well as screen flipping and other duties.

The deleted program *GRAFIX* is a loader which is so full of traps it's best replaced by the program which we called *ZOOM LOADER*. Note that the loader resets the Applesoft start-of-program pointers to $6000, so that *GRAFIX SET-UP* and *GRAFIX PART][* load above hi-res page 2 (from 24577 to 34016). We closed our exit routines with FP so that these pointers and MAXFILES would be normalized on exit from the program; if you RESET out of either program, remember that these are left abnormal.

If your deprotected program seems to run but won't actually print, it is probably because you haven't properly run the set-up routines. As I mentioned before, those of you who have a different version than mine (9APR82) will have to list the programs and find the appropriate changes to make.

A final enhancement

Zoom Grafix has an undocumented feature. If you enter a question mark (?) when the initial screen is displayed, (It will say: *MAKE SURE PRINTER IS READY THEN PRESS RETURN TO GO ON*, a date will appear on the screen, presumably the date-of-manufacture. You can change this to whatever date you like with the following method:

```
MAXFILES 1
BLOAD GRAFIX.OBJ
POKE 36926, first number (month)
POKE 36927, second number (day)
POKE 36928, third number (last two digits of year)
BSAVE GRAFIX.OBJ,A$9000,L$94D
FP
```

Breaking Locksmith 5.0 Fast Copy

Putting Locksmith 5.0 Fast Copy Into A Normal Binary File

by C. V. Fields
(Hardcore COMPUTIST # 14, page 15)

Requirements:
Locksmith 5.0, Rev F
A way to **reset** into the Apple Monitor
A blank disk

In the past, I have seen two procedures for placing the **16 SECTOR FAST DISK BACKUP** portion of *Locksmith 5.0* into a normal binary file that you can BRUN. However, neither procedure was easy to follow and both required that you write memory move routines and save memory from Page 0 through the end of the program. This method resulted in a program that was much longer than necessary, in addition to the procedures being difficult to follow.

Most readers of this magazine should be able to follow this procedure which will produce a 46-sector BRUNable version of the fast copy program. It may be a problem for some readers to enter the Apple Monitor at just the right time. I use a *Replay* card, but several methods should work, for example: the Old Monitor ROM or moving your RAM card to Slot 1 (See Chris Rys' softkey for *Sensible Speller* in **The Book Of Softkeys Volume II** page 75 or Hardcore COMPUTIST # 9).

What follows is a narrative of what I did. Those of you who can't wait or don't care about the detail can skip to the step-by-step procedure.

I first stored $11 in all memory locations from $0800 through $95FF so I could determine where the program loads:

```
CALL - 151
N 800:11
801<800.95FFM
```

I then booted *LS 5.0* and selected the *16 SECTOR FAST DISK BACKUP* from the utility menu. The instant the disk drive light went out I pressed the button on my *Replay* card and then selected **M** to go into the Apple Monitor. A quick scan of memory, **800.BFFF**, showed that the program extended from approximately $800 through $3FFF, although some of the code was suspect. There was also some code above $8000 that is moved there during the program initialization and is not needed as part of the binary file we will save.

Because page $60 was empty, I moved Page $8 there so I could boot a DOS 3.3 slave disk:

```
6000<800.8FFM
```

I then booted a DOS 3.3 slave disk, with the HELLO deleted, and went back to the Monitor so I could move page $8 back down to where it belongs:

```
CALL - 151
800<6000.60FFM
```

Then I saved the file with:

```
BSAVE LS 5.0 FAST COPY,A$800,L$3800
```

Now, I knew this file wouldn't work 'as is' because when you BRUN a program, DOS BLOADs it and then JuMPs to the starting address. So the next thing I had to do was locate the entry point of the program so I could put a jump to it at the beginning of the file.

I used the *Inspector/Watson* utility to examine the code. If you BLOAD the file and then step through the buffers on *Inspector/Watson*, you can view the program in memory with all the ASCII text identified. As I scanned the program, I made notes on likely starting points, where all the ASCII text was located and possible code at the beginning and end that could be eliminated.

I then started testing the entry points that I had on my list. As luck would have it, one of the first points I tested, $2002, bombed me into Zero Page (sometimes a bomb means luck) at an address of $AB. Since the Monitor shows the address two steps beyond the break I subtracted $2 and booted the original *LS 5.0* disk again. After it loaded I used the *Replay* card to check location $A9 and found a $60, which is a 'ReTurn from Subroutine' instruction.

Again I BLOADed my test file and this time I went into the monitor and placed a $60 at $A9 before I typed 2002G. Much to my surprise, it worked perfectly.

As I said earlier, some of the code looked unnecessary to the program so I started cutting off the suspect code a page at a time and testing the program until I determined that only Pages $0A through $36 were required. There was also enough room at the beginning of Page $ØA for our starting code and at the end of Page $36 so we could cut to an even full sector. When you BSAVE a file, the starting address and length are saved at the beginning of the first sector. This extends the code by 4 bytes. The length then becomes $36FF - $AØØ = $2CFF and $2CFF - $4 = $2CFB. The code will, therefore, take up $2D or 45 sectors. Add one sector for a track/sector list, and a total of 46 sectors will be needed.

After some code searching, I also discovered that the slot number for all the copying is stored at location $A4B. Adding this to the fact that there is some free space at the beginning of page $A, I wrote a little startup routine that will allow you to change slot numbers. Using this routine you can place the *16 SECTOR FAST BACKUP* file on a hard disk.

Therefore, to put your *16 SECTOR FAST BACKUP* in a normal DOS 3.3 file, follow these steps:

Step-By-Step Procedure

1 Init a DOS 3.3 slave disk and delete the *HELLO* program. I recommend one of the rapid DOS programs such as *Diversi-DOS*, *ProntoDOS* or my favorite, *RapiDOS II*.

```
INIT HELLO
DELETE HELLO
```

2 Boot your original of *Locksmith 5.Ø, Rev F*, and select the *16 SECTOR FAST DISK BACKUP* from the utility menu.

3 The instant the disk drive light goes out (while loading the Fast Disk Backup) exit to the Apple Monitor using your favorite method.

4 Boot your DOS 3.3 slave disk (Note: Since Page $8 is not needed, we don't have to move it out of the way):

```
6 CTRL P
```

5 Enter the Monitor and key in the following hexdump:

```
CALL -151
0A00:  20  58  FC  A9  2C  A2  0C  20
0A08:  35  0A  A9  08  85  25  20  24
0A10:  FC  A9  56  A2  11  20  35  0A
0A18:  A0  05  84  24  20  0C  FD  91
0A20:  28  C9  B9  B0  F7  C9  B1  90
0A28:  F3  E9  B0  8D  4B  0A  A9  60
0A30:  85  A9  4C  02  20  85  FE  86
0A38:  FF  A0  00  B1  FE  F0  05  91
0A40:  28  C8  D0  F7  60
```

6 Check your typing against this listing:

```
ØAØØ- 2Ø 58 FC     JSR  $FC58
ØAØ3- A9 2C        LDA  #$2C
ØAØ5- A2 ØC        LDX  #$ØC
ØAØ7- 2Ø 35 ØA     JSR  $ØA35
ØAØA- A9 Ø8        LDA  #$Ø8
ØAØC- 85 25        STA  $25
ØAØE- 2Ø 24 FC     JSR  $FC24
ØA11- A9 56        LDA  #$56
ØA13- A2 11        LDX  #$11
ØA15- 2Ø 35 ØA     JSR  $ØA35
ØA18- AØ Ø5        LDY  #$Ø5
ØA1A- 84 24        STY  $24
ØA1C- 2Ø ØC FD     JSR  $FDØC
ØA1F- 91 28        STA  ($28),Y
ØA21- C9 B9        CMP  #$B9
ØA23- BØ F7        BCS  $ØA1C
ØA25- C9 B1        CMP  #$B1
ØA27- 9Ø F3        BCC  $ØA1C
ØA29- E9 BØ        SBC  #$BØ
ØA2B- 8D 4B ØA     STA  $ØA4B
ØA2E- A9 6Ø        LDA  #$6Ø
ØA3Ø- 85 A9        STA  $A9
ØA32- 4C Ø2 2Ø     JMP  $2ØØ2
ØA35- 85 FE        STA  $FE
ØA37- 86 FF        STX  $FF
ØA39- AØ ØØ        LDY  #$ØØ
ØA3B- B1 FE        LDA  ($FE),Y
ØA3D- FØ Ø5        BEQ  $ØA44
ØA3F- 91 28        STA  ($28),Y
ØA41- C8           INY
ØA42- DØ F7        BNE  $ØA3B
ØA44- 6Ø           RTS
```

This code will set up the display on the text screen, get the slot number and place it at $A4B, and store a $6Ø at $A9 before performing the aforementioned JMP $2ØØ2.

7 Make a couple of modifications to the main program so that it works with the previous hexdump, then save it:

```
2ØØ5:2Ø 78 12
2ØØ8:AD 43 ØB FØ ØD 4C 1C ØA
1B13:EA EA EA
B6Ø:ØØ
BSAVE FAST COPY.LS, A$AØØ,L$2CFB
```

You should now have a 46-sector, working copy of this great fast copy utility.

