For The Serious User Of Apple][Computers

COMPUTIST

August 1986 Issue No. 34 $3.75

Softkeys For:

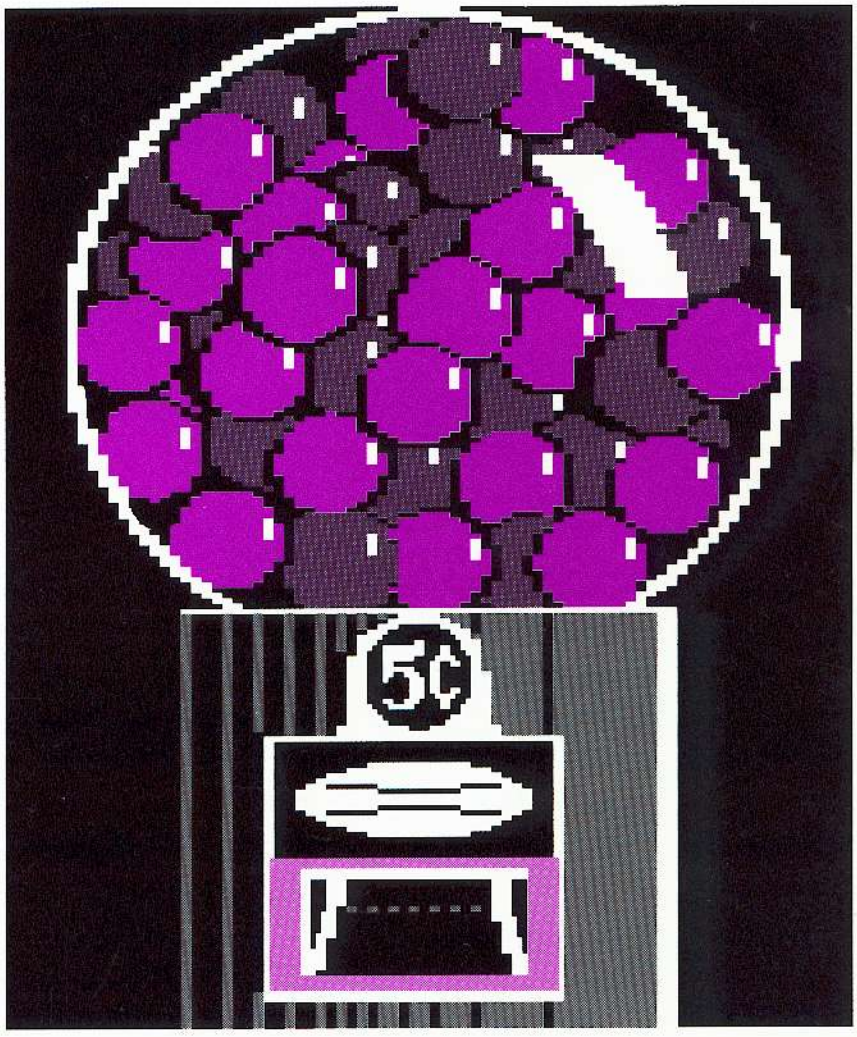
Crisis Mountain
Terrapin Logo
Apple Logo II
Fishies
SpellWorks
Gumball
Conan
Koronis Rift

Core:

Infocom Revealed

Feature:

More ROM Running



(Page 26)

COMPUTIST
PO Box 110846-T
Tacoma, WA 98411

BULK RATE
U.S. Postage
PAID
Tacoma, WA
Permit No. 269

Many of the articles published in COMPUTIST detail the removal of copy protection schemes from commercial disks or contain information on copy protection and backup methods in general. We also print bit copy parameters, tips for adventure games, advanced playing techniques (APT's) for arcade game fanatics and any other information which may be of use to the serious Apple user.

COMPUTIST also contains a special CORE section which focuses on information not directly related to copy protection. Topics may include, but are not limited to: tutorials, hardware/software product reviews and application and utility programs.

What Is A Softkey Anyway? Softkey is a term which we 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).

Commands And Controls: In any article appearing in COMPUTIST, commands which a reader is required to perform are set apart from normal text by being indented and bold. An example is:

PR#6

Follow this with the RETURN key. The RETURN key must be pressed at the end of every such command unless otherwise specified.

Control characters are indicated by being boxed. An example is:

6 CTRL-P

To complete this command, you must first type the number 6 and then place one finger on the CTRL key and one finger on the P key.

Requirements: Most of the programs and softkeys which appear in COMPUTIST 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 (or similar ones) are strongly recommended for readers who wish to obtain the most benefit from our articles:

1) **Applesoft Program Editor** such as Global Program Line Editor (GPLE).
2) **Sector Editor** such as DiskEdit, ZAP from Bag of Tricks or Tricky Dick from The CIA.
3) **Disk Search Utility** such as The Inspector, The Tracer from The CIA or The CORE Disk Searcher.
4) **Assembler** such as the S-C Assembler or Merlin/Big Mac.
5) **Bit Copy Program** such as Copy][Plus, Locksmith or The Essential Data Duplicator
6) **Text Editor** capable of producing normal sequential text files such as Applewriter][, Magic Window][or Screenwriter][

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

Super IOB: This program has most recently appeared in COMPUTIST No. 22. Several softkey procedures will make use of a Super IOB controller, a small program that must be keyed into the middle of Super IOB. The controller changes Super IOB so that it can copy different disks. To get the latest version of this program, you may order COMPUTIST No. 22 as a back issue or order Program Library Disk No. 22.

RESET Into The Monitor: Some softkey procedures require that the user be able to enter the Apple's system 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: 1) Install an F8 ROM with a modified RESET vector on the computer's motherboard as detailed in the "Modified ROM's" article of COMPUTIST No. 6 or the "Dual ROM's" article in COMPUTIST No. 19.

Apple //e - Apple //c: Install a modified CD ROM on the computer's motherboard. Clay Harrell's company (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 //e will void its warranty but the increased ability to remove copy protection may justify it.

Recommended Literature: The Apple][Reference Manual and DOS 3.3 manual are musts for any serious Apple user. Other helpful books include: *Beneath Apple DOS*, Don Worth and Pieter Lechner, Quality Software, $19.95; *Assembly Language For The Applesoft Programmer*, Roy Meyers and C.W. Finley, Addison Wesley, $16.95; and *What's Where In The Apple*, William Lubert, Micro Ink., $24.95.

Keying In Applesoft Programs: BASIC programs are printed in COMPUTIST 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.

10 DATA 67,45,54,52

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

All of this brings us to the 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. Spaces that must be keyed in are printed as delta characters (Δ). All other spaces in a COMPUTIST BASIC listing are put there for easier reading and it doesn't matter whether you type them or not.

There is one exception. If you want your checksums (See "Computing Checksums" section) to match up, you *must not* key in any spaces after a DATA command word unless they are marked by delta characters.

Keying In Hexdumps: Machine language programs are printed in COMPUTIST 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:

CALL -151

Now key in the hexdump exactly as it appears in the magazine ignoring the four digit checksum at the end of each line (a "$" and four digits). If you hear a beep, you will know that you have typed something incorrectly and must retype that line.

When finished, return to BASIC with a:

E003G

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 an assembler. The S-C Assembler is used to generate all source code printed in COMPUTIST. Without this assembler, you will have to translate pieces of the source code into something *your* assembler will understand. A table of S-C Assembler directives just for this purpose was printed in COMPUTIST No. 17. To translate source code, you will need to understand the directives of your assembler and convert the directives used in the source code listing to similar directives used by your assembler.

Computing Checksums Checksums are four digit hexadecimal numbers which verify whether or not you keyed a program exactly as it was printed in COMPUTIST. There are two types of checksums: one created by the CHECKBIN program (for machine language programs) and the other created by the CHECKSOFT program (for BASIC programs). Both programs appeared in COMPUTIST No. 1 and The Best of Hardcore Computing. An update to CHECKSOFT appeared in COMPUTIST No. 18. If the checksums these programs create on your computer match the checksums accompanying the program in the magazine, then you keyed in the program correctly. If not, the program is incorrect at the line where the first checksum differs.

1) To compute CHECKSOFT checksums:

LOAD filename
BRUNCHECKSOFT

Get the checksums with

&

And correct the program where the checksums differ.

2) To compute CHECKBIN checksums:

CALL -151
BLOAD filename

Install CHECKBIN at an out of the way place:

BRUN CHECKBIN,A$6000

Get the checksums by typing the starting address, a period and ending address of the file followed by a CTRL-Y.

xxx.xxx CTRL-Y

And correct the lines at which the checksums differ.

Coping with COMPUTIST

Welcome to COMPUTIST, a publication devoted to the serious user of Apple][and Apple][compatible computers. Our 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.

New readers are advised to read this page carefully to avoid frustration when attempting to follow a softkey or when entering the programs printed in this issue.

a message to our readers

Hello, readers. Let's start by reminding you to fill out and send in the survey we printed in COMPUTIST No. 33. (Look again. It's on pages 31 and 32.) We really would like to know who our readers are and what they are interested in. The survey contains several important questions that ask your opinion on some changes we have been considering making to COMPUTIST.

more hardware

As we pointed out in the last issue, the number of COMPUTIST readers is less than we would like. We are trying several methods for improving this situation. One idea we have is to expand into areas other than softkeys. In particular, we are thinking about including more hardware articles.

Some of the projects we were considering include:

- A two-computer bidirectional communications interface that uses the game port.
- A sixteen-LED panel that shows you in what 4K bank the microprocessor is currently executing.
- A "front panel" that shows you the hexadecimal address of the current instruction being executed. This front panel will also be able to slow your computer and single step the 6502 instructions.
- A card that would allow you to define (and redefine) the character set of your //e (and possibly some][Pluses). This card allows extremely fast, easy-to-progam icon-based graphics.
- The official softkey card. This card has ROM that replaces the motherboard ROM and therefore gives you a whole new (and more powerful) operating enviroment. This card would be practically indespensible for breaking programs.

The hardware projects are listed above in order of price with the least expensive appearing first. We would appreciate it if you would tell us if you are interested in constructing (or purchasing a prebuilt model of) one or more of these devices.

more software

Another area we would like to expand is software printed in COMPUTIST. Programs and articles we've printed have ranged from disk utilities to reviews to games and game cheats. Most of these are submitted by our readers (hint! hint!). Generally, we would like to see enough unique things to keep COMPUTIST different from "other magazines."

Depending on the answers we get out of the reader survey, there may be other additions/improvements to come. The main goal of all this is to make COMPUTIST more useful, educational, and entertaining to a wider audience. We're not selling out to commercialism by shifting our focus.

We **don't** plan to eliminate softkeys from the pages of COMPUTIST. We still feel that copy-protection is un-American. We also believe that you would be better served by our publication if it provided a greater variety of features.

Since you, our readers, are also our writers, we would like to see some articles on a wider range of topics from you. These could be things like hardware modifications, original utility programs, software tips and tricks (DOS, machine language, BASIC), tutorials, stunts you can do with your favorite peripherals and cards, and of course softkeys.

If you have other comments, suggestions, complaints, etc., don't be afraid to write us about them. We have an open mind.

-ed.

FREE COMPUTIST

YES! You Get 2 FREE ISSUES
for every NEW subscriber that you sign up.

We'll extend your current subscription by 2 issues for every new paid subscription you get to fill out the subscriber forms below.

BUT...First Class subscribers MUST recruit First Class subscriptions to receive the 2 extra issues.
This rule (same mailing class) also applies to Foreign rates.
Standard US subscribers can recruit any new subscriptions and extend their standard subscriptions.
US funds drawn on US bank. Send check/money order to: COMPUTIST PO Box 110846-T Tacoma, WA 98411

CURRENT SUBSCRIBER

☐ Add two issues to this subscription.
☐ Renew my subscription. Payment is enclosed.
☐ US: $20 US First Class: $24 ☐ Can/Mex: $34 ☐ Other Foreign: $60

Name _____ ID# _____
Address _____
City _____ State _____ Zip _____
Country _____ Phone _____
VISA MasterCard _____ - _____ - _____ - _____ Exp. _____
Signature _____ CP34

If you let your subscription expire, you are considered a NEW subscriber and are NOT eligible for the free issues.

NEW SUBSCRIBER

Yes, I want to subscribe to your fine publication.
☐ US: $20 US First Class: $24 ☐ Can/Mex: $34 ☐ Other Foreign: $60

Name _____
Address _____
City _____ State _____ Zip _____
Country _____ Phone _____
VISA MasterCard _____ - _____ - _____ - _____ Exp. _____
Signature _____ CP34

CURRENT SUBSCRIBER

☐ Add two issues to this subscription.
☐ Renew my subscription. Payment is enclosed.
☐ US: $20 US First Class: $24 ☐ Can/Mex: $34 ☐ Other Foreign: $60

Name _____ ID# _____
Address _____
City _____ State _____ Zip _____
Country _____ Phone _____
VISA MasterCard _____ - _____ - _____ - _____ Exp. _____
Signature _____ CP34

If you let your subscription expire, you are considered a NEW subscriber and are NOT eligible for the free issues.

NEW SUBSCRIBER

Yes, I want to subscribe to your fine publication.
☐ US: $20 US First Class: $24 ☐ Can/Mex: $34 ☐ Other Foreign: $60

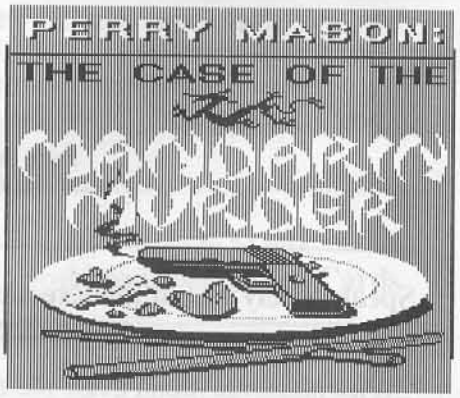
Name _____
Address _____
City _____ State _____ Zip _____
Country _____ Phone _____
VISA MasterCard _____ - _____ - _____ - _____ Exp. _____
Signature _____ CP34

Attention EAMON adventurers

The Computer Learning Center is designing a NEW Public Domain Adventure Game System.

And we are asking for your comments. 'How to improve on the traditional Eamon' is just one of our questions. If you have played Eamon games, tell us what you think could improve them. And if you have created Eamon game(s), tell us how best to improve the editor/dungeon maker. What are your complaints and praises? Do you have any ideas to offer the world of adventuring? Should scenario authors receive more than a mention on the title page of the adventure. Should graphics be used, or should they be more prosaic. What about parodies of popular adventure games, movies, TV shows....

Send your comments to: CLC—EAMON PO Box 110876—T Tacoma, WA 98411



This month's cover:
Graphics from Broderbund's "Gumball."

Address all advertising inquiries to COMPUTIST, Advertising Department, PO Box 110816, Tacoma, WA 98411. Mail manuscripts or requests for Writer's Guides to COMPUTIST, PO Box 110846-K, Tacoma, WA 98411.

Return postage must accompany all manuscripts, drawings, photos, disks, or tapes if they are to be returned. Unsolicited manuscripts will be returned only if adequate return postage is included.

SUBSCRIPTIONS: Rates (for 6 issues): U.S. $20, U.S. 1st Class $24, Canada & Mexico $34, Foreign $60. Direct inquiries to: **COMPUTIST, Subscription Department, PO Box 110846-T, Tacoma, WA 98411.** Please include address label with correspondence.

DOMESTIC DEALER RATES: Call (206) 474-5750 for more information.

Change Of Address: Please allow 4 weeks for change of address to take effect. On postal form 3576 supply your new address and your most recent address label. Issues missed due to non-receipt of change of address may be acquired at the regular back issue rate.

COMPUTIST

August 1986 Issue 34

Publisher/Editor: Charles R. Haight **Managing Editor:** Ray Darrah
Technical Editor: Robert Knowles **Circulation:** Debbie Holloway
Advertising: (206) 474-5750 **Printing:** Valco Graphics Inc., Seattle, WA
COMPUTIST is published monthly by SoftKey Publishing, 5233 S. Washington, Tacoma, WA 98409
Phone: (206) 474-5750

softkeys:

feature:

core:

departments:

Epyx's **Rescue at Rigel** *by M. M. McFadden*, Datamost's **Crazey Mazey** *by Tony Phalen*, Datasoft's **Conan** *by Paul Chan Shun Ho*, Telarium's **Perry Mason: The Case of the Mandarin Murder** *by Charles S. Taylor*, Epyx's **Koronis Rift** *by Contach*

input

Please address letters to:

COMPUTIST
Editorial Department
PO Box 110846-K
Tacoma, WA 98411

Include your name, address and phone number.

Correspondence appearing in the INPUT section may be edited for clarity and space requirements. In addition, because of the great number of letters that we receive and the small size of our staff, a response to each letter is not guaranteed.

Our technical staff is available for phone calls between 1:30 pm and 4:30 pm (PST) on Tuesdays and Thursdays only.

To Copy 40 Tracks

In regard to Mr. Wayne Watkins, COMPUTIST No. 29, page 5 & 6, on how to COPYA his 40-track disk, I found that this method works nicely.

1) Boot DOS 3.3 System Master and load COPYA's machine language part.

PR #6
BLOAD COPY.OBJ0

2) Enter the monitor and modify this code to do 40 tracks.

CALL -151
302:28 N 35F:28

3) Save this modified code.

BSAVE COPY.OBJ0.40,A$2C0,L$10B

4) Return to BASIC and load COPYA.

[C]
LOAD COPYA

5) Modify COPYA by typing the following lines:

```
70 PRINT CHR$(4) "BLOAD COPY.OBJ0.40":REM $2C0
75 POKE 44725,160: POKE 46063,40: POKE 48894,
   40
```

6) Save this new version of COPYA.

SAVE COPYA 40 TRACK

7) Start up this new version and break right at the source drive prompt.

RUN
[C]

8) Delete Line 70 so that it won't reload the machine language portion.

70

9) Enter the monitor and make the following changed to DOS:

CALL -151
B92F:18 60
B98B:18 60
B925:18 60
B988:18 60
BE48:18

10) Start up this new version of COPYA and copy your 40 track disk.

RUN

I cannot take credit for steps 1-6. The credit rests with Mr. R. Boreiko, COMPUTIST No. 21, page 5. You may want to leave the room while COPYA 40 TRACK is copying as the noise could raise the dead.

Incidentally, neither the softkey for *Karateka* nor *Choplifter*, COMPUTIST No. 23, would work for my versions. Was there a bug in either one? I don't have COMPUTIST No. 24 to check. If there was a bug in either one, please print it in an upcoming issue.

Marc Batchelor
Lompoc, CA

Mr. Batchelor: *We currently know of no bug in either the "Choplifter" or "Karateka" softkeys. However we have noticed that the Karateka softkey (as well as most other boot-code trace softkeys) cannot be performed on a //c.*

E-Z Learner Softkey

Requirements:
Apple][Plus, //e
One blank disk

1) Boot the E-Z Learner disk and before the menu comes up (about 4 seconds after boot, or until disk drive changes from running roughly to smoothly) press Reset once. If no cursor appears, press Reset again. If you still do not get a cursor, re-boot and wait a little longer before pressing Reset (but still before the menu comes up).

2) Type the following:

CALL-25153

3) Type the following:

LOAD E-Z LEARNER [R] [B]

4) Type the following:

```
1 TEXT : HOME : NORMAL : PR#0 : IN#0 : CALL 1002:
   B$ = CHR$ (13 ) + CHR$(4 ) : ONERR GOTO 169
```

5) Put the blank disk in the disk drive and type:

INIT E-Z LEARNER

Keith Duff
Mansfield, OH

Speak Nicely About the Executor

A few months ago (COMPUTIST No. 24), you ran an ad by Crane Hill for a batch of utility programs called "Executor" that included two disk data recovery programs. At that time I had been handling a lot of disks for our Apple Computer Club (over 400 members) Library which had been created and written by many non-Apple Disk][disk drives ("compatibles") and, as such, many disks contained glitches, or bad spots usually located between address prolog and epilog byte strings. I found these glitches very hard to locate and even harder and more time consuming to identify and repair. When I saw Mr. Hill's ad for disk repair routines, I was very dubious that they would repair my disks because I had tried every method I had heard about and all the popular disk zap programs I could get my hands on and had spent many, many hours trying to recover bad disks - without much success. So, when I wrote to Crane Hill for his system, I asked for a money back guarantee if I couldn't recover my (good) disk data using his routines. Boy, was I surprised - he agreed and sent me a nice note with the package. And, best of all, the routines worked just as claimed!