CSaver

The Advanced Way To Store Super IOB Controllers

by Ray Darrah
(Hardcore COMPUTIST # 13, page 16)

Requirements:
Apple][with Applesoft in ROM
16K RAM (or language) card
DOS 3.3 (not PRODOS)

After reading Ken Burnell's letter in Hardcore COMPUTIST # 12, I realized that saving *Super IOB* controllers with *The Controller Saver* from **The Book Of Softkeys Volume II** page 146 (Hardcore COMPUTIST # 10) isn't as convenient as:

&Holding

and

&Merging

them. Therefore, I wrote *CSAVER (pronounced "SEE SAVER")* which adds the capability of holding programs and merging them.

Typing It In

Entering the *CSAVER* is quite simple. Merely type in the hexdump at the end of this article and:

```
BSAVE CSAVER,A$4000,L$111
```

To install the *CSAVER*, type:

```
BRUN CSAVER
```

It will appear as if nothing has happened to your precious computer but, in fact, all kinds of strange and wondrous events have occurred: Your entire language card has been filled with the image of your Applesoft ROMS; a small routine that hides and merges programs has been relocated into page $F7 of your language card; a short routine that calls the hider and merger at $F700 has been placed in the end of page three; and the ampersand has been revectored to the short page three routine.

How To Use The CSAVER

Once *CSAVER* has been installed, you should LOAD (yes, an Applesoft file) the controller and then type:

&H

This will put the controller on 'hold', allow you to LOAD *Super IOB*, and finally let you should merge the programs by typing:

&M

It is important to note that when the two programs are merged, the one on 'hold' takes precedence over the one LOADed second. That is: *Lines of the program just LOADed will be overwritten (by the program on 'hold') when the line numbers are the same.*

CSAVER will work on most programs that you wish to merge. Just be sure to first LOAD the program you wish to have precedence. **Note:** Remember to use FP when clearing Applesoft programs when *CSAVER* is installed, not NEW.

The Inner Workings

Those of you who get a little queasy when the conversation swings toward machine language might avoid some discomfort by skipping the following text:

When *CSAVER* is BRUN, the ampersand JuMP vector is set to $3B5. Next, memory locations $4Ø3D through $4Ø57 are copied into locations $3B5 through $3CF. Third, the code from $4Ø1A through $4Ø32 copies the Applesoft ROMs into the language card. The last thing to happen is that the main routines of the *CSAVER* (locations $4Ø58 through $411Ø) are copied into the language card starting at $F7ØØ. This area is usually occupied by the hi-res routines HPLOT, ROT= and XDRAW etc. But since *CSAVER*'s main routines don't use any hi-res routines while holding or merging programs, they can occupy that area.

An Encountered Ampersand

When the Applesoft interpreter comes across the ampersand, it reads one more character (from the input line or BASIC program) into the A-register and JuMPs to address $3F5.

The routine at $3F5 (placed there after *CSAVER* is BRUN) switches the RAM card so that it is just like RAM (you can read from and write to it) and then JuMPs to $F7ØØ which is the entry to the main routines of the *CSAVER*. The enabling of the language card in this manner is necessary because of a routine at $F776 that is like the CHRGET routine (at location $ØØB1) used by Applesoft.

The remainder of the routines in page 3 are used by *CSAVER* as exits. All of them start by disabling the language card and then (depending upon which exit) they JuMP to the final routines used by *CSAVER*.

The CSAVER Main Routines

First of all, *CSAVER* tests to see if a 'hold' was specified. If not, execution continues at $F72E.

The Hold Routine

If a 'hold' was specified, then the code at $F704 through $F712 makes sure that the beginning of program is set $801 (i.e. a program has not already been hidden). If the program pointer isn't set to $801, then *CSAVER* exits via ERROR (at $3BE) which disconnects the language card and JuMPs to $FBE2 (the Monitor bell routine).

Assuming the beginning of program pointer is set correctly, *CSAVER* then stores a $801 into the fake CHRGET routine at $F776. Next, it sets the beginning of program pointer equal to the end of program pointer and simultaneously calculates the length of the program and stores the answer in $FE and $FF. Finally, the hold routine exits via EXIT.HOLD (at $3C4) which disconnects the language card and JuMPs to the Applesoft NEW routine (at $D64B).

The Merge Routine

The code at $F72E through $F731 makes sure that a merge was specified. If not, execution goes to the ERROR routine. If a merge was specified, then the code at $F732 through $F73D makes sure that the beginning of program pointer is not set to $801. If it is, *CSAVER* exits via ERROR. Otherwise, the pointer to the start of variables is set equal to the end of program pointer. This has the effect of erasing all numeric variables and is necessary for the ENTER.LINE routine called later.

Next, *CSAVER* sets the COUT vector to point to DO.1.LIN. This must be done because the ENTER.LINE routine doesn't end with a RTS. Instead, it prints a prompt and waits for another line to enter. Therefore, by setting the COUT vector to DO.1.LIN as soon as ENTER.LINE prints the prompt, *CSAVER* takes over again.

The two PLA instructions at the beginning are meant to remove the JSR to COUT (in order to print the prompt) but eliminate the JSR to the Applesoft command handler the first time through. The two JSRs to CHRGET get the line link pointer. If the MSB is zero, then we have merged the whole program and execution continues at DONE.MERGE.

If the MSB wasn't zero, then DO.1.LIN sets the entry conditions for ENTER.LINE by making LINNUM equal to the line number we wish to enter, putting the tokenized line in the input buffer and setting register Y to 5 more than the length of the line. ENTER.LINE enters the tokenized line just as if you typed it.

CHRGET gets a byte from the program on 'hold', increments its pointer, and returns.

DONE.MERGE starts by setting the beginning of program pointer back to $8Ø1 and then moves the merged program back to $8Ø1 via the Monitor MOVE routine. It does a PR#Ø (JSR SETVID) so that the COUT vector is fixed and then reconnects DOS. Next, it fixes the start of variables pointer by subtracting the length from the end of program pointer. Last, it exits via EXIT.MERGE which disconnects the language card and then JuMPs to a routine in Applesoft which fixes the line number link numbers.

That's all there is to hiding and merging programs.

CSAVER Hexdump

```
4ØØØ: A9 4C 8D F5 Ø3 A9 B5 8D
4ØØ8: F6 Ø3 A9 Ø3 8D F7 Ø3 A2
4Ø1Ø: 1A BD 3F 4Ø 9D B5 Ø3 CA
4Ø18: 1Ø F7 AD 81 CØ AD 81 CØ
4Ø2Ø: A2 ØØ BD ØØ DØ 9D ØØ DØ
4Ø28: E8 DØ F7 EE 24 4Ø EE 27
4Ø3Ø: 4Ø DØ EF BD 5A 4Ø 9D ØØ
4Ø38: F7 E8 EØ B7 DØ F5 6Ø AE
4Ø4Ø: 83 CØ AE 83 CØ 4C ØØ F7
4Ø48: AD 82 CØ 4C E2 FB AD 82
4Ø5Ø: CØ 4C 4B D6 AD 82 CØ 4C
4Ø58: F2 D4 C9 48 DØ 2A A5 67
4Ø6Ø: C9 Ø1 FØ Ø3 4C BE Ø3 A4
4Ø68: 68 CØ Ø8 DØ F7 8D 77 F7
4Ø7Ø: 8C 78 F7 A5 AF 85 67 38
4Ø78: E9 Ø1 85 FE A5 BØ 85 68
4Ø8Ø: E9 Ø8 85 FF 18 4C C4 Ø3
4Ø88: C9 4D DØ D8 A5 67 C9 Ø1
4Ø9Ø: DØ Ø6 A5 68 C9 Ø8 FØ CC
4Ø98: A5 AF 85 69 A5 BØ 85 6A
4ØAØ: A9 4E 85 36 A9 F7 85 37
4ØA8: 68 68 2Ø 76 F7 2Ø 76 F7
4ØBØ: AA FØ 29 2Ø 76 F7 85 5Ø
4ØB8: 2Ø 76 F7 85 51 AØ ØØ 2Ø
4ØCØ: 76 F7 99 ØØ Ø2 C8 AA DØ
4ØC8: F6 C8 C8 C8 C8 4C 6A D4
4ØDØ: AD FF FF EE 77 F7 DØ Ø3
4ØD8: EE 78 F7 6Ø A5 67 85 3C
4ØEØ: A5 68 85 3D A9 Ø1 85 67
4ØE8: 85 42 A9 Ø8 85 43 85 68
4ØFØ: A5 AF 85 3E A5 BØ 85 3F
4ØF8: 2Ø 2C FE 2Ø 93 FE 2Ø EA
41ØØ: Ø3 38 A5 AF E5 FE 85 69
41Ø8: A5 BØ E5 FF 85 6A 4C CA
411Ø: Ø3
```

CSAVER Source Code

```
03F5  AMP.VEC    .EQ $3F5      & command handler
FBE2  BELL       .EQ $FBE2
0067  PRG.BEG    .EQ $67       program beginning pointer
00AF  PRG.END    .EQ $AF       end of program pointer
D64B  NEW        .EQ $D64B     routine that does a new
0069  VARTAB     .EQ $69       pointer to start of variables
0036  COUT.VEC   .EQ $36       location that point to output
                               routine
0050  LINNUM     .EQ $50       used to determine the line we
                               are on right now
0200  BUFF       .EQ $200      Input buffer
D46A  ENTER.LINE .EQ $D46A
00B8  TXT.PTR    .EQ $B8
003C  A1         .EQ $3C       Move from
003E  A2         .EQ $3E       Move to
0042  A4         .EQ $42       Move into
FE2C  MOVE       .EQ $FE2C     Monitor move routine
D4F2  FIX.LINKS  .EQ $D4F2     fixes the line number link bytes
FE93  SET.VID    .EQ $FE93     Routine that does a PR#0
00FE  LEN        .EQ $FE       Length of hidden program
C080  RD.RAM     .EQ $C080     Read Language card
C082  RD.ROM     .EQ $C082     Read ROMS
C083  RD.WR.RAM  .EQ $C083     RAMatize language card
C081  WR.RAM     .EQ $C081     Read ROM, Write RAM
F700  CSAVE1     .EQ $F700     CSAVE is moved to here
                 .OR $4000     out of the way
*-----------------------------------------------------------
* HOOK UP THE AMPERSAND LOCATION
*-----------------------------------------------------------
4000             LDA #$4C      a JMP instr
4002             STA AMP.VEC
4005             LDA #$3D0- END. AMP+BEG.AMP
4007             STA AMP.VEC+1 Make Ampersand
400A             LDA /$3D0- END. AMP+BEG.AMP
400C             STA AMP.VEC+2 Start of program
*-----------------------------------------------------------
* PUT AMPERSAND JUMP ROUTINE INTO PAGE 3
*-----------------------------------------------------------
400F             LDX # END.AMP - BEG.AMP-1
4011  MOVE.AMP   LDA BEG.AMP,X
4014             STA $3D0 - END. AMP+BEG.AMP,X
4017             DEX
4018             BPL MOVE.AMP
*-----------------------------------------------------------
* MOVE THE CSAVE ROUTINE INTO THE RAM CARD
*-----------------------------------------------------------
401A             LDA WR.RAM
```

```
401D              LDA WR.RAM      twice!
4020              LDX #Ø          Start at zero
4022  COPY.ROM    LDA $DØØØ,X     Move ROM into
4025              STA $DØØØ,X     RAMcard
4028              INX
4029              BNE COPY.ROM    Finish page
402B              INC COPY.ROM+2  Next page
402E              INC COPY.ROM+5  LDA and STA
4031              BNE COPY.ROM    Not done!
4033  MOVE.CSV    LDA CSAVE,X     Move CSAVE
4036              STA CSAVE1,X    into RAMcard
4039              INX             Next byte
403A              CPX #END.CSAVE -$F7ØØ
403C              BNE MOVE.CSV    until done
403E              RTS
*------------------------------------------------------------
* DO A HIDE OR MERGE
*------------------------------------------------------------
403F  BEG.AMP     LDX RD.WR.RAM   RAMcard on
4042              LDX RD.WR.RAM   twice!
4045              JMP CSAVE1      Perform function
Ø3BE  ERROR       .EQ $3BE
4048              LDA RD.ROM
404B              JMP BELL        Beep!
Ø3C4  EXIT.HOLD   .EQ $3C4
404E              LDA RD.ROM
4051              JMP NEW
Ø3CA  EXIT.MERGE .EQ $3CA
4054              LDA RD.ROM
4057              JMP FIX.LINKS
      END.AMP
*------------------------------------------------------------
* START OF CSAVE ROUTINES
*------------------------------------------------------------
                  .OR $F7ØØ
                  .TA $4Ø5A
4Ø5A  CSAVE       .EQ $4Ø5A
F7ØØ              CMP #'H
F7Ø2              BNE M.CMP       not an H so try M
*------------------------------------------------------------
* HOLD THE CONTROLLER
*------------------------------------------------------------
F7Ø4              LDA PRG.BEG     Make sure program
F7Ø6              CMP #1          pointer is at $8Ø1
F7Ø8              BEQ TRY.MSB     o.k. so far
F7ØA  HOLD.ERR    JMP ERROR       Error!
F7ØD  TRY.MSB     LDY PRG.BEG+1   See if it is 8
F7ØF              CPY #8          Y, not A because of merge store
F711              BNE HOLD.ERR    No, error
```

```
F713              STA CHRGET+1    for merge later
F716              STY CHRGET+2    CHRGET to get at $8Ø1
F719  HOLD        LDA PRG.END     Make start of program
F71B              STA PRG.BEG     equal to current end of program
F71D              SEC
F71E              SBC #1          Find length of progr
F72Ø              STA LEN
F722              LDA PRG.END+1
F724              STA PRG.BEG+1
F726              SBC #8          Finding length
F728              STA LEN+1
F72A              CLC
F72B              JMP EXIT.HOLD   and do a new
*-------------------------------------------------------------
* TRY FOR MERGE
*-------------------------------------------------------------
F72E  M.CMP       CMP #'M
F73Ø              BNE HOLD.ERR    not M either, error!
*-------------------------------------------------------------
* DO THE MERGE
*-------------------------------------------------------------
F732              LDA PRG.BEG     Make sure
F734              CMP #1          a program is
F736              BNE DO.MERGE    on hold
F738              LDA PRG.BEG+1
F73A              CMP #8
F73C              BEQ HOLD.ERR    Nope!
F73E  DO.MERGE    LDA PRG.END     Squash variable
F74Ø              STA VARTAB      table
F742              LDA PRG.END+1
F744              STA VARTAB+1
F746              LDA #DO.1.LIN   point COUT toO DO.1.LIN
F748              STA COUT.VEC    so that when
F74A              LDA /DO.1.LIN   finished,
F74C              STA COUT.VEC+1  do next one
*-------------------------------------------------------------
* DO ONE LINE
*-------------------------------------------------------------
F74E  DO.1.LIN    PLA             pop off JSR
F74F              PLA
F75Ø              JSR CHRGET      skip LSB link
F753              JSR CHRGET      Get MSB
F756              TAX
F757              BEQ DONE.MERGE  when LINK =Ø, done
F759              JSR CHRGET      get LINNUMBER
F75C              STA LINNUM
F75E              JSR CHRGET
F761              STA LINNUM+1
F763              LDY #Ø          Fill input buffer
```

```
F765  .2         JSR CHRGET
F768             STA BUFF,Y
F76B             INY
F76C             TAX            End of line?
F76D             BNE .2         Nope, move next byte
F76F             INY            Y must equal EOL+5
F77Ø             INY            for the enter line
F771             INY            routine
F772             INY
F773             JMP ENTER.LINE Put this line in the program
F776  CHRGET     LDA $FFFF      Dummy number
F779             INC CHRGET+1
F77C             BNE .1
F77E             INC CHRGET+2
F781  .1             RTS
*-------------------------------------------------------------------
* FINISH MERGE
*-------------------------------------------------------------------
      DONE.MERGE
F782             LDA PRG.BEG    Move program
F784             STA A1
F786             LDA PRG.BEG+1  back to $8Ø1
F788             STA A1+1
F78A             LDA #1
F78C             STA PRG.BEG
F78E             STA A4
F79Ø             LDA #8
F792             STA A4+1
F794             STA PRG.BEG+1
F796             LDA PRG.END
F798             STA A2
F79A             LDA PRG.END+1
F79C             STA A2+1
F79E             JSR MOVE       Move it down!
F7A1             JSR SET.VID    PR#Ø
F7A4             JSR $3EA       Reconnect DOS
F7A7             SEC            Move program end down
F7A8             LDA PRG.END
F7AA             SBC LEN
F7AC             STA VARTAB
F7AE             LDA PRG.END+1
F7BØ             SBC LEN+1
F7B2             STA VARTAB+1
F7B4             JMP EXIT.MERGE Exit via FIX.LINKS
      END.CSAVE
```

The CORE Disk Searcher

by Bryce L. Fowler & Ray Darrah
(Hardcore COMPUTIST # 12, page 19)

Requirements:
48K Apple][with Applesoft
DOS 3.3 (not PRODOS)

Hardcore COMPUTIST publishes some softkeys that involve finding one or more specific bytes on a disk. Often, for one reason or another, the location of these bytes is not known. In these softkeys a disk search utility is required. For those who don't have a program with this capability, I wish to present The *CORE Disk Searcher.*

The *CORE Disk Searcher (CDS)* will search an entire disk in just over a minute and a half for one string of input. *CDS* will display the track, sector and starting byte where each search string is found. Provisions have been made to allow for searching less than the entire disk, searching hard disks, scanning protected disks and skipping tracks or sectors.

Type in the BASIC program listing at the end of this article and:

SAVE CORE DISK SEARCHER

Next, type in the hexdump at the end of this article and:

BSAVE SEARCH.OBJ,A$2F0,L$B9

Using The Program

When *CDS* is RUN, you will first be prompted whether to make any *DOS ALTERATIONS* or not. If you type a [Y], then you may change the address marks, data marks and the option to ignore the checksum or not. If you place a **00** in any of the input string, then that byte is considered ignored. An example would be if you wanted to ignore the second byte of the address start marker you would change the string to **D50096**. This feature allows you to search even some protected diskettes.

Next, you will be prompted for the *DISK SLOT* and *DRIVE* numbers where the disk to be searched will be placed. To enter the default values of the last accessed disk, you merely press RETURN.

Third, you will be prompted for the high track, low track and track step. The *HIGH TRACK* is the highest track you wish to be searched. The *LOW TRACK* is the lowest track you wish searched. The *TRACK STEP* is a whole number that determines whether or not to skip any tracks. A track step of one will evenly search every track. But a track step of two will skip every other track.

A Word About The Inputs

The input values in *CDS* aren't checked very thoroughly (so as to work on drives with more tracks or sectors etc.) so be sure that you enter them correctly. **Note:** *All values in this program are hexadecimal unless otherwise noted.* When two digits are displayed as the default and you must type a preceding zero if you wish to change it to a one digit number (ex. type **ØE** for a high track of $E). Pressing RETURN in the middle of a hexadecimal string will not truncate it at the cursor position. However, pressing RETURN in the middle of an ASCII string will.