I would highly recommend the Executor to any serious Apple enthusiast. In order to use the data recovery program "Tweezers" it is assumed that the user has a working knowledge of DOS disk formats (I recommend "Beaneath Apple DOS") and has a software program that will identify bad sectors on a disk and make a working backup copy of a "bad disk" ignoring the "bad" sectors such as Copy][Plus or the Locksmith Fast Copy routine from Locksmith 5.0.

In addition to the data recovery programs, the disk is full of nice, handy, little utility

input

programs. The second major utility program on the Crane Hill disk is a program called "Cracker" that sets up the Apple for boot code tracing and includes the functions of the Programmer's Aid (old F8) ROM. This little jewel is really handy to use for cracking disks, and Mr. Hill has inlcuded a little tutorial exercise in the instruction manual that comes with the disk.

Thanks for running the ad.

Ken Burnell
Kingdom of Saudi Arabia

Copy][Meets Stubborn Controller Cards

There is something exceedingly peculiar about the ProDOS that comes with Copy][Plus version 6.0. It will not boot (**UNABLE TO LOAD UTIL.SYSTEM**, RELOC/CONFIG ERR, etc.) on some machines (two different Apple][Pluses) that I have tried it on, although those machines have been working fine otherwise. I traced the problem to the disk controller card! I found two disk controller cards that work fine with everything but this! One I tracked down to a particular LS323 & LS259 combination; the other I have not yet traced to the IC level. My advice to anyone encountering this problem is to first try a different 74LS259 (9334).

Keith Davison
Brookline, MA

I, DAMIANO Softkey

I would like to submit the following softkey. "I, DAMIANO" is an interesting graphics adventure game from Bantam Software in which being evil usually makes things easier - until Satan catches up with you! Here's how Bantam protected the disk.

Using Tracer and linguist, one discovers normal formatting except for track $11, which contains no useful data. The boot initially loads sectors 0 to 5 of track 0 into $800; this code loads more DOS into $B700 from track 2. However, before this we find a brief excursion to track $11, where the altered track is looked for.

The deprotection turned out to simply require disabling the subroutine that does the check. To deprotect "I, DAMIANO" simply copy both sides of the entire disk with your favorite quick copier, ignoring any errors on track $11 (the back side is unprotected). Then use a sector editor to find a JSR $083F in track 0, sector 0. On my disk I found the 20 3F 08 at byte $21. Replace this with EA EA EA and you have a COPYAable unprotected backup.

Now let me make a request. How about an article on how to find nibble count routines? So many of your softkey descriptions describe the disabling of nibble counts, yet completely ignore any discussion on how to find and identify the offending code.

A. B. Riley, MD
FPO San Francisco, CA

More Wizardy Stuff

First off, I would like to complement you on a superb magazine. Second, I would like to thank Black Rose for rising to defend fellow pirates (COMPUTIST No. 26). Thirdly, I have a few APT's for the Wizardry series.

1) In COMPUTIST Nos. 28 and 20, there were other Wizardry APT's explaining how to create a "Super Bishop" and in COMPUTIST No.29, to change the class to become a super-samurai, or lord, etc. Why go through all that, when I)dentifying # 5 and J will bring 1000,000,000 exp and GP to the character below the Bishop doing the identifying.

2) In Legacy of Llylgamyn, each character must go through the Rite of Passage Ceremony (making them 1st level, instead of 200 or so), I have found a way to defeat this major drawback. First, create an unmodified character in Wizardry I. Then, using the transfer option of the Utilities, move the character(s) to Wizardry III. Go through the Rite of Passage Ceremony. Transfer the character(s) back to Wizardry I. Use your "Legacy" Bishop to bring these characters up the ranks (Identify 9, J, S). After this is done, have them rest at the Adventurer's Inn, and transfer them back to Wizardry III. It is a good idea to have two parties for Legacy, one dominantly good and one dominantly evil.

3) Wizardry II, The Knight of Diamonds. The answer to the Riddle of the Spinx is on the box cover, in the instruction book and at the beginning of tip #3.

The Zwix

Fahrenheit 451 Softkey

I was recently glancing over a few old issues of COMPUTIST when I came across a softkey for Rendevous with Rama by Jeff Lucia in COMPUTIST No. 19, page 5. I have that game as well as Fahrenheit 451, both by the same company, and both share the same protection method. While the Rendezvous softkey will not work properly for Fahrenheit, I tested the program entitled "IO" and found the correct bytes to change. Here is the softkey for Fahrenheit 451.

Requirements:
Four blank disks
COPYA
Fahrenheit 451

The protection used on Fahrenheit is the same used on Rendezvous with Rama. There is a nibble count on track $10. So all we have to do is find the bytes that check for the nibble count, and defeat them.

1) Copy sides two, three, and four using COPYA or any other normal DOS copier.

2) Run COPYA and fix it so that it ignores any errors.

```
RUN COPYA
70
CALL -151
B942:18
3D0G
RUN
```

3) Copy side one and don't worry about errors on tracks $3 or $10.

4) Boot up a normal DOS after the disk is done copying. Insert side one and load in the program called 'IO".

```
BLOAD IO
```

5) Enter the monitor

```
CALL -151
```

6) Change these bytes so Fahrenheit doesn't check for the nibble count.

```
1BF9:20 29 1C
```

(originally AD 82 C0. Note the Similarity to Rendezvous)

input

7) Save IO back to the disk.

BSAVE IO,A$A00,L$1512

8) You now have a deprotected Fahrenheit 451.

Thank you and keep up the good work with the best magazine around for the Apple.

Greg Poulos
Detroit, MI

The Hobbit & Beast War

I have been subscribing to COMPUTIST for a couple of issues now and let me say that I have enjoyed it tremendously. The softkeys alone are worth the subscription price. Here are a couple of softkeys of my own that might be helpful to someone.

Deprotecting The Hobbit

Upon first inspection the disk seems to be in normal DOS 3.3 format. Logic dictates using COPYA first so I did. The copied disk boots just fine but after displaying the graphics page it shuts the disk drive off and freezes. Since this is a COPYAable disk, it led me to guess that a nibble count had occurred and didn't like what it found. Booting the disk up once again, I waited until the disk drive shut itself off and then pressed Open Apple-CTRL-Reset and CTRL-Reset in rapid succession. This left me in BASIC. Typing CALL -151 to get into the monitor, I checked to see if boot 1 (starts at $801) was still in memory. Since it was, I then followed the code and found it jumped to $200 at $84A. Looking after $200 I found this little bit of code:

```
223- 20 48 03 JSR $348  The protection
                          subroutine?
226- A5 01    LDA $01  get location $01
228- F0 03    BEQ $22D Branch to $22D IF 0
22A- 4C 2A 02 JMP $22A Loop forever
22D- A9 03    LDA #$03 Continue on loading
```

After returning from the subroutine at $348, the program checks memory location $01. If a zero is there, everything is fine and the program continues to load in. However, if anything else is found control passes to $22A which in machine language is the same thing as 10 GOTO 10 is to BASIC. All we have to do now is change the LDA $01 to Load A with a zero and the protection is history. Another victory in the Deprotection Wars!!

Step by Step Instructions

1) COPYA both sides of the original Hobbit disk.

2) On the front side, sector edit Track $00, Sector $01, Bytes $26-27, from A5 01, to A9 00.

3) Write the sector back to the disk and that's all you have to do because the back side is unprotected.

Beast War

This one was easy. Just use the SWAP controller and the resulting copy works like a champ. The one change I made to my copy was to add a "fast" DOS (like Diversi-DOS) to speed up the booting and file loading.

Step by Step Instructions

1) Initialize a slave disk, preferably with a "fast" DOS, and DELETE the hello program.

2) Boot the original Beast War disk.

3) When the program prompts you to input your name, take out the Beast War disk and replace it with your initialized disk (but don't close the disk drive door yet!).

4) Follow the program's prompts.

5) The disk drive will turn on for a second, rattle, beep, and then drop you into BASIC.

6) Get into the monitor.

CALL -151

7) Move the BEAST WAR RWTS to a safe location.

1900<B800.BFFFM

8) Close the disk drive door and boot your initialized disk.

9) Save the RWTS

BSAVE
RWTS.BEASTWAR,A$1900,L$800

10) Make these changes to your SWAP controller before you copy the BEAST WAR disk:

```
1010 TK=3 : LT=35 : ST=15 : LS=15 : CD=WR : FAST=1
10010 PRINT : PRINT CHR$(4 ) "BLOAD
    RWTS.BEASTWAR, A$1900"
```

11) Now use your SWAP controller to copy the BEAST WAR disk and you're all done!

Jim S. Hart
Jacksonville, NC

Most Wanted List

Need help backing-up a particularly stubborn program?

Send us the name of the program and its manufacturer and we'll add it to our Most Wanted List, a column (updated each issue) which helps to keep COMPUTIST readers informed of the programs for which softkeys are MOST needed. Send your requests to:

COMPUTIST
Wanted List
PO Box 110846-K
Tacoma, WA 98411

If you know how to deprotect unlock, or modify any of the programs below, let us know. You'll be helping your fellow COMPUTIST readers and earning MONEY at the same time. Send the information to us in article form on a DOS 3.3 diskette.

Mouse Calc Apple Computer
Apple Business Graphics Apple Computer
Jane Arktronics
Visiblend Microlab
Catalyst Quark, Inc.
Gutenburg Jr. & Sr. Micromation LTD
Prime Plotter Primesoft Corp.
The Handlers Silicon Valley Systems
The Apple's Core: Parts 1-3 The Professor
Fun Bunch Unicorn
Willy Byte ... Data Trek
Cranston Manor Sierra On-Line
Snoggle Broderbund
Robot War Muse
ABM Muse
Mychess II Datamost
Story Tree Scholastic
Agent U.S.A. Scholastic
Handicapping System Sports Judge
Dollars & Sense Monogram
Echo Plus Agranat Systmes
Great Cross Country Road Race Activision
Raster Blaster Budge Inc.
Odin Odesta
Mabel's Mantion Datamost
Brain Bank The Obsevatory
Under Fire Avalon Hill
Crimson Crown Penguin
Crypt of Media Sir Tech
EDD IV Utilico Microware

readers' softkey & copy exchange

M. M. McFadden's softkey for...

Rescue at Rigel

Epyx
1043 Kiel Ct.
Sunnyvale, CA 94089

Requirements:
MUFFIN from DOS 3.3 system master
A blank (or 1/4 empty) disk

Rescue at Rigel is probably one of he first Automated Simulations programs to be published through Epyx. Being five years old, the copy protection is not exactly extreme.

A quick examination with Trax (from the Bag of Tricks disk) shows that, although it is almost standard DOS 3.2, the data prologue marker fluctuates. Searching through the RWTS routines, I discovered that the value for the last byte of the data marker prologue is contained in zero page memory location $31. Why there? It is unaffected by Applesoft, but is zeroed when the monitor is in use. This way, it is impossible to use DOS while in the monitor (which not only screws up attempts to get into the monitor, but also makes it hard to use in a Swap controller with Super IOB).

Anyway, the fastest way to copy this is to use the procedure for Morloc's Tower in COMPUTIST No. 22. Here it is, step by step, with the typos from that softkey corrected.

1) Boot the System Master disk and load MUFFIN.

BLOAD MUFFIN

2) If you have a regular DOS 3.3 disk with at least 100 free sectors, insert it now and skip to step 4.

3) Insert your blank disk now and format it.

INIT HELLO

4) Get into the monitor.

CALL -151

5) Modify MUFFIN's RWTS so that it ignores the last byte of the data prologue.

1A1C:29 00

6) Start up MUFFIN.

803G

7) Use "=" (a wildcard) for the filename and transfer the files from your Rescue at Rigel disk to your DOS 3.3 disk. The first file MUFFIN will come to is HELLO. You may wish to rename this file as RIGEL just to distinguish it as part of Rescue at Rigel.

You now have a working, fully broken copy. To start it, just RUN the HELLO program, or RUN RIGEL if you changed the name.

This same procedure will also work for Hellfire Warrior, and it should work for some of Epyx' other early releases, such as Temple of Asphai and Sorceror of Siva.

Tony Phalen's softkey for...

Crazey Mazey

Datamost, Inc.
8943 Fullbright Ave.
Chatsworth, CA 91311

Requirements:
48K Apple][
A blank disk
COPYA from DOS 3.3 System Master
A sector editor
Crazey Mazey disk

Crazey Mazey is a fun little arcade-type driving game. The object is to outmaneuver or outrun the chase cars while trying to pick up as much money as possible. Dollar signs ($) are placed in the full screen maze and you must run over them to pick them up.

To make a working copy of Crazey Mazey, follow the steps below:

1) Insert your copy of the DOS 3.3 System Master and run COPYA.

RUN COPYA

2) Once loaded, stop the program with [C].

3) Delete line 70 to keep the binary file from being reloaded.

4) Enter the monitor and patch DOS to ignore data and address epilogues, read errors, and the third byte of the data header, in that order.

CALL-151
B925:18 60
B988:18 60
BE48:18
B8FE:00
3D0G

5) Restart COPYA by typing RUN and make a copy of Crazey Mazey.

6) After you copy the disk, boot up your sector editor. Edit track 0, sector 3, and change byte $42 from $38 to $18 (changing a SEC to a CLC). This keeps the computer from getting stuck on track 0 because of errors.

That's it! Boot up and enjoy!

Paul Chan Shun Ho's softkey for...

Conan

Datasoft
19808 Nordhoff Place
Chatsworth, CA 91311

Requirements:
Apple II Plus, //e or //c
One disk drive
Two blank disks
Super IOB v1.5

Conan is a great game inspired by a famous film and series of stories. The mission is to find and slay awesome creatures in a quest of wealth and glory. There are seven screens and levels. In my opinion Conan is one of the best games in my Apple library.

Just like all other Datasoft products, Conan is heavily protected. Conan is a double-sided disk; one side is the program side and the other is the data side.

Fortunately, the disk is fairly normal, except that the address header and end mark had been changed on both sides and there is a nibble count on track $10 of the data side. So, we can use Super IOB and perform some sector edits to deprotect it. Since the nibble count check routine is on the program side, we only need to perform a sector edit on the program side.

The nibble count routine is loaded from track $0, sector $1 into memory starting at $B700 with an entry point at $B731. The routine first seeks the drive head to track $10, turns on the drive, then checks for a byte pattern on the track. If the checking is successful, it will jump to $A000. Otherwise it will jump to a routine to clear the memory and reboot. Therefore we can place a jump to $A000 at the start of the program ($B731), and disable the routine. Moreover, there are many bad sectors on the disk. I used the "ignore unreadable sectors" routine of Super IOB to copy the disk.

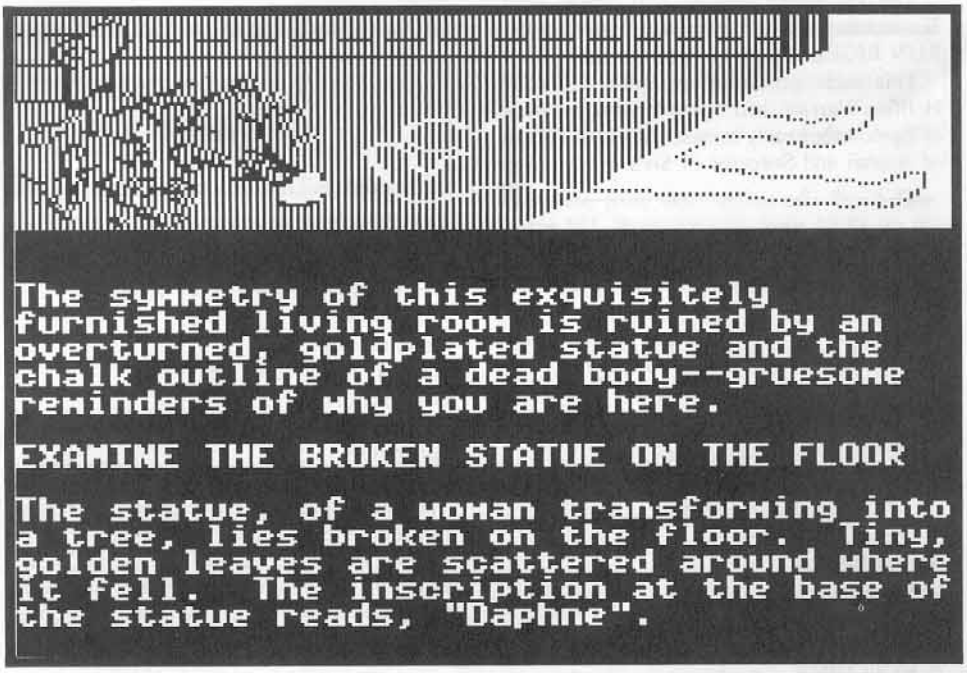
Step By Step

1) Copy the program side of Conan with the program side controller (below) installed into Super IOB 1.5.

2) Copy the data side using the data side controller.

When Super IOB is finished deprotecting the program side, it will have performed the following sector edits:

readers' softkey & copy exchange



Track	Sector	Byte	From	To
$00	$00	$E9	$CC	$D5
$00	$09	$CD	$96	$AA
$00	$09	$D2	$AA	$96
$00	$01	$31	$A9	$4C
$00	$01	$33	$8D	$A0

The first block of sector edits tells Conan to accept normal address markers, the next tells Conan to jump over the nibble count routine directly to the entry point of the program.

Wow!!! You now have a COPYAable Conan disk.

program side controller

```
1000 REM CONAN PROGRAM SIDE
1010 REM IGNORE 1ST ADDRESS END
1020 POKE 47507,0
1030 POKE 775,96 : ONERR GOTO 550
1040 TK=0 :LT=1 :ST=15 :LS=15 :CD=WR :FAST
     =1
1050 GOSUB 490 :T1=TK : IF TK=0 THEN GOSUB 610
     :LT=35 :TK=1
1060 RESTORE : GOSUB 190 : GOSUB 610
1070 TK=PEEK (TRK) : GOSUB 310 : GOSUB 230 :
     GOSUB 490 :TK=T1 : GOSUB 610 : IF PEEK (TRK
     ) =LT THEN 1090
1080 TK=PEEK (TRK) :ST=PEEK (SCT) : GOTO 1050
1090 HOME : PRINT "PROGRAM^SIDE^COPIED" : END
1100 DATA 204,150,170
1110 DATA 5^CHANGES
1120 DATA 0,0,233,213,0,9,205,170
1130 DATA 0,9,210,150,0,1,49,76
1140 DATA 0,1,51,160
```

controller checksums

```
1000 - $356B     1080 - $FECB
1010 - $1270     1090 - $A64B
1020 - $608A     1100 - $3A51
1030 - $F74E     1110 - $4224
1040 - $9065     1120 - $B8BF
1050 - $E1F8     1130 - $C406
1060 - $246E     1140 - $6845
1070 - $6C62
```

data side controller

```
1000 REM CONAN DATA SIDE
1010 REM IGNORE 1ST ADDRESS END
1020 POKE 47507,0
1030 POKE 775,96 : ONERR GOTO 550
1040 TK=0 :LT=35 :ST=15 :LS=15 :CD=WR :FAST
     =1
1050 RESTORE : GOSUB 190 : GOSUB 490 : GOSUB 610
1060 GOSUB 230 : GOSUB 490 : GOSUB 610 : IF PEEK
     (TRK) =LT THEN 1090
1070 TK=PEEK (TRK) :ST=PEEK (SCT) : GOTO 1050
1080 REM RECOPY TRACK 0 SECTOR 0
1090 TK=0 :LT=1 :ST=0 :LS=15
1100 GOSUB 490 : GOSUB 610
1110 GOSUB 490 : GOSUB 610
1120 HOME : PRINT "DATA^SIDE^COPIED" : END
1130 DATA 204,150,170
```

controller checksums

```
1000 - $356B     1070 - $F696
1010 - $1270     1080 - $26B3
1020 - $608A     1090 - $A52D
1030 - $F74E     1100 - $8830
1040 - $F53E     1110 - $310A
1050 - $B4B7     1120 - $6E68
1060 - $8E22     1130 - $A748
```

Charles S. Taylor's softkey for...

Perry Mason: The Case of the Mandarin Murder

Telarium Corp.
One Kendall Square
Cambridge, MA 02139

Requirements:
Disk Muncher
A sector editor
Four blank disk sides

To deprotect PERRY, I adapted the method used on Rendezvous With Rama by Jeff Lucia in COMPUTIST No. 19.

Sides two, three and four are all unprotected and can be copied with any fast copier. Side one is sensitive to the copier used. I achieved about a 50% success rate using Disk Muncher to copy side 1 and making the following sector edits:

Track	Sector	Byte	From	To:
$17	$0B	$5C	$AD	$20
$17	$0B	$5D	$82	$29
$17	$0B	$5E	$C0	$1C

I found the same byte sequence of AD 82 C0 in several other places on the disk; however, changing them is unnecessary.

Contach's softkey for...

Koronis Rift

Epyx Computer Software
1043 Kiel Court
Sunnyvale, CA 94089

Requirements:
48K Apple][and up
COPYA
Original Koronis Rift disk
A sector editor
A blank disk

readers' softkey & copy exchange

The first thing that I do when I buy a program is try to back up the program. I especially want to do so if the program writes to itself. I was able to make a copy of the program using a popular bit copier. However, the copy was not very reliable and I felt there was a better way of backing up the disk. With a little bit of snooping it looked like the protection could be easily removed.

The first thing that I did was observe the boot process. I did not notice anything unusual except there was no cursor. This was not bad but it meant that there were probably no files on the disk. After I did not see anything unusual with the boot, I decided to look at the disk with a sector editor. I got the sector editor running and tried to look at a sector I chose at random. This produced an error from the sector editor. I tried turning the checksums off on the sector editor and read the same sector. This time I was able to read the sector. Knowing this, I tried running COPYA ignoring the checksums.

I tried booting the copy that I had just made. The program would start to load but it would reboot after a certain point. This looked like a nibble count and I decided to try to find it. Since I knew the disk would fail a test and reboot, I ran my sector editor and scanned the disk for a $C600 (as in JMP $C600) by looking for the bytes 00 06. This is where the program would have to jump to reboot. The scan produced two of them on track 1C. After looking at the code, I eliminated one of them. The next trick was finding the code that jumped to this location. With a little bit of looking, I found the nibble count on the same sector. The nibble count had several branches to the reboot routine. I defeated the nibble count by changing these branches to jump to where the nibble count returns the control back to the program. I wrote the sector back to the disk and booted the disk. The disk booted and the program worked perfectly.

This is part of the code for the nibble count:

```
1585- F0 1E      BEQ $15A5
1587- C9 EE      CMP #$EE
1589- D0 F4      BNE $157F
158B- A0 07      LDY #$07
158D- BD 8C C0   LDA C08C,X
1590- 10 FB      BPL $158D
1592- D1 48      CMP ($48),Y
1594- D0 0F      BNE $15A5
1596- 88         DEY
1597- 10 F4      BPL $158D
1599- A2 00      LDX #$00
159B- BD 00 08   LDA $0800,X
159E- 95 00      STA $00,X
15A0- CA         DEX
15A1- D0 F8      BNE $159B
15A3- 18         CLC
15A4- 60         RTS
15A5- (start of reboot code)
```

If the branches at $1585 and $1594 are taken (meaning a bad disk), the program goes to the code at $15A5 and eventually reboots or hangs. Otherwise, the program ends up at $1599 where some memory is copied down to the zero page ($0000-$00FF) and then returns to the calling program.

Every branch to $15A5 should be changed to branch to $1599 instead. This means changing the second byte of the branch to a new value. Calculating the new length of the branch is a matter of counting the number of bytes between the old address and the new address and subtracted from the current branch length.

For example, the old address is $15A5 and the new address is $1599. The offset between these addresses is $0B ($15A4 - $1599 = $0B). The byte at $1595 (the second byte of the BNE instruction at $1594) is changed from $0F to $03 ($0F - $0B = $03). This is done for all the branches going to $15A5.

Step by Step

1) Get COPYA into memory.

RUN COPYA

2) When it asks for the slot and drive numbers, stop the program with [CTRL]-C.

3) Go into the monitor and tell DOS to ignore the checksums and epilogs when reading.

CALL -151
B925:18 60
B988:18 60
BE48:18
3D0G

4) Delete line 70 from COPYA and copy the disk.

70
RUN

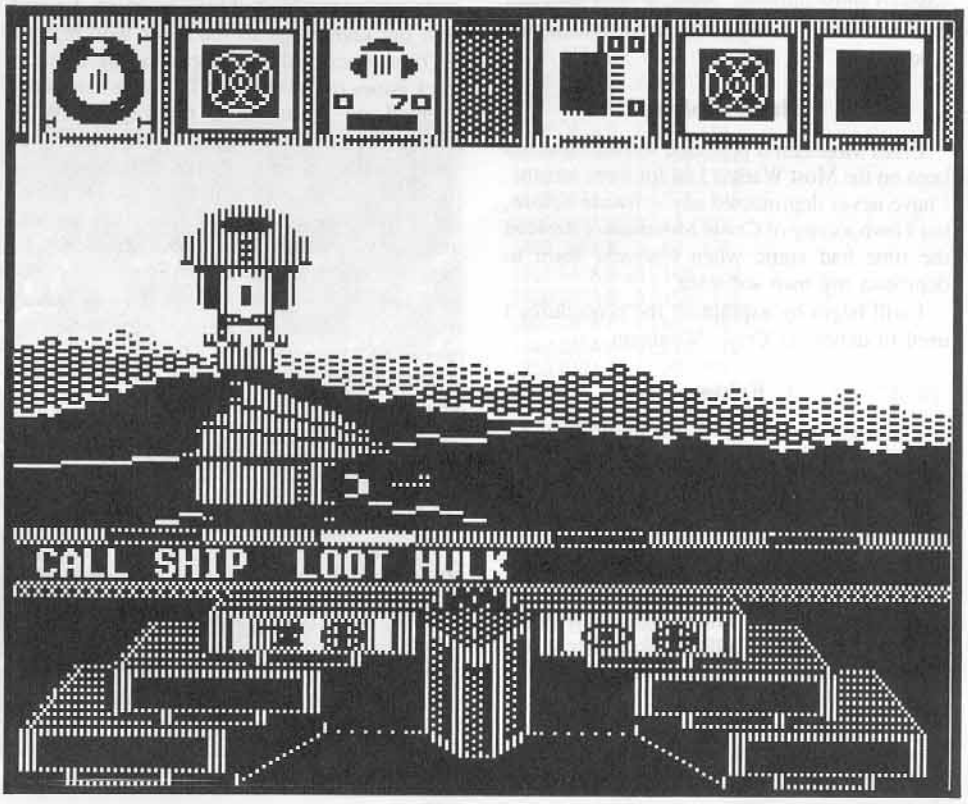
5) Run your favorite sector editor and read track $1C, sector $E.

6) Change the following bytes.

Byte	From	To
3E	66	5A
43	61	55
53	51	45
61	43	37
6E	36	2A
77	2D	21
86	1E	12
95	0F	03

7) Write the sector back out to the disk.

8) Enjoy the game.



Crisis Mountain

by Rene Gaudet

Synergistic Software
830 N. Riverside Drive, Suite 201
Renton, WA 98055

Requirements:
Crisis Mountain
A blank disk
Super IOB 1.5
A sector editor (optional)
A way to enter the monitor (optional)

Crisis Mountain is a variation of the Donkey Kong genre of games. The object of the game is to defuse bombs in the caverns of an active volcano while jumping, dodging, and avoiding boulders and a pesky radioactive bat named "Bertrum."

The Problem

Crisis Mountain is protected software and has been on the Most Wanted List for three months. I have never deprotected any software before, but I own a copy of Crisis Mountain. I decided the time had come when I should learn to deprotect my own software.

I will begin by explaining the procedures I used to deprotect Crisis Mountain.

Friday

The weatherman said that it would be rainy all weekend, so I brought out my copy of DiskView (a nibble viewer) from COMPUTIST and took a look at Crisis Mountain. After two hours of examination, I learned about sync, address, data, and checksum bytes, and what I discovered is illustrated in table 1.

Armed with this knowledge, I wrote a controller for Super IOB. Unfortunately, the controller wouldn't work and I kept getting "Drive Errors". After spending two more hours trying to get the controller to work, I got frustrated with my lack of progress and quit for the night.

Table 1:

Marker	DOS 3.3	Crisis Mtn.
Start Address	D5 AA 96	D5 AA 96
End Address	DE AA	FF FF
Start Data	D5 AA AD	D5 AA AD
End Data	DE AA	FF FF
Sync Byte Before Address Mark	FF	EB
Sync Byte Before Data Mark	FF	AB

Saturday

Still raining outside, I went back to the Apple with a new plan: boot code tracing. Three hours later, I was perplexed at how any program could ever run on this blasted little machine. I noted that my knowledge of the inner workings of DOS was seriously lacking. Reading through back issues of COMPUTIST, I came across the method of capturing a program's RWTS (Read/Write Track/Sector) section and incorporating it into a Super IOB controller. Also, in an input letter in COMPUTIST No. 25, Brian Snook described a way to enter the monitor with a 100 ohm resistor. Radio Shack, here I come! I put down my twenty cents for two 100 ohm resistors and, on the way home, I stopped at my local computer store to pick up a copy of "Beneath Apple DOS" by Worth and Lechner to help my understanding of Apple DOS.

I booted up Crisis Mountain and popped into the monitor with my resistor by connecting it between pins 26 and 29 (any slot except the auxiliary slot on the //e) and moved the RWTS to $1900. Then I booted DOS 3.3 by typing **C600G** and BSAVEd it to my Super IOB disk.

With the Crisis Mountain RWTS, I used the standard fast swap (NewSwap) controller with Super IOB 1.5 to copy the Crisis Mountain diskette. The copy of Crisis Mountain would not boot. Cataloging the newly created diskette showed no files, so I decided to read "Beneath Apple DOS" and try again the next day.

Sunday

I searched the disk for any signs of the VTOC, catalog, and track/sector lists of files if there were any. (Refer to your DOS manual for more information on diskette organization). I only found track/sector lists, and the VTOC and catalog sectors normally found on track 17 ($11) were entirely zeroed out. I rebuilt the VTOC and two catalog sectors by making a simulated catalog entry for each possible track/sector list that I found. However, rebuilding the VTOC did not help deprotect the program because Crisis Mountain accesses the track/sector lists directly from the program. A rebuilt catalog will help readers who wish to develop Advanced Playing Techniques (cheats) for the program. For example, files "B" and "K" (my names) are the two different playing field screens.

Using DiskEdit (a sector editor) from COMPUTIST, I compared Crisis Mountain's RWTS to the DOS 3.3 RWTS and came up with the following differences:

$Addr	Trk,Sct,Byte	Crisis Mtn	DOS 3.3
B83E	0,2,3E	AB	FF
B84A	0,2,4A	A9	48
B84B	0,2,4B	AB	68
B84D	0,2,4D	8D	B9
B84E	0,2,4E	B7	B8
B89E	0,2,9E	FF	DE
B8A3	0,2,A3	FF	AA
B8A8	0,2,A8	FF	EB
B935	0,3,35	FF	DE
B93F	0,3,3F	FF	AA
B991	0,3,91	FF	DE
B99B	0,3,9B	FF	AA
BA29	0,4,29	AA	96
BAAA	0,4,AA	00	AA

On the copy of Crisis Mountain, I applied the Apple DOS values listed above into the Crisis Mountain RWTS. The diskette still would not boot. Browsing through the Crisis Mountain RWTS I found some code in what is supposed to be the DOS primary buffer ($BB00-$BBFF). Disassembling the sector with the "B" command in DiskEdit showed a read to the diskette with the old sector marks. The following patch was applied to correct the read:

$Addr	Trk.Sct	Byte	Crisis Mtn	DOS 3.3
BB24	0,5	24	EB	FF

After applying the above change, the copy of the program would boot, run, and was copyable.

Summary

I am sure that someone with more experience could have deprotected Crisis Mountain in a more simple or elegant manner, but this was my first attempt to deprotect any software. In conclusion, don't wait for terrible weather before you try to deprotect software. Who knows, you might even learn something (like I did). I highly recommend the beginner get the following tools: A sector editor (like DiskEdit), a nibble viewer (like DiskView or The Nibbler), "Beneath Apple DOS" by Don Worth and Pieter Lechner for valuable DOS information, and back issues of COMPUTIST to observe methods of deprotection.

Step By Step

1) Start by creating a RWTS to read the Crisis Mountain diskette. Boot a DOS 3.3 diskette. Enter the monitor and copy (Move) the normal RWTS to lower memory.

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

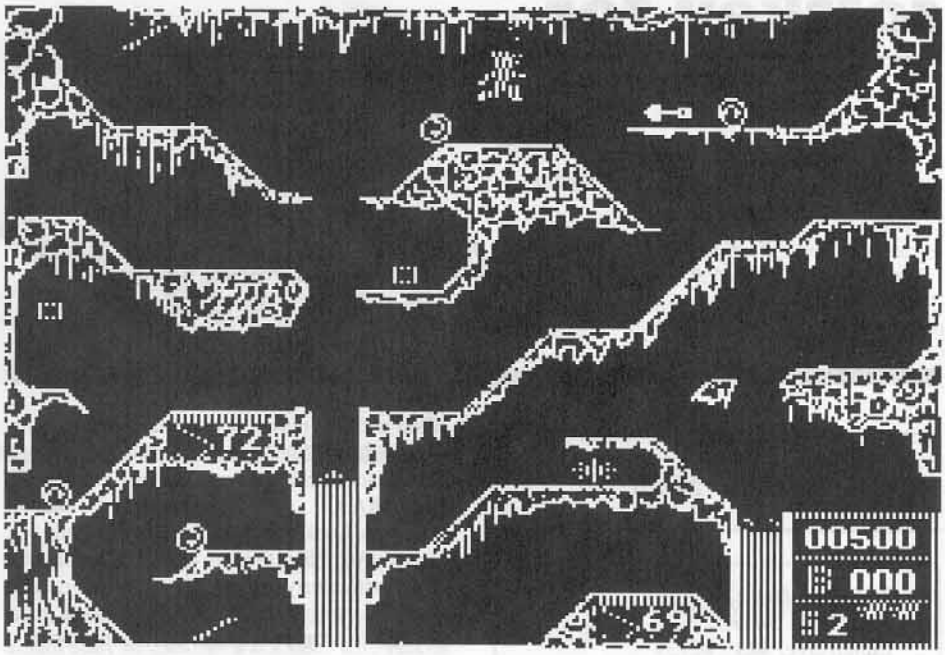
2) Change the normal disk reading values (prologues, epilogues, and sync bytes) in the RWTS to the strange ones.

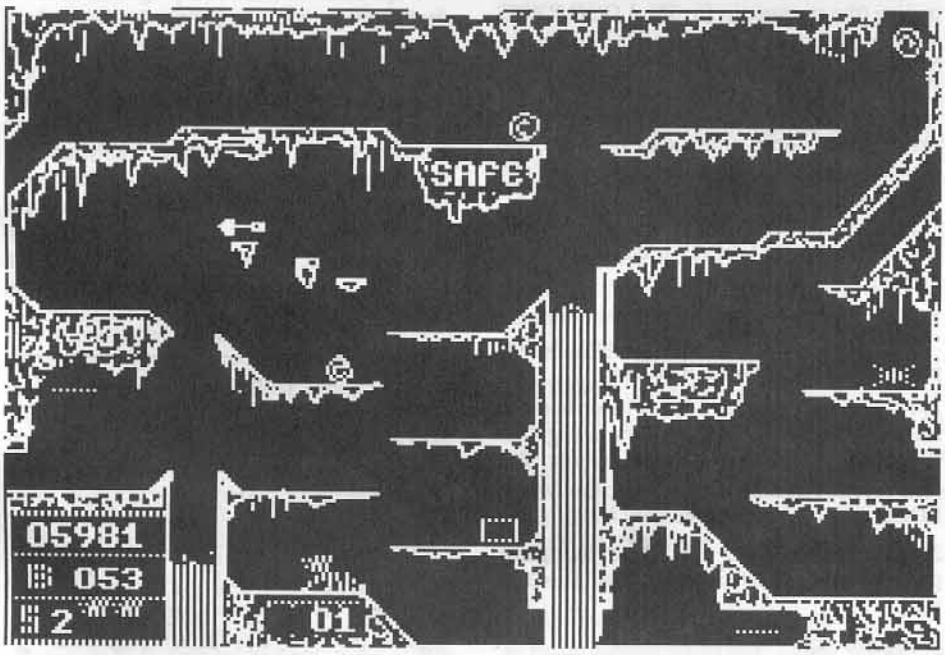
```
193E:AB
199E:FF
19A3:FF
19A8:FF
1A35:FF
1A3F:FF
1A91:FF
1A9B:FF
1B69:AA
1BAA:00
```

3) Return to BASIC and save the new RWTS to your Super IOB disk.

```
3D0G
BSAVE RWTS.CRISIS,A$1900,L$800
```

4) Here's the Super IOB controller to do the rest of the work. Just install it into Super IOB 1.5 as usual and RUN it.





controller

```
1000 REM  CRISIS MOUNTAIN CONTROLLER
1010 TK=0:LT=35:ST=15:LS=15:CD=WR:FAST
     =1
1020 GOSUB 360:GOSUB 490:GOSUB 610:IF TK=
     0 THEN RESTORE:GOSUB 310
1030 GOSUB 360:GOSUB 490:GOSUB 610:IF PEEK
     (TRK)=LT THEN 1050
1040 TK=PEEK(TRK):ST=PEEK(SCT):GOTO 1020
1050 HOME:PRINT "COPY^ DONE":NED
1500 DATA 15^ CHANGES
1510 DATA 0,2,62,255
1520 DATA 0,2,74,72
1530 DATA 0,2,75,104
1540 DATA 0,2,77,185
1550 DATA 0,2,78,184
1560 DATA 0,2,158,222
1570 DATA 0,2,163,170
1580 DATA 0,2,168,235
1590 DATA 0,3,53,222
1600 DATA 0,3,63,170
1610 DATA 0,3,145,222
1620 DATA 0,3,155,170
1630 DATA 0,4,41,150
1640 DATA 0,4,170,170
1650 DATA 0,5,36,255
10010 PRINT CHR$(4)"BLOAD^
     RWTS.CRISIS.A$1900"
```

controller checksums

```
1000 - $356B      1560 - $A14A
1010 - $2544      1570 - $F621
1020 - $18BE      1580 - $CD0E
1030 - $2A2A      1590 - $6312
1040 - $8B66      1600 - $5107
1050 - $C511      1610 - $C94E
1500 - $3EEC      1620 - $E91B
1510 - $DE82      1630 - $733A
1520 - $8AA3      1640 - $9284
1530 - $8C88      1650 - $600B
1540 - $C34D      10010 - $6195
1550 - $7C64
```

softkey for...

Terrapin Logo

by Tom Moore

Terrapin, Inc.
380 Green Street
Cambridge, MA 02139
$99.95

Requirements:
64K Apple II
Super IOB 1.5

Logo is an interpretative computer language which is becoming quite popular in classroom education. Terrapin Logo Version 2.00 is a state-of-the-art extension of the original M.I.T. version. All Logos allow the user to manipulate a "turtle" cursor (Terrapin uses a triangle) around the monitor screen. Version 2.00 includes not only an agile, multi-colored turtle, but also full-language features such as mathematical functions, musical commands, and word and list-handling capabilities. Terrapin's latest Logo seems to be a real trailblazer in the world of specialized computer languages.

The manual's sections on backups ($15 per disk after a short time), warranties (tough luck), liability (they're not liable) and copyright notices (if MIT doesn't own it, Terrapin does) blaze new trails for the English language too. One indigestible sentence tells you not to copy anything in the package, while another jawbreaker graciously allows you to copy the utilities disk and "the computer programs listed in the documentation", for personal and archival purposes only. Since the documentation mentions the Logo program, the language disk should be copyable, right?