Next you are asked for the *HIGH SECTOR, LOW SECTOR* and *SECTOR STEP*. These are similar to the track prompts preceding them except these deal with sectors instead of tracks.

This is followed by the *SLOT FOR PRINTOUT*. A zero will print the search results to the screen. Any other number will attempt to print the results to a printer in the corresponding slot. If you select a slot other than zero, be sure your printer is ON before continuing.

The Wildcard

Next, you will be asked if you wish a wildcard or not. If you answer Y, then the hexadecimal value of this wildcard must be input. If this wildcard is contained in any of the search strings, that byte in the string will match any byte on the disk.

Entering The Search Strings

Finally, you will be asked to enter the strings to search for. There are three types of search strings (displayed at the top of the screen). They are *LOW ASCII, HIGH ASCII* and HEXADECIMAL. *CDS* will first ask you for the type of string that is to follow:

You may press a: quote " for high ASCII
apostrophe ' for low ASCII
dollar sign $ for hexadecimal.

After the type indicator, you may enter the string to search for. Up to eighty search strings may be entered. This should be more than enough for your searching needs. When you finish entering all the search strings, press RETURN when asked for the type of string.

Pausing

CDS will automatically pause every time it finds a search string. To get it going again, press any key. To stop CDS from pausing, press the ESC.

The Assembly Language Subroutine

The machine language portion of the *CORE Disk Searcher* is rather unique. When it is BRUN, *SEARCH.OBJ* hooks itself up to the now-famous ampersand (&) vector. Once hooked up, you pass it commands in the form **& X,Y,Z** where X and Y are the track and sector numbers (respectively) of the sector to be searched, and Z is the number of strings to compare. As soon as Applesoft encounters this statement, *SEARCH.OBJ* will read the specified track and sector (into the input buffer ($2ØØ—$2FF)) and then compare Z number of strings (starting with string Ø) in the first dimensioned array. Therefore, the array that is dimensioned first must be one dimensional and must be set to the strings you wish to search for. In the BASIC program, this is F$.

If you wish to use a wildcard value in the search, you must place a number greater than 127 in location 249 ($F9) and the value of the wildcard in location 25Ø ($FA).

When *SEARCH.OBJ* returns (to whomever called it), location Ø will be incremented if a string has been found and decremented if a disk error has occurred. If a string has been matched, location 1 holds the byte position in the sector where the string starts and location 255 ($FF) holds information about which string it was.

If a string has been found, then another ampersand must not be performed. Instead, a CALL to the 'Continue Scanning' part of the program must occur in order to continue the search.

Closing Notes

Few provisions have been made for error handling. This was done in the BASIC program to ensure compatibility with a wide range of off-line mass storage devices, and in the Assembly program because I wanted it to fit into page 3.

Boink!

I hate programs that don't have some element of humor in them. I program mostly for fun, and want my programs to reflect that. I have, therefore, included a humorous sound routine that is executed just prior to exiting the Applesoft program. The noise it makes sounds like its name, "Boink". You gamers out there might like to use this routine. It's relocatable and very friendly.

Core Disk Searcher BASIC listing

```
1Ø REM -------------------------------------------------
2Ø REM -                  THE CORE                     -
3Ø REM -               DISK SEARCHER                   -
4Ø REM -                                               -
5Ø REM -               BY RAY DARRAH                   -
6Ø REM -------------------------------------------------
7Ø REM
8Ø GOTO 38Ø
9Ø NF = - 1 : IF NS = Z THEN 8ØØ
1ØØ HOME : POKE 34 ,TW : PR# PR : PRINT : VTAB 3
11Ø PRINT "CORE^DISK^SEARCHER" : PRINT : PRINT "STRINGS:"
12Ø FOR A = Z TO NS - 01 : PRINT A + 01 ")^" P$(A) "^-^" ;
13Ø IF P$(A ) = F$(A ) THEN PRINT "LOW^ASCII" : GOTO 16Ø
14Ø IF ASC (P$(A ) ) = ASC (F$(A ) ) - 128 THEN PRINT "HIGH^ASCII" :
    GOTO 16Ø
15Ø PRINT "HEX"
16Ø NEXT : PRINT : IF W$ < > "" THEN PRINT "WILDCARD^=>$" W$ : PRINT
17Ø Y = PEEK (37 ) : POKE 249 ,W1 : POKE 25Ø ,W2 : FOR A = T2 TO T1
    STEP - TS
18Ø FOR B = S1 TO S2 STEP SS : PR# Z : VTAB 01 : HTAB 01
19Ø PRINT "SCANNING^TRACK^" ; : POKE C2 ,A : CALL C1 : PRINT ",^
    SECTOR^" ; : POKE C2 ,B
2ØØ CALL C1 : POKE Z ,Z : & A ,B ,NS : IF PEEK (Z ) = 255 THEN 3ØØ
21Ø IF PEEK (Z ) = Z THEN NEXT : NEXT : POKE - 16368 ,Z : GOTO 37Ø
22Ø HTAB 01 : VTAB Y + 01 : PR# PR :NF = NF + 01 : IF NF / 1Ø < > INT
    (NF / 1Ø ) THEN 24Ø
23Ø PRINT "STRING" SPC( 6 ) "TRACK" SPC( 6 ) "SECTOR" SPC( 6 )
    "BYTE"
24Ø PRINT "^^" NS - PEEK (255 ) + 01 TAB( 14 );
25Ø POKE C2 ,A : CALL C1 : PRINT SPC( 8 ); : POKE C2 ,B
26Ø CALL C1 : PRINT SPC( 8 ); : POKE C2 , PEEK (01 ) : CALL C1 : PRINT
    CM$;
27Ø Y = PEEK (37 ) : IF PEEK ( - 16384 ) < > 155 THEN WAIT - 16384 ,128
28Ø IF PEEK ( - 16384 ) < > 155 THEN POKE - 16368 ,Z
29Ø POKE Z ,Z : CALL C3 : GOTO 21Ø
3ØØ HOME : PRINT CHR$ (7 ) "SECTOR^UNREADABLE!" : PRINT : PRINT
    "PROCEED^TO:" : PRINT
31Ø PRINT "1)^NEXT^SECTOR" : PRINT "2)^BEGINNING^OF^PROGRAM" :
    PRINT "3)^BASIC" : PRINT
32Ø PRINT "^WHICH^?" CH$;
33Ø GET A$ : IF A$ < "1" OR A$ > "3" THEN 33Ø
34Ø HOME : ON VAL (A$ ) GOTO 35Ø ,36Ø ,37Ø
35Ø POKE Z ,Z : GOTO 21Ø
36Ø RUN
37Ø TEXT : CALL 922 : END
```

```
380 TEXT : NORMAL : SPEED= 255 : DIM F$(8Ø ) ,P$(8Ø ) ,HX$(15 )
     ,AD(9 ,2 )
39Ø FOR A = Ø TO 9 : READ AD(A ,Ø ) ,AD(A ,2 ) :HX$(A ) = STR$ (A ) :
     NEXT : FOR A = Ø TO 5
4ØØ HX$(A + 1Ø ) = CHR$ (65 + A ) : NEXT : IF PEEK (768 ) + PEEK (769 )
     < > 155 THEN PRINT CHR$ (4 ) "BRUN^SEARCH.OBJ"
41Ø FOR A = Z TO 8 :AD(A ,1) = AD(A,Ø) + TW :NEXT :AD(A ,1 ) = AD(A,Ø)
42Ø CH$ = CHR$ (8 ) :CU$ = CHR$ (21 ) :CM$ = CHR$ (13 )
43Ø O1 = 1 :Z = Ø :TW = 2 :C1 = 912 :C2 = 918 :C3 = 9Ø5 : PR# Z : IN# Z :
     CALL 1ØØ2
44Ø HOME : PRINT TAB( 9 ) "THE^CORE^DISK^SEARCHER"
45Ø VTAB 5 : PRINT "DOS^ALTERATIONS=>N" CH$;
46Ø GET A$ : IF A$ < > "Y" AND A$ < > "N" AND A$ < > CM$ THEN 46Ø
47Ø PRINT A$ : IF A$ = "N" OR A$ = CM$ THEN 59Ø
48Ø X = Z :Y = TW : GOSUB 1Ø1Ø : VTAB 5 : PRINT "ADDRESS^START=>" ; :
     GOSUB 1Ø8Ø
49Ø GOSUB 1Ø4Ø :X = 3 :Y = 4 : GOSUB 1Ø1Ø : PRINT "ADDRESS^END=>" ; :
     GOSUB 1Ø8Ø
5ØØ GOSUB 1Ø4Ø :X = 5 :Y = 7 : GOSUB 1Ø1Ø : PRINT
51Ø PRINT "DATA^START=>" ; : GOSUB 1Ø8Ø : GOSUB 1Ø4Ø :X = 8 :Y = 9 :
     GOSUB 1Ø1Ø
52Ø PRINT "DATA^END=>" ; : GOSUB 1Ø8Ø : GOSUB 1Ø4Ø
53Ø POKE 47422 ,2Ø1 : IF PEEK (47423 ) = Z THEN POKE 47422 ,41
54Ø PRINT : PRINT "IGNORE^CHECKSUM=>N" CH$; : IF PEEK (47498 ) = Z
     THEN PRINT "Y" CH$;
55Ø GET A$ : IF A$ < > "Y" AND A$ < > "N" AND A$ < > CM$ THEN 55Ø
56Ø IF A$ = CM$ THEN PRINT : GOTO 58Ø
57Ø PRINT A$ : POKE 47498 ,183 : IF A$ = "Y" THEN POKE 47498 ,Z
58Ø PRINT
59Ø PRINT "DISK^SLOT=>" ; :P$ = STR$ ( PEEK (47Ø81 ) / 16 ) : GOSUB
     1Ø8Ø : GOSUB 123Ø
6ØØ IF R < O1 OR R > 7 THEN 59Ø
61Ø POKE 47Ø81 ,R * 16
62Ø PRINT "DRIVE=>" ; :P$ = STR$ ( PEEK (47Ø82 ) ) : GOSUB 1Ø8Ø
63Ø IF P$ < "1" OR P$ > "9" THEN 62Ø
64Ø POKE 47Ø82 , VAL (P$ ) : PRINT
65Ø PRINT "HIGH^TRACK=>" ; :P$ = "22" : GOSUB 1Ø8Ø : GOSUB 123Ø
66Ø T2 = R : IF T2 < Z OR T2 > 35 THEN 65Ø
67Ø PRINT "LOW^TRACK=>" ; :P$ = "ØØ" : GOSUB 1Ø8Ø : GOSUB 123Ø
68Ø T1 = R : IF T1 < Z OR T1 > T2 THEN 67Ø
69Ø PRINT "TRACK^STEP=>" ; :P$ = "Ø1" : GOSUB 1Ø8Ø : GOSUB 123Ø
7ØØ TS = R : IF TS < O1 THEN 69Ø
71Ø PRINT
72Ø PRINT "HIGH^SECTOR=>" ; :P$ = "ØF" : GOSUB 1Ø8Ø : GOSUB 123Ø
73Ø S2 = R : IF S2 < Z THEN 72Ø
74Ø PRINT "LOW^SECTOR=>" ; :P$ = "ØØ" : GOSUB 1Ø8Ø : GOSUB 123Ø
75Ø S1 = R : IF S1 < Z OR S1 > S2 THEN 74Ø
76Ø PRINT "SECTOR^STEP=>" ; :P$ = "Ø1" :GOSUB 1Ø8Ø : GOSUB 123Ø :SS =R
77Ø PRINT
```

```
780 PRINT "SLOT^FOR^PRINTOUT=>" ; :P$ = "Ø" : GOSUB 1Ø8Ø : GOSUB 123Ø
79Ø PR = R : IF PR > 7 THEN 78Ø
8ØØ HOME : PRINT SPC( 13 ) "SEARCH^STRINGS" CM$ : PRINT "USE^A^
     WILDCARD?=>N" CH$;
81Ø P$ = "" :W1 = Z : GET A$ : PRINT A$ : IF A$ < > "Y" THEN 83Ø
82Ø PRINT "WILDCARD^VALUE^=>$" ; :P$ = "ØØ" : GOSUB 1Ø8Ø :W1 = 128 :
     GOSUB 123Ø :W2 = R
83Ø W$ = P$ : PRINT : PRINT "'^=^LOW^ASCII" : PRINT CHR$ (34 ) "^=^
     HIGH^ASCII" :NS = Z
84Ø PRINT "$^=^HEX" : PRINT "<CR>^=^NO^MORE^STRINGS"
85Ø PRINT : PRINT "TYPE^OF^STRING=>" ; : GET A$
86Ø IF A$ < > CHR$ (34 ) AND A$ < > "$" AND A$ < > "'" AND A$ < > CM$
     THEN 85Ø
87Ø IF A$ = CM$ THEN 9Ø
88Ø PRINT A$ : IF A$ < > "$" THEN 94Ø
89Ø P$ = "^^" : FOR A = O1 TO 4 :P$ = P$ + P$ : NEXT : GOSUB 1Ø8Ø
9ØØ P$(NS ) = P$ : FOR A = LEN (P$(NS ) ) TO O1 STEP - O1
91Ø IF MID$ (P$(NS ) ,A ,O1 ) = "^" THEN  NEXT : GOTO 85Ø
92Ø P$(NS ) = LEFT$ (P$ ,A ) :F$(NS ) = "" : FOR A = O1 TO LEN (P$(NS )
     ) STEP TW
93Ø P$ = MID$ (P$(NS ) ,A ,TW ) : GOSUB 123Ø :F$(NS ) = F$(NS ) + CHR$
     (R ) : NEXT A : GOTO 98Ø
94Ø INPUT "" ;P$(NS ) : IF P$(NS ) = "" THEN 85Ø
95Ø F$(NS ) = P$(NS ) : IF A$ = "'" THEN 98Ø
96Ø F$(NS ) = "" : FOR A = O1 TO LEN (P$(NS ) )
97Ø F$(NS ) = F$(NS ) + CHR$ ( ASC ( MID$ (P$(NS ) ,A ,O1 ) ) + 128 ) :
     NEXT
98Ø NS = NS + O1 : IF NS < 81 THEN 85Ø
99Ø GOTO 9Ø
1ØØØ REM FORM P$ FROM ADDRESSES
1Ø1Ø P$ = "" : FOR A = X TO Y : IF PEEK (AD(A ,O1 ) ) = Z THEN P$ = P$ +
     "ØØ" : NEXT : RETURN
1Ø2Ø R = PEEK (AD(A ,Z ) ) :P$ = P$ + HX$( INT (R / 16 ) ) + HX$(R -
     INT (R / 16 ) * 16 ) : NEXT : RETURN
1Ø3Ø REM POKE P$ INTO ADDRESSES
1Ø4Ø A$ = P$ : FOR A = X TO Y :P$ = MID$ (A$ ,TW * (A - X ) + O1 ,TW ) :
     GOSUB 123Ø
1Ø5Ø IF R = Z THEN POKE AD(A ,O1 ) ,Z : NEXT A : RETURN
1Ø6Ø POKE AD(A ,O1 ) ,AD(A ,TW ) : POKE AD(A ,Z ) ,R : NEXT A : RETURN
1Ø7Ø REM INPUT HEX NUMBER
1Ø8Ø A = PEEK (36 ) : PRINT P$; : POKE 36 ,A :B = O1 :A = LEN (P$ )
1Ø9Ø GET A$ : IF A$ = CM$ OR A$ = CH$ OR A$ = CU$ THEN 117Ø
11ØØ IF B > (A ) THEN 1Ø9Ø
111Ø IF (A$ < "Ø" OR A$ > "9" ) AND (A$ < "A" OR A$ > "F" ) THEN 1Ø9Ø
112Ø IF B = O1 AND LEN (P$ ) = O1 THEN P$ = A$ : GOTO 116Ø
113Ø IF B = O1 THEN P$ = A$ + RIGHT$ (P$ ,A - O1 ) : GOTO 116Ø
114Ø IF B = (A ) THEN P$ = LEFT$ (P$ ,A - O1 ) + A$ : GOTO 116Ø
115Ø P$ = LEFT$ (P$ ,B - O1 ) + A$ + RIGHT$ (P$ ,A - B )
116Ø PRINT A$; :B = B + O1 : GOTO 1Ø9Ø
```

```
117Ø IF A$ = CM$ THEN PRINT : RETURN
118Ø IF A$ = CH$ AND B = 01 THEN 1Ø9Ø
119Ø IF A$ = CH$ THEN PRINT A$; :B = B - 01 : GOTO 1Ø9Ø
12ØØ IF B > (A ) THEN 1Ø9Ø
121Ø A$ = MID$ (P$ ,B ,01 ) : GOTO 111Ø
122Ø REM CONVERT P$ TO DECIMAL
123Ø R = Z : FOR B = LEN (P$ ) - 01 TO Z STEP - 01 : FOR C = Z TO 15
124Ø IF MID$ (P$ , LEN (P$ ) - B ,01 ) < > HX$(C ) THEN NEXT
125Ø R = R + INT (16 ^ B * C ) : NEXT B : RETURN
126Ø REM DATA FOR ALTERED MARKS
127Ø DATA 47445 ,24Ø ,47455 ,242 ,47466 ,231
128Ø DATA 475Ø5 ,174 ,47515 ,164
129Ø DATA 47335 ,244 ,47345 ,242 ,47356 ,231
13ØØ DATA 47413 ,1Ø ,47423 ,17Ø
```

Core Disk Searcher hexdump

```
Ø2FØ: A9 ØØ 8D F6 Ø3 A9 Ø3 8D
Ø2F8: F7 Ø3 A9 4C 8D F5 Ø3 6Ø
Ø3ØØ: 2Ø 7B DD 2Ø F2 EB A5 A1
Ø3Ø8: 8D EC B7 2Ø BE DE 2Ø 7B
Ø31Ø: DD 2Ø F2 EB A5 A1 8D ED
Ø318: B7 2Ø BE DE 2Ø 7B DD 2Ø
Ø32Ø: F2 EB A5 A1 85 FF A9 Ø1
Ø328: 8D F4 B7 A9 ØØ 8D FØ B7
Ø33Ø: 8D EB B7 A9 Ø2 8D F1 B7
Ø338: 2Ø E3 Ø3 2Ø D9 Ø3 9Ø Ø3
Ø34Ø: C6 ØØ 6Ø A9 Ø7 85 FE A4
Ø348: FE B1 6B 85 FD C8 B1 6B
Ø35Ø: 85 FB C8 B1 6B 85 FC C8
Ø358: 84 FE A2 ØØ AØ ØØ B1 FB
Ø36Ø: 24 F9 1Ø Ø4 C5 FA FØ ØA
Ø368: DD ØØ Ø2 FØ Ø5 E8 DØ EC
Ø37Ø: FØ Ø8 C8 C4 FD BØ Ø8 E8
Ø378: DØ E4 C6 FF DØ C9 6Ø E6
Ø38Ø: ØØ E8 CA 88 DØ FC 86 Ø1
Ø388: 6Ø A6 Ø1 E8 DØ CE FØ EA
Ø39Ø: A9 A4 2Ø ED FD A9 ØØ 4C
Ø398: DA FD AØ 6E AD 3Ø CØ 98
Ø3AØ: 38 E9 Ø1 DØ FB 88 DØ F4
Ø3A8: 6Ø
```