Meet Catch $22, so to speak: Terrapin has heavily copy-protected the Language disk. Bit-copiers byte the dust. (E.D.D. III, Locksmith 5.0G, Nibbles Away II C2, Back It Up III, and Copy II Plus v5.3) Logo v2.00 is on some parameter lists (normal settings) but apparently that only means that someone got lucky once. Copy cards usually work well on a single-load program like this, but my Wildcard was unable to recreate a functional Logo.

Disk analysis reveals two Track 0 sectors filled with D2 marks, a hint that a nibble count occurs during the boot. At Track 0, Sector 8, data headers change from normal D5 AA CF to D4 AA CF. A disk "quick-scan", such as found on Locksmith 5.0 or Copy II Plus 5.3, shows that bit-insertion techniques are also used-particularly on Track $11, the catalog track. Sectoring looks normal, and a brief Applesoft prompt appears during the boot.

The normal sectoring and the brief prompt are clues that the Logo DOS uses an altered Read/Write Track/Sector format. The Super IOB program has a "swap controller" option that usually works nicely when you can capture an altered RWTS. It works fine for Logo-and with a few pointers from the IOB coach, you and your turtle can leave the copy-protection hare in the dust! The following procedure will completely deprotect the Language disk. You can then combine it with the Utilities disk or upload it to a hard disk at will.

The Language disk contains two binary or "B" files: SDOS (6 sectors) and STARTUP (138 sectors). The HELLO program is SDOS. SDOS checks for a 64K machine, displays a "Loading, please wait..." message, loads STARTUP, restores normal DOS, and finally jumps into STARTUP. At that point LOGO is on line (the starting line?). The LOGO RWTS is captured during the time that SDOS is loading STARTUP, but before normal DOS is restored. Loading 138 sectors takes a long time, so this is an easy big game hunt. After capturing and saving your trophy, the LOGO RWTS, the deprotection of the Language disk is duck (er, turtle) soup.

Here's how to put track shoes on your turtle, with the "Liberate LOGO" procedure that the manual forgot:

1) Boot the Language disk, and wait until the "program loading" message appears. At this point the LOGO RWTS is in use.

2) Press Reset or Control-Reset twice in quick succession-while the disk drive is running. The first Reset will cause an automatic reboot. The fast, second Reset halts the reboot, leaving the LOGO RWTS intact in memory.

3) Move the LOGO RWTS to the game preserve (sorry- "safe memory area") used by IOB:

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

4) Insert your IOB program disk, and boot it:

```
C600G
```

5) Save the LOGO RWTS on the IOB disk:

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

6) Remove the IOB disk, and prepare a blank disk by initializing it with normal DOS 3.3:

```
NEW
INIT HELLO
DELETE HELLO
```

7) Run the IOB program with the swap controller at the end of this article. This will read the LOGO language disk out onto your newly initialized disk. Do not use the "format disk?" option.

8) CATALOG your new disk to verify that you have two files, SDOS and STARTUP.

9) There is lots of room on this new disk for everything on the Utilities disk. Write protect the Utilities disk, and transfer the Utilities programs over by using a "copy files" program. (FID from the System Master works fine, but a "contiguous copier", such as the Copy II Plus "Copy Files" option, will lay the files down in a faster-reading order on your new disk.) Use the wildcard option by typing an equals sign, "=", when asked for "file name?".

10) Finish your disk by replacing the old HELLO program, which came across from the Utilities disk, with a new HELLO program to call SDOS:

```
UNLOCK HELLO
NEW
10 PRINT CHR$(4);"BRUN SDOS"
SAVE HELLO
```

11) Boot your new disk and you should find a COPYA turtle waiting for you!

Notes

Steps 1 and 2: If the IOB program can't read the Language disk, you perhaps waited too long before Resetting. Don't let the "?" Logo prompt come on.

Step 6: Make sure to initialize your new disk with a NORMAL DOS. If in doubt, it's worth the effort to boot your computer with your DOS 3.3 system master before typing "NEW" and initializing a blank disk. When SDOS restores normal DOS, a "speed DOS" such as Diversi-DOS gets scrambled. (If you want a turbocharged turtle, judicious surgery on the SDOS file may be needed.)

controller

```
1000 REM TERRAPIN LOGO CONTROLLER
1010 TK=3:LT=35:ST=15:LS=15:CD=WR:FAST
     =1
1020 GOSUB 360:GOSUB 490:GOSUB 610
1030 GOSUB 360:GOSUB 490:GOSUB 610:IF PEEK
     (TRK)=LT THEN 1050
1040 TK=PEEK(TRK):ST=PEEK(SCT):GOTO 1020
1050 HOME:PRINT "COPY^ DONE":END
10010 PRINT CHR$(4);"BLOAD^ LOGO.RWTS"
```

another look at...

Apple Logo II

by Bill Fogg

Apple Computer

Requirements:
64K Apple][Plus, //e or //c
(LOGO II requires 128K //e or //c)
A ProDOS disk with BASIC.SYSTEM
Copy program that can skip tracks
A blank disk or two

Back in COMPUTIST No. 27, page 8, was Tim Beckmann's softkey for Apple LOGO II. With great anticipation I tried it. However, my version would not budge. It would not be copied. The following softkey is for use on the ProDOS 1.1.1 version of LOGO II. If Tim's softkey didn't work for you, try this one.

Apple LOGO II

Apple LOGO II is an outstanding programming language. It is very user interactive and is an excellent vehicle to teach programming and the use of computers to children. However, it is copy protected. So if good old Murphy (you know, the one who said whatever can go wrong, will) happens to visit the kids one day, there goes your program disk. Fear not, the solution is easy.

The Controller

The controller for Super IOB 1.5 listed here is nearly the same as the one that Tim Beckmann presented. Lines 1030 and 1070 call the "R/W a range quickly" routine in Super IOB to read or write one track at a time. In lines 1050 and 1080 if the next track to be read in or written to is track 1 then LT will equal 1 and TK (the track to be read or written to on the next pass) will become 2, thus bypassing track one. It would be necessary to write a new RWTS to read in track one to use Super IOB to copy that. But why bother? Apple gave us a program to do that for us right on the Apple LOGO II Program Disk.

The Protection

The ProDOS version of Apple LOGO II has two and one half bytes (FF C FC) written between the data header (D5 AA AD) and the actual start of data on track one. The rest of the disk is standard ProDOS. The first eight blocks of data for the file "LOGO1" are located on track one, with the rest of the file filling tracks two through seven. The key block for "LOGO1" is the very first block of the file and is located on block seven. Normally the key block is the second block of a file with ProDOS filling the first block with data before deciding that the file needs a key-block.

The file "LOGO.SYSTEM" is the first file executed when the disk boots after loading ProDOS. LOGO.SYSTEM is merely a loader for the real LOGO Operating System, LOGO1. LOGO.SYSTEM reads in the file LOGO1 and jumps to the beginning of it. By slightly modifying LOGO.SYSTEM we can persuade it to load LOGO1 for us so we can save it to a normal disk.

Step by Step

1) First you must copy the Apple LOGO II Program disk with a copier that will ignore bad sectors or one that can be used to copy specified tracks. Copy track zero and tracks $02-$22 (2-34 decimal). You may use the controller below with Super IOB v1.5.

2) Boot a ProDOS disk containing BASIC.SYSTEM.

3) Put the "Apple LOGO II Program disk" in drive 1.

4) Set the ProDOS prefix to "LOGO".

```
PREFIX /LOGO
```

5) Bload LOGO.SYSTEM. The address and type parameters are necessary for loading "SYS" type files.

```
BLOAD
LOGO.SYSTEM,A$2000,TSYS
```

6) Enter the monitor.

```
CALL-151
```

7) Patch LOGO.SYSTEM so that instead of jumping to LOGO1, after it is loaded, it jumps into the monitor. A checksum must also be defeated.

```
2103:A9 00
2154:4C 59 FF
```

8) Start the loader, (LOGO.SYSTEM modified), to load LOGO1.

```
2000G
```

9) After your friendly Apple beeps and you see the monitor prompt in the bottom left corner of your screen and your drive stops, LOGO1 is loaded in memory from $2000 to $8FFF. Reconnect ProDOS. (The JMP $FF59 disconnects ProDOS).

```
BE00G
```

10) Put the copied disk in drive 1, UNLOCK, BSAVE and LOCK LOGO1. ProDOS takes care of the length but with "SYS" type files you must specify the address and file type (TSYS).

```
UNLOCK LOGO1
BSAVE LOGO1,A$2000,TSYS
LOCK LOGO1
```

11) Unlock LOGO.SYSTEM on the copy disk.

```
UNLOCK LOGO.SYSTEM
```

12) Load and modify LOGO.SYSTEM so that it will load and execute the unprotected version of LOGO1.

```
BLOAD
LOGO.SYSTEM,A$2000,TSYS
CALL -151
2103:A9 00
22E5:A9 00
```

13) Resave LOGO.SYSTEM to the disk and lock it. Again, ProDOS takes care of the length.

```
BSAVE
LOGO.SYSTEM,A$2000,TSYS
LOCK LOGO.SYSTEM
```

14) Make a COPYA copy of this disk for Murphy's Law, and you're finished!

controller

```
1000 REM LOGO!!.CONTROLLER
1010 TK = 0 :ST = 15 :LS = 15 :FAST = 1 :CD = WR
1020 T1 = TK : GOSUB 490
1030 LT = TK + 1 : GOSUB 610
1040 IF PEEK (BUF ) = MB THEN 1060
1050 TK = TK + 1 + (LT = 1 ) : IF TK < 35 THEN 1030
1060 TK = T1 : GOSUB 490
1070 LT = TK + 1 : GOSUB 610
1080 TK = TK + 1 + (LT = 1 ) : IF PEEK (BUF ) < MB
     AND TK < 35 THEN 1070
1090 IF TK < 35 THEN 1020
1100 HOME : PRINT "DONE^ WITH^ COPY" : END
```

controller checksums

```
1000  - $356B      1060  - $A320
1010  - $C83F      1070  - $7D7D
1020  - $1B67      1080  - $CDB2
1030  - $5F7E      1090  - $4C37
1040  - $3700      1100  - $E068
1050  - $26B6
```

Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies

softkey for...

Fishies

by Kevin Sartorelli

Fishies 1.Ø
Harper & Row

Requirements:
Super IOB 1.2
DOS 3.3 slave disk
Method to reset into the monitor

Fishies is an interesting way of having an aquarium when you don't have an aquarium. The program allows your computer screen to be the front of an aquarium complete with fish, a submarine, a mermaid and a frogman. The introduction is very long and I wanted to bypass it, but unfortunately the disk is protected. The protection scheme relies on altered DOS marks and having the main program on a very non-standard track (track 3). Below is a method of deprotecting this disk. It starts out by getting the main program into memory and capturing it, then it uses an IOB controller to copy the rest of the disk.

1) Boot the Fishies disk and when the introduction starts break into the monitor.

2) Modify the loader to exit to the monitor when finished.

BCE5:4C 59 FF

3) Execute the loader to load in the protected track (track 3).

BC56G

To check that it worked, the following code should be at $B15:

```
B15- A9 ØØ      LDA #$ØØ
B17- 8D 11 ØB   STA $ØB11
B1A- 8D 12 ØB   STA $ØB12
```

4) Move the just loaded code to a safe place.

1800<800.1400M

5) Capture the protected RWTS for Super IOB.

2800<B800.BFFFM

6) Boot a slave disk.

C6ØØG

7) Save the RWTS and the code from track 3 to disk.

BSAVE RWTS.FISHIES,A$2800,L$800
BSAVE TRACK3 CODE,A$1800,L$CØØ

8) Use the Super IOB 1.2 controller at the end of this article to copy the disk. You may also have to delete some of the following lines from the IOB program as the controller is a long one and you may run out of room (the following are all REMs): Lines 1Ø, 2Ø, 3Ø, 4Ø, 5Ø, 7Ø, 9Ø, 12Ø, 13Ø, 15Ø, 17Ø, 18Ø, 2ØØ, 22Ø, 23Ø, 24Ø, 25Ø, 26Ø, 27Ø, 28Ø, 3ØØ, 31Ø, 35Ø, 37Ø, 42Ø, 44Ø, 46Ø, 48Ø, 54Ø, 1ØØØØ, and 1Ø13Ø.

The controller performs a number of sector edits to change their DOS to read a normal disk and tell the program about the new track 3.

9) When it has finished copying the disk, load the previously saved TRACK3 CODE back into memory.

BLOAD TRACK3 CODE

10) Enter the monitor and type in the following code.

```
CALL -151
800:A9 08 A0 18 20 D9 03 B0
808:09 EE 21 08 CE 1D 08 10
810:EF 60 20 2D FF 4C 59 FF
818:01 60 01 00 03 0B 2A 08
820:00 18 00 00 02 00 00 60
828:01 00 01 EF D8 00
```

11) Insert your copy of FISHIES and type **8ØØG** to write the code onto track 3.

To avoid the long introduction try the following on your copy. Read track $12, sector 1 (their VTOC) and write this sector to track $11, sector Ø (which is unused by FISHIES). This will allow you to catalog the FISHIES disk. Boot a normal DOS 3.3 disk and LOAD Q[CTRL-Y]WERTY. Don't try to list this program

1.0

or it will destroy itself! Type **25Ø PRINT** then **Ø ONERR GOTO 25Ø**. This will trap any error, including [CTRL-C], and run the next program. Now type the following:

UNLOCK Q[CTRL-Y]WERTY
SAVE Q[CTRL-Y]WERTY
LOCK Q[CTRL-Y]WERTY

Now we need to modify the amount of screen that gets cleared to insure the whole screen is cleared as we don't know what will be on it when the program is interrupted. To do this type LOAD CONTINUE[CTRL-Y].

Type in the following lines.

```
6Ø HCOLOR= 4 : FOR Y = 2 TO 73 : HPLOT 2Ø ,Y TO 24Ø
   ,Y : NEXT Y
65 FOR Y = 74 TO 1Ø7 : HPLOT Ø ,Y TO 235 ,Y : NEXT Y
```

Now type the following:

UNLOCK CONTINUE[CTRL-Y]
SAVE CONTINUE[CTRL-Y]
LOCK CONTINUE[CTRL-Y]

Now when you wish to leave the introduction and enter FISHIES proper just type a [CTRL-C] and the introduction will stop and the main program will load.

controller

```
1ØØØ TK = Ø : ST = Ø : LT = 35 : CD = WR
1Ø1Ø T1 = TK : GOSUB 49Ø : POKE 47397 ,24 : POKE
     47398 ,96 : POKE 47496 ,24 : POKE 47497 ,96
1Ø2Ø GOSUB 43Ø : GOSUB 1ØØ : ST = ST + 1 : IF ST <
     1Ø THEN 1Ø2Ø : REM READ 'NORMAL' SECTORS
1Ø3Ø POKE 47397 ,188 : POKE 47398 ,14Ø : POKE
     47496 ,168 : POKE 47497 ,2Ø8
1Ø4Ø FOR I = 11623 TO 11651 : READ A : POKE I ,A
     : NEXT : REM PATCH CODE
1Ø5Ø A = 234 : FOR I = 11652 TO 11746 : POKE I ,A
     : NEXT : POKE 11747 ,169 : POKE 11748 ,154
1Ø6Ø POKE 1Ø748 ,173 : POKE 11Ø14 ,78 : POKE
     11178 ,17Ø : POKE 1194Ø ,78 : POKE 121Ø9 ,76
     : POKE 1211Ø ,7 : POKE 12111 ,191 : POKE
     12243 ,7Ø
1Ø7Ø POKE 12291 ,3 : POKE 12292 ,76 : POKE 12293
     ,72 : POKE 12294 ,19Ø : POKE 12295 ,134 :
     POKE 12296 ,72 : A = 7Ø
1Ø8Ø POKE 1231Ø ,A : POKE 12345 ,A : POKE 12373
     ,A : POKE 12375 ,A : POKE 12379 ,A : POKE
     12445 ,A : POKE 12451 ,A : POKE 12453 ,A
1Ø9Ø FOR I = 1Ø8Ø6 TO 1Ø817 : READ A : POKE I ,A
     : NEXT : FOR I = 1Ø898 TO 1Ø9Ø9 : READ A : POKE
     I ,A : NEXT : FOR I = 11113 TO 11156 : READ
     A : POKE I ,A : NEXT
11ØØ GOSUB 36Ø : ONERR GOTO 55Ø
111Ø GOSUB 43Ø : GOSUB 1ØØ : ST = ST + 1 : IF ST <
     DOS THEN 111Ø
112Ø IF BF THEN 115Ø
113Ø ST = Ø : TK = TK + 1 : IF TK = 3 THEN TK = TK +
     1 : GOTO 111Ø
114Ø IF TK < LT THEN 111Ø
115Ø GOSUB 49Ø : TK = T1 : ST = Ø : GOSUB 36Ø
116Ø GOSUB 43Ø : GOSUB 1ØØ : ST = ST + 1 : IF ST <
     DOS THEN 116Ø
117Ø ST = Ø : IF TK = 2 THEN TK = TK + 1
118Ø TK = TK + 1 : IF BF = Ø AND TK < LT THEN 116Ø
119Ø IF TK < LT THEN T1 = TK : GOSUB 49Ø : GOTO 11ØØ
12ØØ HOME : PRINT "EVERYTHING^ OK.^^DO^ REST^
     OF^ SOFTKEY.^" : END
1ØØ3Ø PRINT CHR$ (4) "BLOADRWTS. FISHIES
     ,A$19ØØ"
62Ø1Ø DATA 141 ,38 ,3 ,169 ,11 ,141 ,39 ,3 ,169
     ,8 ,141 ,43 ,3 ,169 ,Ø ,141 ,42 ,3 ,32 ,56
     ,3 ,238 ,43 ,3 ,2Ø6 ,39 ,3 ,16 ,245
62Ø2Ø DATA 2Ø8 ,1Ø ,234 ,189 ,14Ø ,192 ,16 ,251
     ,2Ø1 ,17Ø ,24Ø ,92 ,2Ø8 ,174 ,234 ,189 ,14Ø
     ,192 ,16 ,251 ,2Ø1 ,17Ø ,2Ø8 ,164
62Ø3Ø DATA 32 ,168 ,166 ,173 ,219 ,191 ,16 ,5
     ,169 ,2 ,32 ,141 ,177 ,169 ,Ø ,141 ,219 ,191
     ,141 ,218 ,191 ,173 ,186 ,179 ,76 ,239 ,157
     ,169 ,Ø ,141 ,237 ,157 ,141 ,238 ,157 ,141
     ,217 ,191 ,32 ,118 ,186 ,76 ,81 ,168
```

controller checksums

1ØØØ	- $2129	113Ø	- $BØ37
1Ø1Ø	- $56E7	114Ø	- $DØB8
1Ø2Ø	- $F29B	115Ø	- $CØ6B
1Ø3Ø	- $5ACF	116Ø	- $9C6D
1Ø4Ø	- $62D5	117Ø	- $84A2
1Ø5Ø	- $D2Ø5	118Ø	- $59CF
1Ø6Ø	- $F636	119Ø	- $A255
1Ø7Ø	- $3C34	12ØØ	- $Ø223
1Ø8Ø	- $8C75	1ØØ3Ø	- $C12A
1Ø9Ø	- $2EFF	62Ø1Ø	- $5227
11ØØ	- $EØ2Ø	62Ø2Ø	- $9Ø66
111Ø	- $DE21	62Ø3Ø	- $AAE9
112Ø	- $Ø411		

Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies
Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies Fishies
Fishies Fishies Fishies Fishies
Fishies Fishies Fishies
Fishies Fishies
Fishies

Infocom Revealed

By George and John Bigelow

Infocom text adventures are fascinating and often very difficult games. They are filled with situations where not only is the solution to a particular situation not obvious, but may require having made some crucial decision earlier in the game. In such situations the player can benefit immensely from clues. Infocom does provide a set of clues, but you are required to purchase them separately from the game. This is not particularly helpful when one is deep into the game on a Friday night! In addition, it somehow doesn't seem right to pay for the game, then have to pay additionally for help (however, Infocom doesn't seem to mind!).

For many text adventures, there is a faster and cheaper way to get clues when stuck: you merely fire up a sector reader such as Copy][Plus, and snoop through the disk for any text contained within the game's files. For example, the files of the Ultima series and Transylvania can be read this way.

Since these files are generally organized as a set of disconnected sentences and sentence fragments, they do not give the game away or tell one what to do in a particular situation. They do, however, provide clues as to what objects, actions, and situations are important in the game. Often this is enough to get you moving again when stuck in a tough situation.

Unfortunately, this technique does not work with Infocom! Those who have tried it have found very little (if any) in the way of text on any Infocom disk. Where do all the words for the game come from? The answer is that Infocom uses a two-for-three coding technique in order to store its text more efficiently.

The essential idea is this: a byte of information contains 256 different possibilities. However, text needs only a small part of these possibilities, say 26 letters plus a few others. It is possible, therefore, to recode a string of characters so as to make it fit into less bytes. In Infocom's case, three characters are recoded so as to fit within two bytes.

Figuring Out Infocom's Approach

How does Infocom do this? The following steps parallel how we went about discovering this. You may be interested in trying them out on your own. Initial work was done with the Macintosh version of Hitchhiker's Guide to the Galaxy (Hitchhiker). Later work was done on the Apple version.

1) We figured that Infocom wasn't going to do anything tricky. After all, the purpose of coding the text was to conserve space, not to hide it from players. We started with this idea: a byte is 8 bits of information. Two bytes together make 16 bits. If those bits were divided into three parts, it's possible to make three five-bit nibbles, with one bit left over.

A five bit nibble can represent two-to-the-fifth, or 32 possibilities. This should be enough possibilities for one byte of text (assuming one doesn't use numerals).

2) We wrote a program which reads files and prints them out in binary format (the final evolution of this program will be described later). We found sections, none very long, where every other byte began with "0". We figured that these might be the start of two byte sequences, in which byte pairs would translate into three characters.

3) We instructed the program to look for beginning zeroes, to combine byte pairs into 16 bit strings, and to divide the string into an initial zero bit, plus three five-bit strings. The program then translated each of these strings into a number (ranging, of course, between 0 and 31).

4) We then tried the logical thing, assigning an "A" to 0, "B" to 1, and so forth. The result was garbage! We then did a frequency count. We noticed, with quickening pulse, that if we assigned the alphabet starting with the sixth number rather than the first, that the frequencies were what we would expect of statistical distributions of letters; e.g. lots of vowels, not so many Q's, X's and so forth. We then instructed the program to make these assignments.

5) Bingo! The resulting translation results in strings of sense, separated by strings of garbage. In studying what we found, we figured out that some of that garbage is coded words. For example, "<?" is "and", "=F" is "but", and ">U" is "between". We surmise that Infocom reduces the size of its text in two passes. In the first pass, the entire text is searched for commonly used string segments which are translated into shorter character sequences. In the second pass, every three characters is translated into two bytes. We have not been able to figure out how to tell which byte is the start of a two byte sequence. Printouts show that the program will be on track for a while, then lapse into garbage, then make sense again.

Our hunch is that there is an address table somewhere that tells where to start sequences. Lacking knowledge of that table, our approach is to decode each block of bytes twice, starting first with the first byte in the sequence, then decoding again starting with the second byte. One of the two is accurate for every part of the code.

A Program to Decode Infocom Text

In this section, we describe a program, "Infocom Text Reader". It is an Applesoft BASIC program (plus a little machine language to speed things up a bit) developed on an Apple][Plus which will read and decode any text on an Infocom disk (with some reservations about the very latest releases, we hear that they have changed something), and thereby give you a deeper understanding of the Infocom game. It starts by reading one sector from the disk into memory. It then calls a machine language subroutine which combines each pair of hexadecimal numbers into a sixteen digit binary number, divides that number into three binary numbers (ignoring the MSB) and converts them into letters and symbols. It translates and prints the sector twice, starting with even byte pairs, then shifting one and repeating the process.

Keying it In

Type in the Applesoft portion of Infocom Text Reader and save it with:

SAVE INFOCOM TEXT READER

Next, using the instructions on the inside front cover of this magazine, key in the hexdump and save it on the same disk with:

BSAVE OBJ.READER,A$300,L$43

You're now ready to do some serious snooping!

Using Infocom Text Reader

Enter the program "Infocom Code Reader" and save it to your disk. When the program is run you will be asked to select a track. Remove your disk and put in the Infocom disk you wish to research. Select a track, then sector. The heading called DOS is automatic and will take care of itself. A few things that are not obvious that should be mentioned: type in Q whenever TRACK is displayed to quit the program (normal DOS is also reinstated). Typing + or -, will select the next track or sector and next lower track or sector respectively. Just typing return will select the same track or sector. One thing to keep in mind, the next sector is not the numerically next sector! Infocom sectors skip around some, and + and - take that into account. For those who care, the sector order is:

0, 13, 11, 9, 7, 5, 3, 1,
14, 12, 10, 8, 6, 4, 2, 15

To print the screen on paper select "Y" when asked if you want to print the screen. You will then be asked if you want single or double spacing. Turn on your printer and select either single or double spacing and away you go!

Why Two Passes?

Say we have the numbers 1, 2, 3, 4, 5 to make pairs of any two numbers that are next to each other. The possibilities are 1-2, 3-4, and skip 5 or 2-3, 4-5 and skip 1. Either of these combinations of hexadecimal bytes as read off an Infocom disk may have meaningful information in them. Infocom seems to have no set way (by sector, anyway) to combine the hex bytes. So to retrieve as much information as possible, we go back and make Pass 2, skipping the first and last hex numbers (we may lose up to four letters that way, but we can't think of a better way without getting too complicated). The screen will show 10 lines (pass 1) then 10 more lines (pass 2). Somewhere in both passes is all the text you are looking for!

Abbreviations

The latest disk we have is Hitchhiker, and the program works for it and all previous disks (such as the Zorks, Planetfall, Enchanter and Sorcerer). Each disk has a list of all acceptable words that can be typed in (of course only out to six letters). On most of these disks the list may be found on track 7, and on some track 6. These are several sectors long, so once you have found it use <RETURN> for track and + for sector.

The words are evenly divided between Pass 1 and Pass 2. We suggest printing the screens and using Hi-Liters.

The words that the program prints (THERE IS A VIOLENT EXPLOSION, and so forth) are scattered all around, generally from track 4 on. In reading these you get a sense of frequent garbage. For instance, in Hitchhiker you might see this:

```
?WHEN;=J<>COME;<F<BSENSES;<?SOLVE;<RPUZ
ZLE@P
```

This brings us to abbreviations! Infocom has a list of short forms of words, to further save space. For instance, "=J" means "will", "<>" is "you". The question mark capitalizes the following letter, and the "@P" is a question mark. The translated statement will read "When will you come to your senses and solve the puzzle?". To find the list of abbreviations look on track 3 sectors 0,13, and on some 11 (the 13 and 11 are automatically selected from sector 0 by using + when selecting the next sector).

But just a list of the words is no help, you have to assign the abbreviation to the word. An abbreviation is always two symbols long and start with one of these three characters: "<, =, >".

The second character of an abbreviation is (in order); ";<=>?@ABC ... Z". Therefore, to get a list of the abbreviations and the words they are associated with, do this: print out track 3 sectors 0 and 13 double spaced. You should see a lot of semicolons, then a list of words. Starting with the first word write above it "<;" and above the second "<<" and so forth (<=, <>, <@, <A and so on). After the word designated by "<Z" start with "=;" through "=Z", then ">;" through ">Z". In some games the list is quite short and you will not use all the symbols.

In Hitchhiker "<;" is "the", "<<" is a comma, "<=" is "You" and "<?" is "and". These are different on every disk, so this must be done for every title you try. Sorry about that!

Some other things you must know: ";" is a space, "@M" is a period, "@O" is an exclamation point, "@P" is question mark, "@S" is an apostrophe, "@T" is a quote mark, "@W" is a dash, "@X" is a colon, "@Y" is a left parenthesis and "@Z" is a right parenthesis. These are all we have figured out, the rest of the letters must mean something, perhaps numbers.

Zork II is different in that the = capitalizes the next letter and ">S" is an apostrophe and there are probably more differences. Most of our investigations have been in Hitchhiker.

Final Notes

This program will not answer all your questions, and indeed may raise several, but the list of allowable words alone is worth a lot if you are stuck. Here's something to mull over: in Hitchhiker on track 18 sector 10 Prosser has a game manual. And we'll let you take it from there!

Infocom Text Reader

```
10 REM *----------------------*
20 REM *  INFOCOM TEXT READER *
30 REM *                      *
40 REM *   BY:                *
50 REM *    GEORGE BIGELOW    *
60 REM *    & JOHN BIGELOW    *
70 REM *                      *
80 REM *   ENHANCED BY:       *
90 REM *    RAY DARRAH        *
100 REM *----------------------*
110 REM
120 TEXT : HOME
130 IF PEEK (768 ) < > 169 THEN PRINT CHR$ (4 )
    "BLOAD^ OBJ.READER"
140 REM SET UP VARIABLES
150 COMDOS = 188 :STDDOS = 173 : POKE 47356
    ,STDDOS :DOS$ = "STANDARD"
160 REM MAKE SELECTIONS
170 HOME : HTAB 10 : PRINT "INFOCOM^ TEXT^
    READER"
180 VTAB 2 : HTAB 9 : PRINT T
190 VTAB 2 : HTAB 3 : INPUT "TRACK:" ;T$ : IF T$
    < > "Q" AND T$ < > "" AND T$ < > "+" AND T$ <
    > "-" THEN T = VAL (T$ )
200 IF T$ = "Q" THEN POKE 47356 ,STDDOS : HOME :
    END
210 IF T < 0 OR T > 34 THEN T = 0 : GOTO 170
220 GOSUB 420
230 VTAB 2 : HTAB 21 : PRINT S
240 VTAB 2 : HTAB 14 : INPUT "SECTOR:" ;S$ : IF
    S$ < > "+" AND S$ < > "-" AND S$ < > "" THEN
    S = VAL (S$ )
250 IF S < 0 OR S > 15 THEN VTAB 2 : HTAB 21 : PRINT
    "^ ^ " : GOTO 240
260 GOSUB 500
270 VTAB 2 : HTAB 26 : PRINT "DOS:" ;DOS$
280 REM  CALL RWTS
290 POKE 47083 ,0 : POKE 47084 ,T : POKE 47085 ,S
    : POKE 47088 ,0 : POKE 47089 ,150 : POKE
    47092 ,1
300 CALL 768
310 REM AUTO DOS SELECTION
320 ERR = PEEK (47093 ) : IF ERR < > 64 THEN 380
330 DOS = PEEK (47356 )
340 IF DOS = STDDOS THEN POKE 47356 ,COMDOS :
    VTAB 2 : HTAB 30 :DOS$ = "INFOCOM^ " : PRINT
    DOS$ : GOTO 360
350 POKE 47356 ,STDDOS : VTAB 2 : HTAB 30 :DOS$
    = "STANDARD" : PRINT DOS$
360 CALL 768
370 REM PRINT DECODED SECTOR
```

```
380 POKE 776 ,0 : CALL 775 : PRINT : POKE 776 ,1
    : CALL 775
390 POKE - 16368 ,0 : PRINT : VTAB 24 : PRINT
    "PRINT^ THIS^ PAGE?^ (Y,N) :" ; : GET A$ : IF
    A$ = "Y" THEN 610
400 GOTO 170
410 REM PRINT TRACK
420 IF T$ = "" THEN 470
430 IF T$ = "+" THEN T = T + 1 : GOTO 450
440 IF T$ = "-" THEN T = T - 1
450 IF T < 0 THEN T = 34
460 IF T > 34 THEN T = 0
470 VTAB 2 : HTAB 9 : PRINT T; "^ "
480 RETURN
490 REM PRINT SECTOR
500 IF S$ = "" THEN 580
510 IF S$ = "+" THEN S = S + 13
520 IF S = 28 THEN S = 0 : GOTO 580
530 IF S > 15 THEN S = S - 15
540 IF S$ = "+" THEN 580
550 IF S$ = "-" THEN S = S - 13
560 IF S = - 13 THEN S = 15 : GOTO 580
570 IF S < 0 THEN S = S + 15
580 VTAB 2 : HTAB 21 : PRINT S; "^ "
590 RETURN
600 REM PRINT PAGE
610 VTAB 24 : HTAB 7 : PRINT "DOUBLE^ SPACE?^
    (Y/N) :" ; : GET A$
620 PR# 1
630 FOR Y = 1 TO 22
640 FOR X = 1 TO 40
650 VTAB Y : HTAB X
660 XY = SCRN( X - 1 ,2 * (Y - 1 ) ) + 16 * SCRN(
    X - 1 ,2 * (Y - 1 ) + 1 )
670 PRINT CHR$ (XY ) ;
680 NEXT : VTAB 1 : PRINT
690 IF A$ = "Y" THEN PRINT
700 NEXT
710 VTAB 1 : PRINT
720 PR# 0
730 VTAB 24 : HTAB 1 : PRINT "PRESS^ A^ KEY^ TO^
    CONTINUE-->" ; : GET A$
740 GOTO 170
```

Infocom Text Reader Source Code

```
*-------------------------------------------------------------------*
*                Infocom Text Reader Machine Routines               *
*                          By Ray Darrah                            *
*-------------------------------------------------------------------*

          .OR $300
          .TF OBJ.READER

9600-  BUFFER  .EQ $9600
FF-    CHAR1   .EQ $FF      FIRST CHARACTER AT $FF
FE-    CHAR2   .EQ $FE      SECOND CHARACTER AT $FE
FD-    CHAR3   .EQ $FD      FINAL CHARACTER AT $FD
FDF0-  COUT    .EQ $FDF0    PRINT A BYTE

0300: A9 B7           LDA #$B7     SET UP FOR RWTS CALL
0302: A0 E8           LDY #$E8
0304: 4C D9 03        JMP $3D9     CALL RWTS AND RETURN

0307: A0 00           LDY #0       0 FIRST TIME, 1 NEXT TIME
0309: B9 00 96  GET1  LDA BUFFER,Y
030C: 29 7F           AND #$7F     STRIP OFF MSB
030E: 4A              LSR
030F: 4A              LSR
0310: 85 FF           STA CHAR1    GOT FIRST CHAR
0312: B9 00 96        LDA BUFFER,Y
0315: 29 03           AND #$3      ONLY BITS 0 AND 1
0317: 0A              ASL
0318: 0A              ASL
0319: 0A              ASL
031A: 85 FE           STA CHAR2    GOT BITS 3 AND 4
031C: C8              INY
031D: F0 23           BEQ RTS1     IF DONE, RETURN WITHOUT PRINTING
031F: B9 00 96        LDA BUFFER,Y
0322: 4A              LSR
0323: 4A              LSR
0324: 4A              LSR
0325: 4A              LSR
0326: 4A              LSR
0327: 05 FE           ORA CHAR2    MIX BITS 0 - 2 WITH 3 AND 4
0329: 85 FE           STA CHAR2    GOT SECOND CHAR
032B: B9 00 96        LDA BUFFER,Y
032E: 29 1F           AND #$1F     BITS 0 THROUGH 4 PLEASE
0330: 85 FD           STA CHAR3    GOT THIRD CHAR

0332: A2 02           LDX #2       PRINT THREE CHARACTERS
0334: B5 FD     PRINT1 LDA CHAR3,X GET A CHARACTER
0336: 18              CLC
0337: 69 BB           ADC #187     ADD 187 TO GET ASCII
0339: 20 F0 FD        JSR COUT     PRINT IT
033C: CA              DEX
033D: 10 F5           BPL PRINT1
033F: C8              INY
0340: D0 C7           BNE GET1
0342: 60        RTS1  RTS
```

Hexdump

```
0300: A9 B7 A0 E8 4C D9 03 A0    $E152
0308: 00 B9 00 96 29 7F 4A 4A    $1C10
0310: 85 FF B9 00 96 29 03 0A    $AE19
0318: 0A 0A 85 FE C8 F0 23 B9    $9F39
0320: 00 96 4A 4A 4A 4A 4A 05    $0D88
0328: FE 85 FE B9 00 96 29 1F    $E073
0330: 85 FD A2 02 B5 FD 18 69    $4A04
0338: BB 20 F0 FD CA 10 F5 C8    $4F8A
0340: D0 C7 60                   $9758
```

checksums

```
10  - $BADD     380 - $B7AC
20  - $9B13     390 - $3527
30  - $4D3B     400 - $8F61
40  - $AD92     410 - $45D0
50  - $C899     420 - $A85F
60  - $FF65     430 - $2EA9
70  - $A3BF     440 - $A53D
80  - $A900     450 - $4828
90  - $924D     460 - $41BB
100 - $CB63     470 - $05A2
110 - $BD13     480 - $31BC
120 - $962A     490 - $DCF4
130 - $14D3     500 - $D65D
140 - $BD65     510 - $19D0
150 - $8A62     520 - $1D67
160 - $88AF     530 - $0587
170 - $7F32     540 - $7811
180 - $8E60     550 - $B96E
190 - $129A     560 - $26D3
200 - $64F3     570 - $1E3F
210 - $5B48     580 - $3541
220 - $90BA     590 - $6182
230 - $EA43     600 - $55A6
240 - $C0F8     610 - $67A4
250 - $F78B     620 - $4D2B
260 - $A3CD     630 - $D37D
270 - $33DC     640 - $209A
280 - $E4A3     650 - $6A90
290 - $D49B     660 - $5DB5
300 - $8DE4     670 - $DDA2
310 - $74DE     680 - $41DD
320 - $732D     690 - $57A9
330 - $D77F     700 - $C23C
340 - $DD0F     710 - $386E
350 - $75C8     720 - $09C1
360 - $38AD     730 - $E920
370 - $A187     740 - $501B
```

More ROM Running

By Wes Felty

COMPUTIST has had some very good articles on modifying ROMs in the past, especially COMPUTIST No. 19. The method of using switchable dual ROMs presented in that issue eliminated any concerns I had about incompatibilities between software and modified ROMs. With the switchable ROM system, you can always switch on a normal ROM if any problem shows up with your modified one.

While COMPUTIST's articles had very good patches, if you don't know assembly language, you were left in the dark as to why what was placed where. In this article, I intend to expand on some of the ideas to help you develop your

own modified ROMs further, fix and improve some of the][Plus routines, give some information about the Don Lancaster patch (explained later) for the //e and //c computers, and offer some improvements on the Lancaster patch.

Let's Start at the Beginning

First of all, lets look at EPROMs in general. The 2716 and 2732 EPROMs are not directly compatible with the Apple][Plus but the 2764 and 27128 EPROMS *are* compatible with the Apple //e and //c's. These EPROMs can be directly substituted with no hardware modification needed.

However, you should be careful when buying EPROM's. I recently bought a number of 2732's that were rated at 350 nanoseconds (ns) and none of them would work in my Apple][Plus. I went to another place and bought ones rated at 250 ns and they worked great. Every batch that I buy is cheaper than the last, but I've learned to watch the speed rating and to not buy too many of a batch until I have checked them out.

Number Crunching

What does the "16" in 2716 mean anyway? Doesn't it mean 16K? Yes and no. A 2716 EPROM holds 2K bytes. So, why don't the numbers agree? Because every byte is made up of 8 bits, the 2716 holds 16K bits which translates into 2K bytes (16K / 8 = 2K).

If this is true, then, why does the data to fill a 2716 take up 10 disk sectors and the data to fill a 2732 take up 18 disk sectors instead of 8 and 16 sectors respectively? Remember that for a DOS 3.3 file, one extra sector is required for a track/sector list. Additionally, binary files are stored on disk with the address and length of the file preceeding the actual data, so these extra four bytes push the data over onto an extra sector that holds just four bytes. Therefore the full binary files each require two additional sectors as overhead.

Using a Language Card for Testing

There is a much easier way to test your routines as you develop them than burning them into an EPROM, testing them, erasing them and reburning them which can be particularly annoying if you don't have an EPROM programmer.

The Language or RAM card (they are one and the same) can be used for testing purposes since it occupies the same memory positions as ROM. The RAM card, which is built into //e's and //c's, can be bank switched so that a machine language program can be loaded into RAM at high ROM addresses. For obvious reasons, these procedures need to be done with standard DOS 3.3 and not a version of it that is moved to the Language Card. ProDOS can't be used, since it occupies the Language Card.