Search.obj source code

```
1ØØØ  FRM.EVAL     .EQ  $DD7B       Evaluates the BASIC expr
1Ø1Ø  AMP.VEC      .EQ  $3F5        BASIC JSRs here when it
                                    gets an &
1Ø2Ø  INT.CONV     .EQ  $EBF2       Converts FAC into an
                                    integer at $AØ and $A1
1Ø3Ø  COM.CHK      .EQ  $DEBE       Makes sure COMM as
                                    separate parameters
1Ø4Ø  RWTS         .EQ  $3D9        Hooked to RWTS
1Ø5Ø  IOB.TRK      .EQ  $B7EC       RWTS track parameter
1Ø6Ø  IOB.SECT     .EQ  $B7ED       Sector number
1Ø7Ø  FIND.IOB     .EQ  $3E3        Loads A and Y with the
                                    addr of the IOB
1Ø8Ø  IOB.BUF      .EQ  $B7FØ       Pointer to user data
                                    buffer
1Ø9Ø  IOB.CMD      .EQ  $B7F4       IOB command
11ØØ  VAL          .EQ  $A1         Where INT.CONV stores its
                                    answer
111Ø  IOB.VOL      .EQ  $B7EB       RWTS volume expected
112Ø  NUM.STRINGS  .EQ  $FF         Number of strings to
                                    match
113Ø  YSAVE        .EQ  $FE         String currently on
114Ø  ARRAY.PTR    .EQ  $6B         Points to start of arrays
115Ø  LEN          .EQ  $FD         Length of current string
116Ø  STR.PTR      .EQ  $FB         And $FC point to current
                                    string
117Ø  WILD.CARD    .EQ  $FA         Wildcard character
118Ø  WILD.ON      .EQ  $F9         If B7=1 Then wildcard
                                    character is valid
119Ø  BUFF         .EQ  $2ØØ        Sector buffer
12ØØ  FOUND.FLG    .EQ  $Ø          If found, +1
121Ø  POS          .EQ  $1          Position it was found in
122Ø  COUT         .EQ  $FDED       Print A as ASCII
123Ø  PRBYTE       .EQ  $FDDA       Print A as Hex
124Ø  SPEAKER      .EQ  $CØ3Ø       Toggle speaker location
125Ø
126Ø               .OR  $2FØ        $2FØ-$2FF is expendable
127Ø               .TF  SEARCH.OBJ
128Ø
129Ø  *-----------------------------------------------------------
13ØØ  * HOOK UP TO AMPERSAND
131Ø  *-----------------------------------------------------------
132Ø
133Ø               LDA  #MAIN.PRG   LSB
134Ø               STA  AMP.VEC+1
135Ø               LDA  /MAIN.PRG   MSB
```

```
1360                STA  AMP.VEC+2
1370                LDA  #$4C          JMP opcode
1380                STA  AMP.VEC
1390                RTS                Hookup complete
1400
1410 *-----------------------------------------------------
1420 * GET PARAMETERS FOR SCAN
1430 * AND READ SECTOR
1440 *-----------------------------------------------------
1450
1460 MAIN.PRG       JSR  FRM.EVAL      Get track#
1470                JSR  INT.CONV      Integer please
1480                LDA  VAL           Get answer
1490                STA  IOB.TRK
1500                JSR  COM.CHK       Comma
1510                JSR  FRM.EVAL
1520                JSR  INT.CONV
1530                LDA  VAL
1540                STA  IOB.SECT
1550                JSR  COM.CHK
1560                JSR  FRM.EVAL      Put number of
1570                JSR  INT.CONV      strings in VAL
1580                LDA  VAL
1590                STA  NUM.STRINGS
1600                LDA  #1            Read command
1610                STA  IOB.CMD
1620                LDA  #0
1630                STA  IOB.BUF       BUF=$200
1640                STA  IOB.VOL       Any vol
1650                LDA  #2
1660                STA  IOB.BUF+1
1670
1680 *-----------------------------------------------------
1690 * TRY TO GET THE SECTOR
1700 *-----------------------------------------------------
1710
1720                JSR  FIND.IOB      Get IOB.PTR
1730                JSR  RWTS          Get SECTOR
1740                BCC  NO.ERR        DOS could read it
1750                DEC  FOUND.FLG     Tell BASIC
1760                RTS
1770
1780 *-----------------------------------------------------
1790 * SETUP STRINGS
1800 *-----------------------------------------------------
1810
1820 NO.ERR         LDA  #7            First
1830                STA  YSAVE         OFFSETT=7
1840
```

```
185Ø  *---------------------------------------------------
186Ø  * GET THE NEXT STRING
187Ø  *---------------------------------------------------
188Ø
189Ø  GET.STNG     LDY                     YSAVE CURRENT $
19ØØ               LDA  (ARRAY.PTR),Y   Get
191Ø               STA  LEN                length
192Ø               INY
193Ø               LDA  (ARRAY.PTR),Y  ADDR
194Ø               STA  STR.PTR
195Ø               INY
196Ø               LDA  (ARRAY.PTR),Y
197Ø               STA  STR.PTR+1
198Ø               INY
199Ø               STY                     YSAVE 4 next $
2ØØØ
2Ø1Ø  *---------------------------------------------------
2Ø2Ø  * SCAN FOR A STRING
2Ø3Ø  *---------------------------------------------------
2Ø4Ø
2Ø5Ø  TRY.BUF      LDX  #Ø                 BUFF=Ø
2Ø6Ø  TRY.CHAR     LDY  #Ø                 STRING=Ø
2Ø7Ø  NXT.CHAR     LDA  (STR.PTR),Y        Get CHAR
2Ø8Ø               BIT  WILD.ON            Active?
2Ø9Ø               BPL  NO.WILD
21ØØ               CMP  WILD.CARD          Match?
211Ø               BEQ  MATCHED1
212Ø  NO.WILD      CMP  BUFF,X             Match?
213Ø               BEQ  MATCHED1
214Ø               INX
215Ø               BNE  TRY.CHAR
216Ø               BEQ  NXT.STRING
217Ø
218Ø  MATCHED1     INY                     String done?
219Ø               CPY  LEN
22ØØ               BCS  TELL.B             Yes!
221Ø               INX                     Next BUFFR
222Ø               BNE  NXT.CHAR           Not EOB
223Ø
224Ø  *---------------------------------------------------
225Ø  * TRY FOR ANOTHER STRING
226Ø  *---------------------------------------------------
227Ø
228Ø  NXT.STRING
229Ø               DEC  NUM.STRINGS        Done?
23ØØ               BNE  GET.STNG
231Ø               RTS
232Ø
233Ø
```

```
233Ø  *-----------------------------------------------------------
234Ø  * TELL BASIC ABOUT MATCH
235Ø  *-----------------------------------------------------------
236Ø
237Ø  TELL.B      INC  FOUND.FLG     Found!
238Ø              INX                For continue
239Ø  BACKUP      DEX                Backup to
240Ø              DEY                Start of $
241Ø              BNE  BACKUP
242Ø              STX  POS
243Ø              RTS
244Ø
245Ø  *-----------------------------------------------------------
246Ø  * CONTINUE SCANNING
247Ø  *-----------------------------------------------------------
248Ø
249Ø              LDX  POS           Get old POS
250Ø              INX
251Ø              BNE  TRY.CHAR      Go!
252Ø              BEQ  NXT.STRING    .always
253Ø
254Ø  *-----------------------------------------------------------
255Ø  * PRINT A HEX NUMBER
256Ø  *-----------------------------------------------------------
257Ø
258Ø              LDA  #$A4          Print a "$"
259Ø              JSR  COUT
260Ø              LDA  #Ø            Dummy number
261Ø              JMP  PRBYTE        Print it
262Ø
263Ø  *-----------------------------------------------------------
264Ø  * BOINK!
265Ø  *-----------------------------------------------------------
266Ø
267Ø              LDY  #$6E          Set loop
268Ø  TOGGLE      LDA  SPEAKER       Click
269Ø              TYA
270Ø  DELAY       SEC                Delay loop gets
271Ø              SBC  #1            shorter each
272Ø              BNE  DELAY         time through
273Ø              DEY
274Ø              BNE  TOGGLE
275Ø              RTS
```

Modified ROMs

As explained in the article *How To Use This Book* on page 5, some softkeys require the user to enter the Monitor during the execution of a copy-protected program. One way would be to install a modified Reset vector on the computer's motherboard. Here's how.

by Ernie Young
(Hardcore COMPUTIST # 6, page 14)

WARNING

SoftKey Publishing assumes no responsibility for any damage done to the computer while following this procedure.