First National Apple Switch Bank

The idea of bank switching is a very important one to understand in order to deal with today's hardware and software. Bank switching refers to the fact that the computer can have more than one bank of information stored at the same memory addresses but at different physical locations.

It is similar to having two or more different books on the same shelf of a bookcase. These books do not have the same information on the same page numbers, but they do have identical page numbers. You can read information from only one book at a time, but you can switch from book to book. This is similar to bank switching. The computer can be told to read the information at location $FDED from the mother board's ROM, the Language card RAM in slot 0, or a ROM or RAM card in any of the other slots. Furthermore, if one of the books is a notebook, then you can write to it as well as read from it on any given page. You aren't allowed, however, to write in the original book. ROMs do not have the hardware to be written to. That's why they are called Read Only Memory chips.

Moving Right Along...

Next, you need to understand the soft switches that are used to switch from one bank of information to another. These soft switches are special memory locations that act like switches that get thrown whenever they are accessed. So a PEEK, POKE, LDA, STA, BIT, etc. to one of these special memory locations essentially "flips" that switch on and all related switches off. If you are familiar with the screen soft switches which switch from text to lo-res or hi-res screens, then you should pretty well understand how the language card soft switches.

In flipping these switches, you need to remember that while the RAM card and ROM memory occupy essentially the same addresses in the computer's memory, they cannot both be active at the same time (just as hi-res page 1 and hi-res page 2 cannot be active simultaneously). Therefore, you must flip the soft switches back and forth to read or write enable RAM when you want to use it and read enable ROM when you need it.

There are soft switches for every possible combination of reading ROM or reading RAM and at the same time being able to write to RAM or not. Just because you wish to read from RAM doesn't mean that you want to be able to write to it, so there is a write protect RAM feature to prevent any changes being written there.

The soft switches are arranged as follows:

Table 1:

RAM Card Soft Switches (Slot 0)
For Slot n: Address = $C0(n+8)X

Addr	Bank	Read	Write	RAM Protected?
$C080	1	RAM	N/A	YES
$C081	1	ROM	RAM	NO
$C082	1	ROM	N/A	YES
$C083	1	RAM	RAM	NO
$C088	2	RAM	N/A	YES
$C089	2	ROM	RAM	NO
$C08A	2	ROM	N/A	YES
$C08B	2	RAM	RAM	NO

By flipping $C082, you can only read from ROM. This is the default situation.

By flipping $C081, you can read from ROM and write to RAM at the same addresses.

By flipping $C080, you can read from RAM but not be able to write to it.

One important difference between the reading and writing switches is that the write soft switches take two accesses to enable them.

Can you see now how fouled up a computer could get if a program that used the RAM Card ended without setting the soft switches properly? This is why many programs have a "Quit" option or tell you to reboot the computer to run another program.

When your Apple][Plus "dies" and you can't even get control with Reset, it is probably up in the $F8 page of RAM trying to run the Reset routine that is actually over in the $F8 bank of ROM.

Turning off the Language Card can be done by referencing location $C082, location $CFFF which is the soft switch to disconnect all peripheral cards, or a Reset on //e's and //c's.

Many programs that use the RAM card copy the $F8 ROM over into the $F8 RAM so they don't have to bank switch back and forth. Some protected software programs simply do not use the normal ROM routines. After loading they leave the $F8 RAM switched on at all times with their own $F8 programs. Just use the COMPUTIST modified $F8 ROMs to try to Reset out of AppleWriter, Homeword, the Flight Simulator, or Skyfox and see how far you get. On Reset, the hardware goes to $FFFC for the address of the Reset routine to use, but on][Plus's, it goes to $FFFC in *whatever bank is switched on* at the time.

Experimenting With the Card

Here are some exercises to get you used to soft switch flipping.

Note: When following these exercises, the disk drive may unpredictably turn on and garbage any disk in the drive.

a) Boot the DOS 3.3 master so that it loads Integer BASIC into the language card and then *remove the disk.*

b) Drop into the Monitor and disassemble the code at $E000. This is the start of Applesoft BASIC.

```
CALL -151
E000L
```

c) Now use table 1 to figure out which soft switch to flip to read enable the ROM and write protect the RAM on the language card. Did you come up with $C080? To activate this switch from the Monitor, you can type "C080" and return. This instructs the Monitor to read the contents of that address and therefore flips the switch. Try this now.

```
C080
```

d) If you disassemble $E000 again you should find different code there. This is Integer BASIC from the Language card.

```
E000L
```

You can start up either one of these BASICs with an E000G from the Monitor.

e) Can you switch back from RAM to ROM by using table 1? To read enable ROM, enter "C082" and return.

```
C082
```

f) Now try writing an $EA to location $E000.

```
E000:EA
```

g) Did it work? Disassemble it to check it.

```
E000L
```

h) Of course it didn't work. You can't write to ROM. So switch back to RAM and try it again.

```
C080
E000:EA
E000L
```

i) Were you surprised? Why didn't the "EA" write to RAM? It's because location $C080 read enables but write protects the RAM card. You could use "C081" which would allow you to write to RAM but you wouldn't see what you had done since you would be reading the ROM.

j) All this brings us to location $C083. This is the one that read and write enables the RAM card. Try the following:

```
C083
C083
E000:EA
E000L
```

Finally, you have made a change to the language card. Play around with different settings and try to answer to yourself why different things happen in different settings.

k) When you are done playing with this, turn off the computer because you may have a real mess inside the language card.

The Preparation

Here is what we have to do to use the language card to test our new ROM routines. We will have to bank switch the ROM memory off and read and write enable the language card RAM. Notice that this requires us to access the soft switch at $C083 twice.

In more detail, we need to copy the $F8 ROM into the language card RAM, read and write enable the language card RAM and finally transfer our new routines into the language card RAM card where they can be executed.

Actually Using the RAM Card

First boot the DOS 3.3 system master or some other disk that loads Integer BASIC into the language card. We won't be using Integer BASIC, but this process initializes the Language Card for us. If you want an image of the autostart $F8 ROM in the RAM card then instead of loading Integer BASIC, you can use the following commands:

```
C081
C081
F800<F800.FFFFM
```

The first two commands write enable the language card RAM while still using the $F8 ROM for all reads. The next command reads every byte in the $F8 ROM and stores them in the same address of language card RAM.

Next, we need to read and write enable the RAM card. To do this we need to flip the soft switch $C083 twice. Therefore, enter this simple routine from the Monitor:

```
300:2C 83 C0 2C 83 C0 60
300G
```

If the RAM card later gets disconnected, just type "300G" from the Monitor or "CALL 768" from BASIC to reconnect it.

You can now read, write, modify, and run the routines from the language card as if they were in ROM.

Checksums

One type of diagnosis that the Apple computer can do to itself or programs can do to it is calculating a checksum. It is very simple for the computer or a program to check out a ROM by adding up the values of all of its bytes and comparing this sum to a known value for that particular ROM. Why do it? The Apple //e's self diagnosis (closed apple Reset keys) can determine easily if a chip has gone bad or a program can determine if you have changed the $F8 ROM in order to gain control of the Reset key back from them.

Fortunately, it isn't too hard to deal with a ROM's checksum once you are aware of its existence. Program 1 below, "Checksummer", uses a routine similar to the one built into //e's and //c's to find the checksum of a ROM, just part of a ROM, or a routine in memory that is going to end up in a ROM. To tell the program what locations to checksum, you have to change the FOR NEXT statement in line 10. It is shown set up for the entire $F8 ROM of an Apple][Plus, starting at $F800 (63488) and ending at $FFFF (65535). For an Apple //e or //c, you need to checksum their whole ROM, $C100-FFFF. Don't checksum locations $C000-$C0FF. Doing so flips soft switches and your computer will just go crazy when you try it.

The following is an example of how you can use this program to alter a ROM and still end up with the same checksum as before the alteration.

Lets say you have moved an image of the $F8 ROM to $1000 and plan to modify the 12 bytes of it that start at $FCC9 on the original ROM. This code would start at $1CC9 in the image of the ROM.

Before making the changes to the RAM copy of the ROM, change line 10 of checksummer to "FOR X = 5321 TO 5340", RUN the program and write down the checksum value it comes up with.

Notice that although we used the correct starting address (5321 = $1CC9), I chose an ending address that is a number of bytes beyond the end of the impending changes. I did this to allow room for bytes to be changed beyond the code changes. This is necessary to balance the checksums. You should always allow more room for your routine than the length of it so you can match the checksum. Any bytes that remain unchanged just subtract out later, so it is no problem to have checksummer go farther than the actual changed bytes.

Now make the changes to the ROM and add a couple of zero bytes directly after the changes. Run the checksummer program again and subtract the two checksum values. If the final checksum was larger than the original one, you need to add more zero bytes following your changes.

Finally, convert the difference in checksums to a hex value and replace one of the zeros with it. If the difference in checksums is more than 255, then change one zero to $FF and change the following byte to the difference minus 255.

You should RUN the checksummer program after you are done to confirm that the modified code has the same checksum as the original code.

The Don Lancaster "Old ROM" Absolute Reset Package

There have been some questions raised about the Don Lancaster absolute "old Monitor" Reset seen advertised in various magazines. It is a really good buy for the Apple //e or //c. It comes with a tremendous amount of documentation, including much detail on how EPROMs are written to and read from and how to build adapters for burning 27128's and 2764's on a 2732 EPROM burner. It also has patches for the //c and the enhanced //e's. The programs and patches are supplied on disk, so you don't have to type them in. And the patched ROM's checksum is fixed so that the built in diagnostics shouldn't be able to detect that the ROM has been modified. This Reset kit is available by contacting:

Don Lancaster
Synergetics
746 First St. POB 809
Thatcher, Arizona 85552
(602) 428-4073

Because Lancaster's absolute Reset is designed for //e's and //c's where COMPUTIST absolute Resets have been designed for Apple]['s, they are quite different. Lancaster's patch resides in the CD ROM (//e & //c ROMs are 2764's and hold four times as much code as the]['s 2716) and is nowhere near the $F8 code.

Don Lancaster didn't have too much trouble deciding what routine to sacrifice as room for his patch. He placed it right on top of a routine that is designed to blast holes in any program in memory during a cold Reset (since the //e can do a cold boot without turning off the power (open apple Reset)). Therefore, with the

Synergetics absolute Reset you lose nothing and gain the ability to Reset out of any program directly into the Monitor, all for $19.50 and the price of a 2764 EPROM.

Lancaster's Faults

I am very picky and found two things with the Lancaster Reset that I felt needed changing, so I'll give here my patch to his patch.

First of all, his jump into the Monitor leaves some soft switches used for bank switching scrambled. Lancaster explains this in the documentation and tells you how to manually correct for it. Secondly, it leaves the power up byte scrambled, so a subsequent Reset will cold boot the machine.

You can fix each of these problems yourself while in the Monitor by entering "C006:00" to fix the soft switches and "FB6FG" to fix the power up byte.

Well, I don't agree with this approach. I believe that that the computer is supposed to take care of mundane details like these. I therefore developed a patch to Lancaster's patch which causes it to do these steps automatically. But first, why didn't he do this himself?

The answer is simple: he ran out of room. The hole blaster routine that he covered with his patch was too small for anything fancy and he didn't wish to sacrifice any other routines. He felt that the cassette routines were important since if you ever blew DOS, you could still recover your data on tape. He didn't realize that even if you obliterated the tape load routine, you could still save to tape and later use a tape load routine from disk to read the information back into the computer. Or better yet, you could use the dual ROM idea of COMPUTIST No. 19.

Using table 2, address $D8C9 seems like the best place to put a new routine since you will probably want to keep the new patch on the same chip with the first patch. Since you will be wiping out a routine (the cassette load routine), it is a good idea to place some kind of warning message there. I usually use a jump to $FF2D which rings the bell and prints "ERR". Therefore the new patch will actually start at $B8CC.

Table 2:

Possibly Expendable Subroutines

```
$D8B0 - D8C8   Save to cassette tape
$D8C9 - D8EF   Load from cassette tape
$D8F0 - D900   Get length for tape save
$D901 - D93D   Set up for cassette save

$FCC9 - FCD5   Writes tone header on tape
$FCD6 - FCEB   Write a bit to tape
$FCEC - FCF9   Read bits from tape
$FCFA - FD0B   Sense cassette tape

$FECD - FEFC   Write data to cassette tape
$FEFD - FF01   Read data from cassette
$FF02 - FF2C   Read hi-res from tape

$F800 - F818   Plot a lo-res point at X,Y
$F819 - F825   Draw lo-res horizontal line
$F826 - F830   Draw lo-res vertical line
```

Now that we have decided to sacrifice a routine, we have enough room for more embellishments. Why not change the Reset vector ($3F2) to jump into the Monitor on future Resets? And also, wouldn't it be nice to know where the Reset vector had been pointing? For some programs this is a soft re-entry point. For others, a whip memory routine with cold boot. Either way, it is usually a source of valuable information, so let's save the previous Reset vector before installing the new one to the Monitor. We will do this at $2FE and $2FF, the end of the keyboard input buffer.

Patching the Patch

There are two steps to fixing the Lancaster absolute Reset patch. The first is to change the jump into the Monitor (JMP $FF59) with a jump to the new patch (JMP $D8CC), fixing its checksum as we change the jump instruction. The second part is to write in the new patch routine. This patch will eventually be at $C25C so if you have the Lancaster "grabbed" CD ROM code starting at $1000, then the patch will be at $125C. Here is the exact procedure:

1) Use Lancaster's instructions all the way up to but not including burning the EPROM.

2) Load the C ROM.

BLOAD IIEMON.C,A$1000

3) Load the D ROM.

BLOAD IIEMON.D,A$2000

4) From the Monitor, change Lancaster's jump and his checksum fixer byte.

125C:CC D8 31

5) Enter the leftmost portion of the following disassembly. This is our new patch.

```
28C9:4C 2D FF ! JMP $FF2D  tape load=err
28CC:8D 06 C0 ! STA $C006  flip switch
28CF:AD F2 03 ! LDA $03F2  save old Reset
28D2:8D FE 02 ! STA $02FE  vector at $2FE
28D5:AD F3 03 ! LDA $03F3  & $2FF
28D8:8D FF 02 ! STA $02FF
28DB:A9 65    ! LDA #$65   set Reset
28DD:8D F2 03 ! STA $03F2  vector to
28E0:A9 FF    ! LDA #$FF   Monitor
28E2:8D F3 03 ! STA $03F2
28E5:20 6F FB ! JSR $FB6F fix powerup byte
28E8:4C 65 FF ! JMP $FF65  jump to Monitor
28EB:FF       !            next five bytes
28EC:6A       !            correct the
28ED:00       !            checksum
28EE:00       !
28EF:00       !
```

6a) If your ROM burner can burn 32K EPROMS then save the new code with:

BSAVE CD,A$1000,L$2000

6b) If your ROM burner can't burn 32K EPROMS then save the new code with:

BSAVE ROM.C,A$1000,L$1000
BSAVE ROM.D,A$2000,L$1000

7) Burn a 2764 EPROM with the code from step 6 and replace the CD ROM on the motherboard with it.

Using the New Patch

The new patch is used in exactly the same way as the old one, but it does give you some changes. If you press the Reset key after using the Don Lancaster absolute Reset, it will no longer cold boot but jump into the Monitor. If you look at memory locations $2FE and $2FF, you will find the address where the Reset vector pointed before using the absolute Reset.

When Resetting into the Monitor with either Lancaster's patch or our new patched patch, DOS will be disconnected. This is often helpful when breaking into commercial software because some commercial software set the DOS hooks to point to a routine that wipes memory as soon as any command is typed. You can use "3EAG" (for DOS 3.3) or "BE00G" (for ProDOS) from the Monitor to re-connect DOS. If a strange DOS is being used, these commands may not have any effect. You may prefer to type "E000G" to enter BASIC instead.

Why the COMPUTIST Reset Doesn't Always Work

There are some programs, such as AppleWriter, Homeword, Flight Simulator, and Skyfox, that the COMPUTIST Reset will not escape out of. This is due to the fact that the RAM card remains enabled on Apple][Plus's when Reset is pressed. If the commercial software has not copied your F8 ROM over into it but instead has activated it with their own version of the F8 then you will not be able to Reset into the Monitor.

On Apple][Plus's when you press Reset, the Apple jumps to whatever address it finds at $FFFC, but that $FFFC could be the one from the ROM or the RAM depending on the soft switch settings. The only way to break through these would be with a copy card whose hardware is able to control the bank switching.

On Apple //e's and //c's however, the ROM is automatically enabled when the Reset key is pressed. Therefore any ROM changes you make will be usable on any program!

Program 1: Checksummer

```
10 FOR X = 63488 TO 65535
20 IF SUM > 65280 THEN SUM = SUM - 65536
30 SUM = SUM + PEEK (X) : NEXT
40 IF SUM < 0 THEN SUM = SUM + 65536
50 PRINT "CHECKSUM^ =^ " ;SUM
```

checksums

```
10  - $38AD     40  - $E38E
20  - $4FC9     50  - $542D
30  - $EB3E
```

SpellWorks

by A. L. Head, Jr.

Advanced Logic Systems, Inc.
1195 East Arques Ave.
Sunnyvale, CA 94086
$49.95

Requirements:
128K Apple //e or //c
2 drives preferred
SpellWorks
ProDos User's Disk
An INITialized DOS 3.3 disk
A formatted ProDOS data disk
A blank disk
NMI Capability (or other way to break into the monitor)

SpellWorks is produced by Advanced Logic Systems (ALS) and is programmed to interface with AppleWorks. On the front side is the Spelling Checker with a 90,000-word dictionary. On the back side is the Mail Merge program. Both sides are copy protected.

The Spelling Checker is designed to check AppleWorks files. When checking a file another file called MISSPELL is created in which all misspelled or unrecognized words are placed in alphabetical order. After the document is spell checked and the MISSPELL file is created, you can boot AppleWorks and put both the document being checked and the MISSPELL file on the desk top. This facilitates the comparison of the MISSPELL file with the document being spell checked. Unique words can be placed in a custom dictionary.

The Mail Merge program is designed to merge two AppleWorks word processor files. A typical application is a form letter of some sort with as many fields as desired. The list file or sets of data file can be created by using either the AppleWorks word processor or data base. In the latter case the data base file must be printed to the clipboard and then saved as a word processor file. With the list file and the form letter file on disk the Mail Merge program can be run. A merged file will be created and stored on disk under whatever file name selected. AppleWorks must then be re-booted to examine and print the merged file.

In both of these SpellWorks programs ProDOS conventions must be observed. The opening menus provide BASIC program options and the additional options of changing the prefix and indexing the volume whose prefix is set. In the Spelling Checker the index option will display all of the files on the volume selected without showing the type of file. In the Mail Merge the index option will display only AppleWorks word processor files. This is not explicitly stated in the documentation; however, Mail Merge will work only with AppleWorks word processor files.

Each of the SpellWorks programs takes advantage of the capabilities of AppleWorks; but each requires constant operator intervention. On the positive side both will get the job done, and the cost is reasonable, especially if purchased through mail order houses. On the negative side there are several troublesome features. Both programs require the creation of files and the saving of them to the selected volume. Yet, in the index (catalog) option on each the free space available on a volume is not shown. Neither is there a way to determine the data disk volume name from within the program. These are inconvenient deficiencies that could be easily remedied. In the Spelling Checker the alphabetical listing of misspelled and unrecognized words in a MISSPELL file leaves it to one's own devices and knowledge of AppleWorks to get the corrections made. Neither displays of flagged words in context nor suggested spellings are featured. In the Mail Merge the creation of a merged file could be unacceptable if many form letters were involved. At best the file would occupy much disk space. If one had a hard drive with both AppleWorks and SpellWorks installed, the constant flipping between the programs and AppleWorks would be facilitated. SpellWorks cannot be installed on a hard drive unless it is deprotected. This provides the justification for and the purpose of this article - the deprotection of SpellWorks for bona fide owners.