Requirements:
48K Apple]I Plus
One disk drive
A supply of 2716 16K EPROM chips
Several 24-pin, low-profile sockets
Access to an EPROM burner
Knowledge of Assembly Language is helpful...

In the article ***Hidden Locations Revealed*** (Hardcore COMPUTIST # 3, page 1Ø) the author addressed the idea of what to do when performing a softkey on a program that uses the text page for valid code use.

He suggested a small hardware modification to the Apple to enable one to see the text page of a program while another page (i.e., hi-res page) was supposed to be displayed.

What I am about to discuss is an inexpensive modification to your computer that is about as good (maybe even better) than those cards that save all of memory. Needless to say, this is a very good means to achieve that end on programs which use volatile (easily eraseable) memory.

Volatile memory includes such locations as a great deal of page zero, page one, page two and the text page location at $4ØØ to $7FF. For example, as you type and characters are echoed onto the text screen, locations $4ØØ—$7FF change. Each ASCII character is represented on the screen in one of the locations in the range of $4ØØ to $7FF.

In most softkeys it has been assumed that by hitting RESET (with the old-style Monitor, of course) and dropping into Monitor,we can snoop through memory and find what we need to save or disregard.

When the text page and the keyboard input buffer (page 2) is used for valid program storage, resetting will destroy these volatile memory locations. Indeed, it is no new trick to many software publishers that using the text page and other volatile memory locations is a good way to keep the public from snooping through their programs and possibly reducing them to a more copyable form.

Hitting RESET

To understand this more clearly, let's examine what happens when you press RESET.

Instead of going through the input latch at $CØØØ as the other keys on the keyboard do, the RESET key is connected directly to pin 4Ø of the 65Ø2 microprocessor chip. This is the big long chip just in front of the slots on the motherboard.

When RESET is pushed, pin 4Ø gets connected to ground and the computer unconditionally jumps to the address contained in locations $FFFC and $FFFD. These locations are in the F8-monitor ROM and, depending on which F8-monitor ROM you have, your computer can do one of two things.

There are two flavors of Monitor ROMs in the Apple world, known as the 'autostart' and the 'old-style' Monitor ROMs.

In the 'autostart' F8-ROM, which is used in the Apple][Plus model, these locations point to a series of routines that check to see if the computer is just being powered up or if RESET had been pressed before. If it finds the power already on, it jumps to the routine pointed to by locations $3F2 and $3F3. Normally, they point to the BASIC warm-start and you get the Applesoft prompt.

The 'old-style' Monitor found in the older Apple][models has $59 and $FF stored in locations $FCCC and $FCCD. This causes the Apple to jump to the routines at $FF59. These routines set the keyboard for input, the monitor (or TV) for output, the text page for viewing and, finally, it puts you in the machine language monitor with an asterisk prompt.

In the 'autostart' ROM, anyone can program where they want their Apple to go, when RESET is pressed, by changing the code at $3F2 accordingly. In the 'old-style' Monitor, however, there is no way to prevent a reset from occuring and, eventually, giving you the Monitor prompt. This is obviously essential if you want to break into a program to examine the code but this is no guarantee to performing a working softkey.

Destructive RESET

The problem is that when we RESET into the Monitor, many locations are destroyed. These locations may or may not be essential to thne line when the Monitor prompt is printed. This scrambles locations $4ØØ—$7F8 and destroys locations $4ØØ—$427 completely.

Most software publishers know this and use it against us to keep us 'unwanteds' from snooping through their code. This becomes evident if you hit RESET and find that the text page is filled with garbage and other incomprehensible junk.

Ultimately, we would like to be able to save locations $ØØ — $8FF in non-volatile memory upon hitting RESET and, then, to exit to the Monitor for examination. I include $8ØØ — $8FF, even though it doesn't get destroyed on RESETing, because it gets wiped out when we do a 48K slave disk boot. The best place to store this information would be at $2ØØØ—$28FF, since this is normally the hi-res page used in most games and is not destroyed by booting a slave disk. (Remember that $ØØ—$8FF and $96ØØ—$BFFF are destroyed when booting a 48K slave disk).

In order to save these volatile memory addresses into locations $2ØØØ—$28FF, we need to change what normally happens when we hit RESET. This may seem impossible to do since what happens when we hit that key is predetermined in ROM and, therefore, is not changeable.

Well, yes and no.

We can copy the code from the F8-ROM down to RAM, change it and burn it into a new ROM!

Help from EPROM

Of course, this assumes that you have access to an EPROM (Eraseable, Programmable, Read-Only Memory) programmer and some 2716 EPROM chips.

Most computer stores that are worth anything will be willing to burn you a new 2716 chip for a reasonable fee if you do not have your own access to an EPROM programmer. These 2716 chips are available from many sources. See the back of any *BYTE* magazine for names of suppliers of the EPROM if you can't find any locally in your area

So, assuming you have access to these two resources, all we have to do is alter the normal F8-code.

First, let's develop the code that we will need to jump to when the RESET key is pressed, moving memory from $ØØ—$8FF to $2ØØØ—$28FF. This requires some knowledge of Assembly language but if you are not familiar with machine code try and follow along anyway.

(While going to the extreme of burning new ROM, it would also be a good idea to change the NMI (non-maskable interrupt) vector to point to our new routine).

1 Let's start by moving the code at $F800—$FFFF down to RAM. Location $2800 would be a good place for it.

2 We need a place to put our 'Super Saver' routine. We don't have any open memory locations, but we do have some routines in the monitor code which we never use, like the tape read and write routines. These will have to be sacrificed for our new routines. To enter this code, **carefully** type in the hexdump.

3 Now, change the reset and the NMI vectors in our RAM version of the F8-ROM

```
2FFA:CD FE CD FE
```

4 Save the file onto a disk with the command:

```
BSAVE MODIFIED F8.ROM,A$2800,L$800
```

5 Now, burn your new EPROM!

Plugging-in your New F8 Monitor ROM

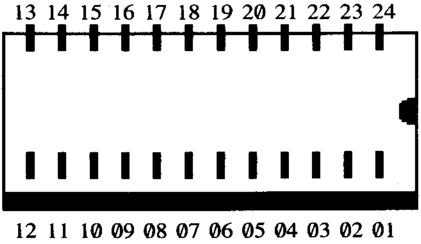
In order to use your new, modified ROM, we must install it in the motherboard (or in an Integer card).

But, first, we have to make one explanation: both the 2716 you just burned and the original 9316 ROM, used by Apple, are Read-Only-Memory devices, containing 2K bytes of information, which gives us 16K bits of information. Hence, 16K ROMs. But, unfortunately, they are not totally compatible. The arrangement of the pin numbers are slightly different.

To use your new EPROM, you must either make these changes directly to the chip itself (not advisable), or to a jumper socket which your new chip will plug into. This, then, will be plugged into the motherboard, or Integer card.

For the latter, you will need a 24-pin, low-profile socket, which is available from Radio Shack or similar stores.

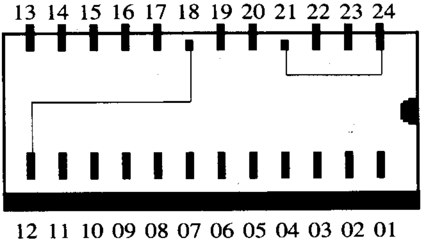
With the socket upside-down and the pins looking you in the face, it should look like this:



Your soldering skills come in handy now. Using some short, hi-gauge wire (wire-wrap is preferrable, but anything in the 26-30 gauge will work), solder a piece between pins 21 and 24 and then solder a piece between pins 12 and 18.

Be extremely careful not to short out the wire or to cross-solder any pins! Also, try and solder as close to the base of the socket as possible, since you have to cut off pins 18 and 21 after you have finished soldering them.

The next step is to cut off pins 18 and 21 as close to the base of the socket as you can, without cutting the freshly soldered wires. Both pins must be short enough so they will not touch the socket you will be plugging this one into. The socket should look like this:



Double-check your soldering and the connections to be certain that pins 18 and 21 are cut off.

Make sure the power is OFF before continuing.

Now, carefully remove the ROM labelled F8 (it is the socket farthest to the left that has 24 pins, as you face the keyboard) and plug this jumper socket into the motherboard. (You could plug this socket into your integer card in the same fashion). Now, plug the modified EPROM into this jumper socket and you will all be done!

Go ahead and turn on your Apple and, if you had followed these instructions correctly, you will see the text page filled with "garbage." At this time, press RETURN to get things going as usual.

How to use the 'Super Saver' F8 ROM

From now on, whenever you press RESET the computer will just freeze (no beep or anything). Then, you must press one of three keys, depending on what you want the computer to do.

First of all, by pressing RETURN the computer will just do the usual kind of reset (i.e. JuMP to BASIC). Secondly, by typing - the computer will act as if you have the 'old-style' F8-ROM and JuMP into the Monitor without any memory saves. Finally, pressing : will engage the 'Super Saver,' thus, moving the volatile memory (locations $0000 through $900) into locations $2000 through $2900, with the stack pointer saved at $2901 and it will put you in the Monitor.

Here's the process again:

1 Acquire a blank 2716 and access to an EPROM programmer.

2 Boot a disk with normal DOS and enter the monitor:

```
CALL-151
```

3 Move the memory from ROM to RAM:

```
2800<F800.FFFFM
```

4 Type in this hexdump.

```
2ECD: 2C 00 C0
2ED0: 10 FB 8D 10 C0 AD 00 C0
2ED8: C9 2D F0 7D B0 03 4C 62
2EE0: FA BA 8E 01 29 A0 00 B9
2EE8: 00 00 99 00 20 B9 00 01
2EF0: 99 00 21 4C FD FE 20 00
2EF8: FE 68 68 D0 6C C8 D0 E7
2F00: 84 3C 84 42 84 3E A9 09
2F08: 85 3F A9 02 85 3D A9 22
2F10: 85 43 20 2C FE 20 2F FB
2F18: 20 58 FC 4C 59 FF 60
```

5 Alter the reset and the NMI vectors:

```
2FFA: CD FE CD FE
```

6 Save the modifications:

```
BSAVE MODIFIED F8ROM,A$2800,L$800
```

7 Burn the blank 2716 with this saved code.

8 Using a low-profile, 24-pin socket, solder pin 12 to pin 18, then solder pin 21 to pin 24.

9 Cut off pins 18 and 21 as close to the socket body as possible.

10 Plug the jumper socket into the F8 socket on the motherboard or Integer card.

11 Plug the modified 2716 into the jumper socket and you are done!

The Armonitor

by Nick Galbreath
(Hardcore COMPUTIST # 12, page 23)

Requirements:
48K Apple with DOS 3.3

Wouldn't it be nice to be able to find the location of that darned *I/O ERROR* on your disk? You could get out your disk-editing program and fix it right then and there!

Or, how about reconstructing track/sector lists from blown files?

What about an easy way of learning what the Disk Operating System (DOS) does?

Dream no more. Your wish has been answered.

This small program, the *Armonitor*, does all of the above, and more.

In a nutshell, the *Armonitor* monitors the disk head, recording and printing all it does in a simple, logical way. The format of the recording is:

X Y Z

where:

X is the operation the disk head is performing.

The possible operations are:

S *for a* **seek** *operation*
R *for a* **read** *operation*
W *for a* **write** *operation*

Y and **Z**, the two-digit hexadecimal numbers which follow **X**, specify the track and sector on which the former operation is in effect.

For example, **R 04 0A** would mean that DOS is attempting to read from track $4, sector $A.

Typing it In

1 Enter the Monitor:

CALL -151

2 Type in this hexdump:

```
0300: A9 4C 8D F5 03 A9 10 8D
0308: F6 03 A9 03 8D F7 03 60
0310: AD 00 BD CD 68 03 F0 0C
0318: A2 03 BD 68 03 9D 00 BD
0320: CA 10 F7 60 A9 20 8D 00
0328: BD A9 39 8D 01 BD A9 03
0330: 8D 02 BD A9 EA 8D 03 BD
0338: 60 48 98 48 20 8B FD AE
0340: F4 B7 BD 65 03 20 ED FD
0348: A9 A0 20 ED FD AD EC B7
0350: 20 DA FD A9 A0 20 ED FD
0358: AD ED B7 20 DA FD 68 85
0360: 48 68 85 49 60 D3 D2 D7
0368: 84 48 85 49
```

3 Save it.

```
BSAVE ARMONITOR,A$300,L$6C
```

How to Use It

Using this utility is quite simple: BRUN the program and then press [&] install the *Armonitor*. Unfortunately, the *Armonitor* really slows down disk-access time and can be a nuisance for long files. However, removing it is as easy as installing it -- just re-press[&] and it will 'unhook' itself, waiting for another [&] to revive it again.

Don't Worry

You can still get Volumes I and II of
The Book Of Softkeys

Volume I: Issues 1-5 ($12.95)

contains softkeys for: Akalabeth | Ampermagic | Apple Galaxian | Aztec | Bag of Tricks | Bill Budge's Trilogy | Buzzard Bait | Cannonball Blitz | Casino | Data Reporter | Deadline | Disk Organizer II | Egbert II Communications Disk | Hard Hat Mack | Home Accountant | Homeword | Lancaster | Magic Window II | Multi-disk Catalog | Multiplan | Pest Patrol | Prisoner II | Sammy Lightfoot | Screen Writer II | Sneakers | Spy's Demise | Starcross | Suspended | Ultima II | Visifile | Visiplot | Visitrend | Witness | Wizardry | Zork I | Zork II | Zork III | **PLUS** how-to articles and program listings of need-to-have programs used to make unprotected backups.

Volume II: Issues 6-10 ($12.95)

contains softkeys for: Apple Cider Spider | Apple Logo | Arcade Machine | The Artist | Bank Street Writer | Cannonball Blitz | Canyon Climber | Caverns of Freitag | Crush, Crumble & Chomp | Data Factory 5.Ø | DB Master | The Dic*tion*ary | Essential Data Duplicator I & III | Gold Rush | Krell Logo | Legacy of Llylgamyn | Mask Of The Sun | Minit Man | Mouskattack | Music Construction Set | Oil's Well | Pandora's Box | Robotron | Sammy Lightfoot | Screenwriter II v2.2 | Sensible Speller 4.Ø, 4.Øc, 4.1c | the Spy Strikes Back | Time Zone v1.1 | Visible Computer: 65Ø2 | Visidex | Visiterm | Zaxxon | Hayden Software | Sierra Online Software | **PLUS** the complete listing of the ultimate cracking program...Super IOB 1.5

Volume I	$12.95	+	$2 shipping & handling
Volume II	$12.95	+	$2 shipping & handling
Both volumes!	$20.90	+	$3 shipping & handling

Foreign orders (except Canada & Mexico), please add $5.00 per book shipping & handling. U.S. funds drawn on U.S. banks only. Most orders shipped within 5 working days, however please allow 4-6 weeks delivery for some orders. Washington State orders add 7.8% sales tax. Send your orders to:

SoftKey Publishing PO Box 110846-B Tacoma, WA 98411

Ordering Volumes I and II of
The Book Of Softkeys

Easy as
1 2 3

Name____________________________________ID# _______

Address ____

City____________________State______Zip____________

Country ____________________ Phone ____________________

VISA ________-________-________-________ Exp.________

Signature ____ BSK3

1 □ Volume I.............. $12.95 + $2 ship/handling

2 □ Volume II............. $12.95 + $2 ship/handling

3 □ Both volumes!..... $20.90 + $3 ship/handling

Foreign orders (except Canada & Mexico), please add $5.00 per book shipping & handling. U.S. funds drawn on U.S. banks only. Most orders shipped within 5 working days, however please allow 4-6 weeks delivery for some orders. Washington State orders add 7.8% sales tax.
Send your order to:

SoftKey Publishing
PO Box 110846-B
Tacoma, WA 98411

We are NOT PIRATES! but we're not fools, either.

We're serious programmers and software users who just want to have backup copies of any software we own. COMPUTIST magazine shows us HOW TO BACKUP COMMERCIAL SOFTWARE regardless of the maker's attempt to stop us from having legal copies. Don't let them stop you from protecting your own rights.

Remove copy-protection

from your valuable library of expensive software. The publisher of COMPUTIST has been showing subscribers how to unlock and modify commercial software for the past 5 years. Don't be one of the users abused by user-FIENDLY locked-up software. Subscribe.

SUBSCRIPTIONS RATES FOR 12 ISSUES:

□U.S.-$32 □U.S. First Class-$45 □Canada, Mexico-$45 □Other Foreign-$75

SAMPLE COPY: □U.S.- $4.75 □Foreign- $8.00

US funds drawn on U.S. bank. Send check or money order to:

COMPUTIST PO Box 110846-B Tacoma, WA 98411

□NEW subscriber □Renew my subscription.

Name____________________

Address ____________________

City __________ State _____ Zip _______

Country__________Phone__________

VISA _____-_____-_____-_____ Exp._____

Signature ____________________ BSK3

FREE STARTER KIT

(a $32.00 value)

With your new subscription, you will receive a FREE software kit, containing several utility programs to help you remove copy-protection and modify your locked-up software.

TOP SECRET ?

Not any more. The publisher of (Hardcore) COMPUTIST magazine has been showing Apple-computer users how to modify and make copyable backups of uncopyable (copy-protected) commercial software for the past 5 years.

Remove copy-protection!

It's legal. It's easy. It's fun. And once the protection is removed (normalized), you can then modify, make backups of all modified versions, pack them on hard disk, and maintain an archival file of originals while you use the backups. Don't risk crashing the original. Use only backups.

WARNING

Softkey information in this Book Of Softkeys should NOT be used to make copies of copyrighted software for illegal distribution. The publisher does NOT condone software piracy.

Don't be a software

PIRATE.

But don't be a fool, either.

This Book Of Softkeys, like the previous volumes, shows you how to remove copy-protection from specific commercial software. This volume shows you how to deprotect the following software packages:

- Alien Addition
- Alien Munchies
- Alligator Mix
- Computer Preparation SAT
- Cut And Paste
- Demolition Division
- Einstein Compiler version 5.3
- Escape From Rungistan
- Financial Cookbook
- Flip Out
- Hi-Res Computer Golf II
- Knoware
- Laf Pak
- Last Gladiator
- Learning With Leeper
- Lion's Share
- Master Type v1.7
- MatheMagic
- Minus Mission
- Millionaire
- Music Construction Set
- One On One
- PFS software
- The Quest
- Rocky's Boots
- Sabotage
- Seadragon
- Sensible Speller IV
- Snooper Troops II
- SoftPorn Adventure
- Stickybear series
- Suicide
- TellStar
- Tic Tac Show
- Time Is Money
- Transylvania
- Type Attack
- Ultima III Exodus
- Zoom Graphics

Plus other helpful articles and utility software LISTings.