The Protection

SpellWorks uses the ProDOS operating system. The discussion given in this section describes to some extent how ProDOS boots and how the protection scheme is implemented. The steps outlined here to determine the nature of the protection are applicable to other protected ProDOS programs. Step-by-step procedures for deprotecting SpellWorks are presented later.

The use of ProDOS was verified by booting the ProDOS Users Disk, entering BASIC, and obtaining a catalog of each side. The volume name for side 1 is SPELLWORKS. The files are SPELL.SYSTEM, LEX, CP, PRODOS, TMP, and MISSPELL. The volume name for side 2 is MAILMERGE. The files are CLAUSDB, CLAUSLIST, CLAUSMERGE, PRODOS, CLAUSDBLIST, MERGE. SYSTEM, CLAUS, and MRGMSG. On this side all of the CLAUS- files are associated with the examples provided on the disk. The fact that a catalog can be obtained indicates that a near normal ProDOS has been used.

Using a nibble editor I examined the format of both sides. As expected, a more-or-less standard 16-sector format is used. The sync-byte used is ''AA'' instead of the standard ''FF''. ProDOS could care less. Also, on each track in the large sync-field preceding sector $00, fourteen (14) non-sync bytes are inserted

as follows: AD AA B3 FE AE FE AE FE AE FE AE AC AC AC. This looks like it could be an address field; however, standard address and data marks follow after another large sync-field. I believe that the dummy address field is inserted to attempt to fool bit copiers that find the track start by looking for the largest sync-field. At this point I decided to copy each side onto a separate disk using a standard copier. I used the copy program on the ProDOS Users Disk. COPYA would work just as well. Each side copied with no indication of error; however, each would hang during boot after loading ProDOS. As a matter of interest, I also successfully bit-copied each side using EDD-III with default settings. This produces satisfactory, protected backup copies. An interesting feature of bit-copying is that one can see the track lengths. The track lengths of the original are extremely short. The corresponding copied tracks are approximately $80 bytes longer with the copy drive running at the standard speed of 300 revolutions per second. This indicates that no track length check is being done.

It will help to understand a little about the ProDOS operating system. Most of us are comfortable with DOS 3.3 because we have worked with it for a long time. Conversely, we are not as comfortable with ProDOS yet. Since ProDOS is a superior operating system we must take the plunge and learn it. During the boot, ProDOS is loaded into RAM. It then loads the first "xxxx.SYSTEM" file it can find at memory location $2000 and then executes it. Usually, this is BASIC.SYSTEM, but it can be any system file. If it is BASIC.SYSTEM, control is passed back to the BASIC Interpreter (BI); and a BASIC file, STARTUP (ProDOS' equivalent of HELLO), is ran if it can be found on the disk. Otherwise, the machine is left in BASIC as indicated by the BASIC cursor,']'. A custom system file does not have to pass control back to BASIC. It can provide its own interpreter. Therefore, the system file is a good place for someone wishing to protect a program to bury their protection scheme. To determine the nature of the protection used in the SpellWorks programs, I decided to to see if I could run either program from a ProDOS BASIC environment. With the ProDOS Users Disk in drive 1 and the protected program disk in drive 2,

1) Boot the User's Disk and enter BASIC.

2) Set the prefix of the protected disk.

PREFIX,D2

3) Run the system file. (SPELL.SYSTEM, Side 1; MERGE.SYSTEM, Side 2).

-SPELL.SYSTEM

If the protected program executes and functions normally, a standard ProDOS is being used and the protection is contained in the system file. Following the above procedure, both sides of SpellWorks execute and function normally. The next step is to find where the protection is in the system file and then circumvent it. It has already been determined that the standard copies hang after loading ProDOS. A disk monitor would help greatly at this stage to find out exactly what is happening when the programs hang. Fortunately, such a monitor is available in COMPUTIST No. 21. It is called Proshadow. This is a great little program that monitors what the disk is doing. I patched Proshadow so that it would leave its display on the screen. Continuing, if one can find where the copies hang, one is close to what one needs to know. To do this I rebooted ProDOS into BASIC and then "BRUN PROSHADOW". With Proshadow installed I executed steps 2 and 3 above using the standard copies in drive 2. The Spelling Checker, side 1, reads the file "TMP" and then hangs in the file "CP". This information is not directly displayed. The display shows whether the disk head is reading, writing, or formatting; the slot and drive numbers; the memory buffer being accessed; and the corresponding block numbers. In other words, PROSHADOW tells one what is happening and where it is happening. To translate this information into file names means that one must know where the files are on the disk in terms of ProDOS block numbers. Copy][Plus 6.0 has a disk map function that works with either ProDOS or DOS 3.3. The files can be located on the disk in terms of DOS 3.3 logical sectors. DOS 3.3 logical sectors can be translated into physical sector numbers which can then be expressed in terms of ProDOS block numbers. Quality Software's "Beneath Apple DOS" and "Beneath Apple ProDOS" are indispensable references for anyone doing this kind of work. If these books are not available, the article "DOS to ProDOS and Back" in COMPUTIST No. 25 contains the appropriate translate table. Following the same procedure, the Mail Merge program, side 2, hangs in the file "MRGMSG". Armed with this knowledge, the system files will be closely examined.

In the following, I will examine the Spelling Checker first and then give the same appropriate treatment to the Mail Merge program. The system file could be loaded into memory at location $2000 and examined; however, due to the disk access involving the files "TMP" and "CP", it is better to run the system file using the original disk and the interrupt it. Repeat steps 1, 2 and 3 above using SPELL.SYSTEM. Interrupt the program at the first menu and enter the monitor. As explained below in the Step-By-Step section, Spelling Checker cannot be interrupted by simply pressing Control-Reset. A direct reset to the monitor or a non-maskable interrupt capability is needed. Conversely, Mail Merge can be interrupted with a control-reset keypress. Continuing with Spelling Checker,

4) enter the monitor and disassemble the code starting at $2000 with the "L" command (as in **2000L**) and follow along with the discussion below. Type another "L" for each screenful of code.

At $2000 the execution immediately jumps to $2088 where the kernel and interpreter are initialized and some memory management takes place through $209F. At $20A1 is a JSR (jump to subroutine) to $BF00. This is the entry to the Machine Language Interface (MLI) in the ProDOS kernel. Twenty-six (26) different functions may be performed to access and manipulate files. Beneath Apple ProDOS describes all of these operations well. Just enough information will be given here for one to follow what is happening.

After a call to $BF00, ProDOS expects to find two data items following the JSR instruction. Byte 0 (the first byte) defines the function code. Bytes 1/2 define the address of the parameter list in lo-hi form. Referring to $20A1, there is a $C8 at $20A4 which is the code to open a file. The next two bytes point to the address $457D for the parameters of this call. The format of the parameter list is as follows: the zero byte ($457D) is an $03 which means there are three parameters to follow; the 1/2 bytes ($457E and $457F) point to the address of the file's pathname in lo-hi form ($459A); the 3/4 bytes point to the buffer address used by MLI while the file is open; and byte 5 is the reference number assigned to this open file ($A0 in this case). If one disassembles the code beginning at $459A and deciphers the hex code into ASCII, he will find "/spellworks/tmp" as the pathname. Do not worry about the lowercase letters. ProDOS will accept either lowercase or uppercase. So, we have discovered that the file TMP has been opened. Examining the code from $20A1 through $2103 shows that the following functions are being done through the MLI:

```
$20A1 open TMP as described above
$20B8 write $200 bytes at $6000 to TMP
$20C3 open CP
$20D4 read CP into A$6000, L$0200
$20DC JSR to $6000 (do protection code)
$20E1 an endless loop if checks fail
$20E4 close CP
$20EC reset the data pointer of TMP to 0
$20F4 read TMP into A$6000, L$0200
$20FC close TMP
$2104 start of program if checks pass
```

From the above summary, space is made in memory at $6000 by writing $0200 bytes to a temporary file on disk named TMP. Then, the file named CP, the copy protection code, is loaded into $6000 and executed. After this an endless loop is encountered if the checks fail. The file CP is closed. Then, the TMP data pointer is reset to the zero position, and the code stored in TMP is loaded back into memory where it came from. TMP is closed, and the execution falls through to the start of the program at $2104. Detailed examination of the code shows that the execution of the protection code at $6000 alters SPELL.SYSTEM in memory pages $27 and $41 through $45.

The same type of information derived above for the Spelling Checker can be obtained for the Mail Merge. Run MERGE.SYSTEM as done above and interrupt the program by pressing Control-Reset. Disassemble the code beginning at location $2000. Briefly, this code initializes the kernel and interpreter, performs memory management, and set up some page

$33 data between addresses $2000 and $204D. At $204E is the first call to the MLI. A summary of what happens follows:

```
$204E open MRGMSG
$2056 set up an endless loop
$2071 read MRGMSG into A$6000, L$18D
$2079 JSR $6000 (execute protection code)
$207E start of program if all checks clear
```

The Mail Merge program protection is similar to the one used for the Spelling Checker. As summarized above, the system file opens MRGMSG, a file containing a merge message, reads it into memory at $6000, and then executes it. If all the checks clear, the execution of the code falls through to the start of the program at $207E. If the checks fail, the execution goes into an endless loop at $2056. Detailed examination of the protection code at $6000 shows that its execution modifies MERGE.SYSTEM in memory pages $33 through $36.

The Deprotection

Each of the SpellWorks programs uses a protection scheme that involves loading code from the disk, performing checks, and modifying the system file in memory before jumping to the start of the program. With the knowledge derived in the last section, it should be possible to run the original programs and let them satisfy all of the protection checks. Then, interrupt each at the first menu, patch the system code to bypass the checking part of the code, and jump directly to the start of the program. The entire modified code can be saved back to the disk as a modified system file. Before saving this to disk, the validity of this approach can be checked by executing the modified code from the monitor. This was done successfully for each program. A step-by-step deprotection procedure follows for each program.

Spell Checker Step By Step

Spell Checker cannot be interrupted with just Control-Reset. A direct reset to the monitor or a non-maskable interrupt (NMI) capability is needed. I use the Senior PROM marketed by Cutting Edge Enterprises. The procedure involves the interruption of the program at the first menu, the patching of the code in memory, and the saving of the whole thing. The saving to disk step is involved because SPELL.SYSTEM is rather long, $7001 bytes. It cannot be saved directly because the BASIC.SYSTEM file is not present. Even if the program is run from a BASIC environment, the BASIC.SYSTEM file is overwritten or disabled. The modified SPELL.SYSTEM occupies memory from $2000 through $9001. There is no place in memory where it can be put without its being overwritten by any subsequent reboot of a ProDOS disk. However, the memory area in question is not overwritten by a DOS 3.3 slave disk. Therefore, a DOS 3.3 slave disk will be warm booted and the file will be saved as a DOS binary file. It will then be converted to a ProDOS system file and saved over the copy disk's protected SPELL.SYSTEM.

1) Copy the Spell Checker using a standard copier.

2) Boot the original Spell Checker and interrupt the program at the first menu using your favorite method.

3) Enter the monitor and patch Spell Checker to skip the protection.

20A1:4C 04 21

4) Insert a DOS 3.3 slave disk in drive 1 and type 6[Ctrl]P to boot.

5) Save the patched SPELL.SYSTEM to disk.

BSAVE SPELL.SYSTEM,A$2000,L$7001

6) Boot the ProDOS users disk and run the CONVERT program.

7) With the DOS 3.3 disk containing SPELL.SYSTEM in drive 2 and a ProDOS data disk in drive 1, SET ProDOS Prefix and TRANSFER SPELL.SYSTEM to the ProDOS disk.

8) Place the ProDOS users disk in drive 1 and the ProDOS disk containing SPELL.SYSTEM in drive 2. Select the QUIT option of the CONVERT program, and run BASIC.SYSTEM when prompted .

9) Load the SPELL.SYSTEM that you just converted from DOS 3.3 into memory.

BLOAD SPELL.SYSTEM,A$2000,D2

10) Insert the copied version of Spell Checker in drive 2.

11) Tell ProDOS to do everything on drive 2.

PREFIX,D2

12) Replace the protected SPELL.SYSTEM with the modified SPELL.SYSTEM.

UNLOCK SPELL.SYSTEM
DELETE SPELL.SYSTEM
CREATE SPELL.SYSTEM,TSYS
BSAVE
SPELL.SYSTEM,TSYS,A$2000,L$700
/1
LOCK SPELL.SYSTEM
DELETE TMP (optional step)
DELETE CP (optional step)

Spell Checker is now completely deprotected, and it can be installed on a hard drive. It can also be copied by COPYA or any standard copier.

Mail Merge Step by Step

The Mail Merge program is not as complicated to deprotect as the Spell Checker. It can be interrupted at the first menu with a Control-Reset keypress. Consequently, it will be convenient to use the ProDOS users disk as much as possible. The copy procedure follows:

1) Boot the ProDOS Users Disk

2) Select the FILER utilities and then the VOLUME commands.

3) Select the Copy program, insert a blank disk in drive 2 (use the back side of the Spell Checker that was just deprotected), and copy Mail Merge.

4) Return to the menus, Quit, and run BASIC.SYSTEM (go to BASIC).

5) Delete the old MERGE.SYSTEM and CREATE a blank .SYSTEM file.

PREFIX,D2
UNLOCK MERGE.SYSTEM
DELETE MERGE.SYSTEM
CREATE MERGE.SYSTEM,TSYS

6) Insert the original Mail Merge in place of the copy and type:

-MERGE.SYSTEM

7) At the first menu, drop into BASIC by pressing Control-Reset.

8) Patch the MERGE.SYSTEM in memory to skip the protection.

CALL-151
204E:4C 7E 20

9) Boot a DOS 3.3 slave disk by typing **C600G**.

10) Save MERGE.SYSTEM to the disk.

BSAVE MERGE.SYSTEM,TSYS
,A$2000,L$2504

11) Boot the ProDOS User's Disk and use the CONVERT utility to transfer the patched MERGE.SYSTEM from DOS 3.3 to a ProDOS data disk.

12) Exit the CONVERT utility and Quit to BASIC.

13) Load MERGE.SYSTEM into memory.

BLOAD MERGE.SYSTEM,A$2000

14) Insert the copy disk and save the new .SYSTEM file.

BSAVE MERGE.SYSTEM,TSYS
,A$2000,L$2504
LOCK MERGE.SYSTEM
DELETE MRGMSG (optional)

Mail Merge is also completely deprotected and can be installed on a hard drive.

Reflections

In the procedure I have used to deprotect the SpellWorks programs it has not been necessary to analyze the actual protection being employed. However, the modification of the respective system files has been captured and the protection circumvented thereafter. For instance, it has not been determined whether or not the protection actually does a signature check of the dummy address field prior to sector $00 on each track. ALS has obfuscated the protection code loaded into $6000 very well. All of this is bypassed with the interrupt after the protection has been executed. It is likely that ALS uses this type of protection on other software. If so, those products can probably be deprotected in the same manner.

That's all folks.

Bootcode Tracing...

GUMBALL

Rich Etarip

Broderbund Software
17 Paul Drive
San Rafael, CA 94903

Requirements:
48K
Initialized disk with no Hello program
Gumball Diskette

For myself, I could call Broderbund's *Gumball* two games on one disk. First, there is the arcade type game where the object is to sort the colored gumballs in the proper colored containers. Then there is the mind boggling game called "Boot Code Trace The Disk". The object of this game is to control the boot process up to the point where the game is loaded in and then save the game on normal DOS 3.3. Since Gumball is a challenging game and Broderbund's protection is no less challenging to break, I hesitate to say which game is easier. With enough time and dedication, though, both games can be conquered.

In this article, I will focus on the techniques used to win at the second game: deprotecting the program.

I knew I was in for somewhat of a challenge when I began tracing the boot code on this disk. The last Broderbund game I deprotected was Puck Man a few years ago. Since then, our friends at Broderbund have really progressed in the complexity of their boot codes. If I remember correctly, Puck Man had a mere three boot stages which doesn't compare to the nine that Gumball has. Fortunately, you will not need to trace through all of them. More on that later.

Right now it is time to start in level 1 of our game.

1) The first step is entering the monitor to begin the boot code trace.

CALL -151

2) Next, we want to move the boot program at $C600 to a place in memory where we can modify it to read in the first boot stage and return control to us. The code at $C600 is relocatable anywhere in memory but I like to use $9600.

9600<C600.C6FFM

3) Now, if you look at $96F8, you will see a JMP $801. This is where the first boot stage begins. If we change that JMP instruction to jump to the monitor the first boot stage will load in and we will have control of it. We will change the JMP $801 to a JMP $9801 and at $9801 we will put a routine to turn off the drive and jump to the monitor.

96FA:98 N 9801:AD E8 C0 4C 59 FF

In case you are confused by the "N" between instructions here, this is the monitor's "Normal" (rather than "inverse characters") command. It can be used to separate instructions, kind of like a colon in a BASIC program.

4) At this point we are ready to execute the boot at $9600.

9600G

5) When the drive stops we will move the boot code at page $8 to page $98 and change a few bytes so it will run at that location.

9800<800.8FFM

6) If you list through this boot stage you will see that it relocates itself to page $2 and jumps to $20F. We have to change it so it moves page $98 to $200 rather than page $8 because page $98 is what we are working with.

9805:98

7) Looking at $9837 you will see a JMP $301 which is the jump to the next boot stage. If we change this to jump to the monitor, boot stage 2 will load in and we will again have control of it.

9838:59 FF

8) We are now ready to execute the boot program at $9600.

9600G

9) When the Apple beeps, turn off the disk drive motor by accessing $C0E8.

C0E8

10) Boot stage 2 is now loaded in at $300 and we can move it to page $93 to modify it.

9300<300.3FFM

11) We also have to change the jump that exits boot 1 to jump to the modified boot 2 at $9300.

9838:01 93

12) If you list through page $93 you will see that it clears hi-res page 2 ($4000) and does some memory decoding. At the end you will find it loading the X register with #$CF, transferring it to the stack pointer and executing an RTS. This is an encoded jump since an RTS instruction takes two values off the stack and jumps to that address plus one. These encoded jumps are scattered generously throughout the boot making it a bit harder to trace. What we

want to do is write a short routine to pull the two top values off the stack and store them somewhere so we can see what they are. We will put our routine at $9500 and jump there at the end of boot stage 2.

```
932B:4C 00 95
9500:A2 CF 9A 68 8D 00 20 68
9508:8D 01 20 4C 59 FF
```

13) That little routine sets the stack pointer at #$CF, pulls two values off the stack and stores them at $2000 and $2001. Everything is set and the boot program can be executed again.

```
9600G
```

14) When the Apple beeps again, turn off the drive and check locations $2000 and $2001 to see where boot 2 is jumping.

```
C0E8 N 2000.2001
```

15) You should see a $2F and a $01. Adding 1 to that, you can see that it is jumping to $0130. The routine at $130 is not loaded in from the disk, though. If you look at $9320 you will see that it is moving page $3 to page $1, but in the process, Exclusive-ORing the bytes with the value at location $48. Now rather than going through this process every time, I prefer to modify this code before it is moved to save some confusion. First, we should move the decoded memory back to page $93 where it came from. Actually it came from page $3, but remember, our page $3 is at $9300.

```
9330<130.1FFM
```

16) Now we have to change boot 2 so it moves this routine from page $93 instead of page $3 and remove the EOR instruction.

```
9322:93 EA EA
```

17) If you examine the routine that goes at $130 you will see that it is loading in the next boot stage at page $4, which is the text page. We cannot modify code when it is on the text page because it will scroll off the screen after we change it. So, we will have to change the routine so it loads into safe memory like $6400 and later memory move it back to $400 after we change it. At $130 the X register is loaded with $04 which specifies page $4 but it also uses that for a counter to load in 4 pages of memory. If we change it to a $64 it will load into $6400 but it will also attempt to load in $64 pages of memory which we don't want. Location $86 is used for the page counter so we want a $04 in this byte. What we can do is change the routine at $130 so it stores nothing in $86 and put a $04 in $86 before jumping there. Since we are jumping to location $9500 upon exit of boot 2 we can do it right here and then jump to $130

```
9503:A9 04 85 86 68 68 4C 30 01
```

18) We also have to change the routine that goes at $130 to load into page $64 but not store a $64 in location $86.

```
9331:64 EA EA
```

19) The next step is finding the jump that exits the routine at $130. At $937D is another RTS (I warned you) and this is where the boot is exited. Again, we will have to jump to a routine to pull the values off the stack and store them at $2000 and $2001.

```
937D:4C 10 95
9510:68 8D 00 20 68 8D 01 20
9518:4C 59 FF
```

20) Again, we are all set to execute the boot program at $9600.

```
9600G
```

21) The text page boot is now loaded in at $6400. Turn off the drive and check locations $2000 and $2001.

```
C0E8 N 2000.2001
```

22) They should be $FF and $03 so the next address jumped to is $400. Now we must change the routine at $130 so it will load the text page boot where it belongs. If we did not do this, the boot stage would load right over our modified boot code at $6400. Once we modify the code at $6400, it can be moved to the text page and executed. This step will change the routine at $130 back to what it was. Remember, $9330 is moved into $130.

```
9331:04 86 86
```

23) Now write a memory move at $9510 to move the text page boot code back to the text page and then jump to $400.

```
9510:A2 00 BD 00 64 9D 00 04
9518:BD 00 65 9D 00 05 BD 00
9520:66 9D 00 06 BD 00 67 9D
9528:00 07 E8 D0 E5 68 68 4C
9530:00 04
```

24) Now the code that loads into the text page moves pages $5 through $7 into $BD through $BF. Then it loads in another boot stage at $500 and jumps to it. This boot stage loads in yet another at $100. The first time I traced through this boot code, I found that it ends up back at the original text page boot, which would then be at $BD00. Knowing this, it would be pointless to go through the two extra boot stages that load in at $500 and $100. It could be described as running around the block to get to your house when you are already there. If you list through page $67 (which will be at $BF00) you will find valid machine code up to $679F, then garbage... or so it seems. If you look at the routine at $66B0 you will see that it decodes the memory from $BF9F to $BFFF by means of Exclusive-ORing. Rather than letting the boot routine itself call this routine, we will execute this routine and see what lies beneath the encrypted code. Since pages $BD through $BF are at $6500, we will have to modify the routine at $66B0 to decode the memory where it is.

```
66B4:67 N 66B7:66 N 66BA:67
```

25) Now we can execute this routine and put an RTS at the start so it will no longer be used.

```
66B0G N 66B0:A2 60 60
```

26) We must load the X register with $60 because that is done by the routine before returning. If you list through page $67 you will see some values being pushed on the stack at $67E3 and then an RTS. Having gone through this boot code prior to writing this article, I found that the RTS will cause a jump to zero page location $003C. If you were to stop the boot at that point and look at $3C through $3E you would see 003C- 3C 00 3F which is not rational machine code. The reason is that those locations are used by the monitor and they become overwritten when you use the monitor.

Fortunately for you, I went through the torture of tracing this boot code time and time again to find out where everything was jumping. At location $003C is a JMP $B200 before it gets overwritten. If you were to put a jump to the monitor at $67E3 and execute the boot, you would eventually come up with the title program for Gumball after tracing through 3 more short boot routines. What one would usually have to do in a situation like this would be to load in the title program, trace through it, and find the program's load routine. In this case, though, we don't have to. Broderbund uses a universal loading routine at $BF6F. This load routine consists of 4 somewhat simple stages and the exit from the first one is at $BFE3. You would find this at $67E3 because this code is first moved to the text page by our routine and then to pages $BD through $BF. The boot code will first go to $BF6F to load in the title program and then again to load in the game afterward.

What we can do is put a JMP $BF00 at $67E3. Then we can write a routine at $BF00 that will change the JMP $BF00 to a JMP $FF59 and jump to $003C to continue loading in the title page. That way, when the game begins loading in, the boot routine will jump to $FF59 this time instead of $BF00. I chose $BF00 for the routine because it is just the reset routine that clears memory and reboots. Right now $BF00 is at $6700 so we have to write the routine at $6700. We must also put a jump to our routine at $67E3.

```
6700:A9 59 8D E4 BF A9 FF 8D
6708:E5 BF 4C 3C 00
67E3:4C 00 BF
```

27) Now we are ready to execute the boot routine again. When the title program begins, press a key and the drive will start, and you will have control.

```
9600G
```

28) The boot code would have initially jumped to $003C and at $003C is a JMP $B200 so, with the drive still running, list through $B200. At $B20C is a JMP $B500 which is another boot stage, so change it to a JMP $FF59 and execute $B200.

```
B20D:59 FF N B200G
```

29) When the Apple beeps, turn off the drive.

```
C0E8
```

30) If you list through $B500, you will see a JSR $B700 and a few other instructions before

a JMP $16C4. At first I was fooled but later I learned that it never returns from the JSR $B700. Listing through $B700, at $B713 is an ADC $B709 which is not really suspicious looking until you look at the instruction before it. At $B710 is a DEC $B713 and that instruction turns the ADC $B709 (at $B713) into an indirect jump to $B709 which would actually be a jump to $300. If we put a jump to the monitor at $B710 then we can take control before it jumps to $300. Make this modification, turn on the drive and execute the code at $B500. This will load in almost all of the game.

B710:4C 59 FF
C0E9
B500G

31) Now turn off the drive.

C0E8

32) When you list through $300, you will see it pushing values on the stack and you will see another example of self modifying code. At $375 is a DEC $378 and the byte at $378 is a $29. When decremented it becomes a $28 which is the code for a PLP (pull processor status from stack) and then at $379 is a DEC $37C and at $37C is a $61. When decremented it becomes a $60 which is the code for an RTS. What is done here is simply a PLP and an RTS. Rather than bother with the self modifying code, we will just exit the boot stage at $375.

Since this is the very last boot stage (yes, AT LAST!) we must jump to a routine that will save the lower pages of memory that become scrambled such as the zero page, the stack, and the keyboard buffer. I find it best to save pages $0 to $3 because sometimes programs use the data stored in them. In this case it doesn't appear so at first but the later levels will not work correctly without them. Since $BF00 is only the Broderbund reset routine, we can put our routine there. In this routine we also have to pull the processor and check the values on the stack. From my research I found that four values must be pulled from the stack because the first two point to $BEB0 which is a routine that ends in an RTS. The 3rd and 4th values will be the values for the jump to the start of the game. First we will change $375 to JMP $BF00 and then write the routine.

375:4C 00 BF
BF00:28 68 68 68 8D 00 44 68
BF08:8D 01 44 A2 00 BD 00 00
BF10:9D 00 40 BD 00 01 9D 00
BF18:41 BD 00 02 9D 00 42 BD
BF20:00 03 9D 00 43 E8 D0 E5
BF28:4C 59 FF

Type **BF00L** and compare it with the following listing to assure that it was entered correctly.

```
BF00-  28          PLP
BF01-  68          PLA
BF02-  68          PLA
BF03-  68          PLA
BF04-  8D 00 44    STA $4400
BF07-  68          PLA
BF08-  8D 01 44    STA $4401
BF0B-  A2 00       LDX #$00
BF0D-  BD 00 00    LDA $0000,X
BF10-  9D 00 40    STA $4000,X
BF13-  BD 00 01    LDA $0100,X
BF16-  9D 00 41    STA $4100,X
BF19-  BD 00 02    LDA $0200,X
BF1C-  9D 00 42    STA $4200,X
BF1F-  BD 00 03    LDA $0300,X
BF22-  9D 00 43    STA $4300,X
BF25-  E8          INX
BF26-  D0 E5       BNE $BF0D
BF28-  4C 59 FF    JMP $FF59
BF2B-  00          BRK
```

33) Continuing on, the values pulled from the stack will be stored at $4400 and $4401 and pages $0 through $3 will be at page $40. This is a safe place to store memory because I found it to be a blank hi-res page. Now all we have to do is turn on the drive and execute $300.

C0E9 N 300G

34) When you hear the beep the game will be completely loaded in. Now check locations $4400 and $4401 to see where it was jumping. It should be $F8 and $31. If not, you may have done something wrong along the way. If everything is right, then you are ready for the next step. The program goes from $800 to $B100 so we have to move the code that occupies the DOS area somewhere else. We still have the rest of hi-res page 2 to work with so this is a good place to move it. We will move pages $95 through $B0 into $4400 so we will be saving everything from $800 to $9500.

4400<9500.B0FFM

35) We must also move page $8 to a safe place so it is not overwritten when we reboot DOS. We can put it at $9500.

9500<800.8FFM

36) Now insert a 48K slave disk with no hello program and reboot DOS.

6[CTRL-P]

37) Next, enter the monitor and then move page $95 back to $800 where it belongs.

CALL-151
800<9500.95FFM

38) Now we must write a memory move routine to restore pages $0 to $3 and pages $95 to $B0. Since there is just 'junk' memory in page $42 (which moves to page 2) we can put it right there.

4200:A2 00 BD 00 40 9D 00 00
4208:BD 00 41 9D 00 01 BD 00
4210:42 9D 00 02 BD 00 43 9D
4218:00 03 E8 D0 E5 BD 00 44
4220:9D 00 95 E8 D0 F7 EE 1F
4228:42 EE 22 42 AD 1F 42 C9
4230:60 D0 EA 4C F9 31

After entering this routine, type **4200L** and compare it to the following listing to make sure there were no typing errors.

```
4200-  A2 00       LDX #$00
4202-  BD 00 40    LDA $4000,X
4205-  9D 00 00    STA $0000,X
4208-  BD 00 41    LDA $4100,X
420B-  9D 00 01    STA $0100,X
420E-  BD 00 42    LDA $4200,X
4211-  9D 00 02    STA $0200,X
4214-  BD 00 43    LDA $4300,X
4217-  9D 00 03    STA $0300,X
421A-  E8          INX
421B-  D0 E5       BNE $4202
421D-  BD 00 44    LDA $4400,X
4220-  9D 00 95    STA $9500,X
4223-  E8          INX
4224-  D0 F7       BNE $421D
4226-  EE 1F 42    INC $421F
4229-  EE 22 42    INC $4222
422C-  AD 1F 42    LDA $421F
422F-  C9 60       CMP #$60
4231-  D0 EA       BNE $421D
4233-  4C F9 31    JMP $31F9
```

This will perform the memory moves and jump to the start of the game at $31F9.

39) We also have to put a jump to this routine right before the start address of the program.

7FD:4C 00 42

40) A few modifications must be made to the game so it will not attempt to access the disk for the cartoons or high scores. From examining the boot code, I found that it uses the address of the character output routine ($FDED) for the disk access routine. I searched through the program for the bytes $ED and $FD and made the necessary changes.

10FA:4C E5 10
58C7:EA EA EA
15B6:60
15C4:60

41) A modification must also be made so the game will end after level 5. The reason this change must be made is that the game normally ends during the cartoon if you complete level 5.

10CA:C5
3560:06

42) All we have to do now is patch DOS so we can save a file longer than $7FFF bytes and then we can save the game.

A964:FF
BSAVE GUMBALL,A$7FD,L$8D03

Now you have yourself a BRUNable file of Gumball. For some strange reason (which I can't explain) levels 3 and up will not work if you BLOAD the game and execute $7FD. If you BRUN the file, however, it will work fine. I have also included an APT or rather a cheat, for Gumball. At locations $5DB8 through $5DBC are the required quotas for levels 1-5 respectively. These locations can be changed to make it easier (or harder).

Remember that you must save the altered file, and coldstart the computer before BRUNing the game. Why this has to be done I can't explain, but as I mentioned earlier, level 3 will not work if you do not do this. Good Luck!

Coming soon...

(to a magazine near you)

COMPUTIST No. 35:

Softkeys for: Hi-res Cribbage, Olympic Decathlon, Revisiting F-15 Strike Eagle, Masquerade, The Hobbit, Pooyan, The Perfect Score, Alice in Wonderland, The Money Manager, Good Thinking.

Feature: Putting a New F8 on your Language Card.

Core: Exploring ProDOS by installing a CPS Clock Driver.

Review: The Senior PROM.

APT: Sword of Kadash.

COMPUTIST No. 36:

Softkeys for: Flight Simulator II v1.05, Autoduel, Critical Reading, Troll's Tale, Robot War, General Manager, Plasmania, Telarium Software, Kidwriter v1.0, Color Me.

Features: Screenwriter meets Flashcard, The Bus Monitor, Mousepaint for non-Apples.

Core: a character editor: The Bard's Dressing Room

APT: Championship Lode Runner.

BULLSHIRT!

by bobby
PHD, phD, Phd, PhD,
BHT, BLT, MSG, McDLT

Requirements
some $$$
postage stamp
Envelope
Pen or compatible word processor
scissors (optional)

Now that HARDCORE has changed its name to plain old 'computist', I've been assigned the regrettably redundant responsibility of changing all the Diskbusters T-shirts. Instead, using my own imaginary initiative, I've come up with a reasonable facsimile of an original idea...

"Why not," I asked myself, "call all the old Diskbusters T-shirt something catchy like: Hardcore Classic?"

"Wow!" I answered myself, "Then we could sell people the NEW computist T-shirts this summer!"

(I've heard that some people suspect that there's a rumor that such an artifact does not yet exist...VAPORWEAR...)"

If you like my idea, then PLEASE order the CLASSIC Hardcore Computist DiskBusters T-shirt (simply known as CLASSIC T-shirt)

1) Fill out the order form below using the Pen and then insert the order form with appropriate $$$ or acceptable substitutes into the envelope, and then pen-in the address onto the envelope, and then attach the stamp and that's it.

Enjoy!

I want the CLASSIC! so rush me _____ (total) Classic DISKBUSTERS T-SHIRTS in the sizes indicated below. I have enclosed $9.95 (plus tax and shipping) for each shirt.

ADULT MENS: _____Small _____Medium _____Large _____Xtra large

Name_____ID#_____

Address_____

City_____State_____Zip_____

Country _____ Phone _____

VISA MC _____-_____-_____-_____ Exp. _____

Signature_____CP34

Send check or money order to: Bullshirt; PO Box 110816-T; Tacoma, WA 98411. Washington orders add 7.8% sales tax. Foreign orders add 20% shipping and handling. US funds drawn on US bank. OFFER GOOD WHILE SUPPLIES LAST.

Description of Available Back Issues

33 *Softkeys* | Word Juggler | Tink! Tonk! | Sundog v2.0 | G.I. Joe & Lucas Film's Eidolon | Summer Games II | Thief | Instant Pascal | World's Greatest Football Game | *Readers' Softkeys* | Graphic Adventure #1 | Sensible Grammar & Extended Bookends | Chipwits | Hardball | King's Quest II | The World's Greatest Baseball Game | *Feature* | How to be the Sound Master | *Core* | The Mapping of Ultima IV |

32 *Softkeys* | Revisiting Music Construction Set | Cubit | Baudville Software | Hartley Software | Bridge | Early Games for Young Children | Tawala's Last Redoubt | *Readers' Softkeys* | Print Shop Companion | Kracking Vol II | Moebius | Mouse Budget, Mouse Word & Mouse Desk | Adventure Construction Set | *Feature* | Using Data Disks With Microzines | *Core* | Super IOB v1.5 a Reprint | .....

31 *Softkeys* | Trivia Fever | The Original Boston Computer Diet | Lifesaver | Synergistic Software | Blazing Paddles | Zardax | *Readers' Softkeys* | Time Zone | Tycoon | Earthly Delights | Jingle Disk | Crystal Caverns | Karate Champ | *Feature* | A Little Help With The Bard's Tale | *Core* | Black Box | Unrestricted Ampersand | .....

30 *Softkeys* | Millionaire | SSI's RDOS | Fantavision | Spy vs. Spy | Dragonworld | *Readers' Softkeys* | King's Quest | Mastering the SAT | Easy as ABC | Space Shuttle | The Factory | Visidex 1.1E | Sherlock Holmes | The Bards Tale | *Feature* | Increasing Your Disk Capacity | *Core* | Ultimaker IV, an Ultima IV Character Editor | .....

29 *Softkeys* | Threshold | Checkers v2.1 | Microtype | Gen. & Organic Chemistry Series | Uptown Trivia | Murder by the Dozen | *Readers' Softkeys* | Windham's Classics | Batter Up | Evelyn Wood's Dynamic Reader | Jenny of the Prairie | Learn About Sounds in Reading | Winter Games | *Feature* | Customizing the Monitor by Adding 65C02 Disassembly | *Core* | The Animator | .....

28 *Softkeys* | Ultima IV | Robot Odyssey | Rendezvous | Word Attack & Classmate | Three from Mindscape | Alphabetic Keyboarding | Hacker | Disk Director | Lode Runner | MIDI/4 | *Readers' Softkeys* | Algebra Series | Time is Money | Pitstop II | Apventure to Atlantis | *Feature* | Capturing the Hidden Archon Editor | *Core* | Fingerprint Plus: A Review | Beneath Beyond Castle Wolfenstein (part 2) | .....

27 *Softkeys* | Microzines 1-5 | Microzines 7-9 | Microzines (alternate method) | Phi Beta Filer | Sword of Kadash | *Readers' Softkeys* | Another Miner 2049er | Learning With Fuzzywomp | Bookends | Apple Logo II | Murder on the Zinderneuf | *Features* | Daleks: Exploring Artificial Intelligence | Making 32K or 16K Slave Disks | *Core* | The Games of 1985: part II | .....

26 *Softkeys* | Cannonball Blitz | Instant Recall | Gessler Spanish Software | More Stickybears | *Readers' Softkeys* | Financial Cookbook | Super Zaxxon | Wizardry | Preschool Fun | Holy Grail | Inca | 128K Zaxxon | *Feature* | ProEdit | *Core* | Games of 1985 part I | .....

25 *Softkeys* | DB Master 4.2 | Business Writer | Barron's Computer SAT | Take 1 | Bank Street Speller | Where In The World Is Carmen Sandiego | Bank Street Writer 128K | Word Challenge | *Readers' Softkeys* | Spy's Demise | Mind Prober | BC's Quest For Tires | Early Games | Homeword Speller | *Feature* | Adding IF THEN ELSE To Applesoft | *Core* | DOS To ProDOS And Back | .....

24 *Softkeys* | Electronic Arts software | Grolier software | Xyphus | F-15 Strike Eagle | Injured Engine | *Readers' Softkeys* | Mr. Robot And His Robot Factory | Applecillin II | Alphabet Zoo | Fathoms 40 | Story Maker | Early Games Matchmaker | Robots Of Dawn | *Feature* | Essential Data Duplicator copy parms | *Core* | Direct Sector Access From DOS | ..

23 *Softkeys* | Choplifter | Mulplot | Flashcalc | Karateka | Newsroom | E-Z Draw | *Readers' Softkeys* | Gato | Dino Eggs | Pinball Construction Set | TAC | The Print Shop: Graphics Library | Death In The Caribbean | *Features* | Using A.R.D. To Softkey Mars Cars | How To Be The Writemaster | *Core* | Wheel Of Money | .....

22 *Softkeys* | Miner 2049er | Lode Runner | A2-PB1 Pinball | *Readers' Softkeys* | The Heist | Old Ironsides | Grandma's House | In Search of the Most Amazing Thing | Morloc's Tower | Marauder | Sargon III | *Features* | Customized Drive Speed Control | Super IOB version 1.5 | *Core* | The Macro System | .....

20 *Softkeys* | Sargon III | Wizardry: Proving Grounds of the Mad Overlord and Knight of Diamonds | *Reader' Softkeys* | The Report Card V1.1 | Kidwriter | *Feature* | Apple][Boot ROM Disassembly | *Core* | The Graphic Grabber v3.0 | Copy II+ 5.0: A Review | The Know-Drive: A Hardware Evaluation | An Improved BASIC/Binary Combo | .....

19 *Readers' Softkeys* | Rendezvous With Rama | Peachtree's Back To Basics Accounting System | HSD Statistics Series | Arithmetickle | Arithmekicks and Early Games for Children | *Features* | Double Your ROM Space | Towards a Better F8 ROM | The Nibbler: A Utility Program to Examine Raw Nibbles From Disk | *Core* | The Games of 1984: In Review-part II | .....

17 *Softkeys* | The Print Shop | Crossword Magic | The Standing Stones | Beer Run | Skyfox | Random House Disks | *Features* | A Tutorial For Disk Inspection and the Use Of Super IOB | S-C Macro Assembler Directives (reprint) | *Core* | The Graphic Grabber For The Print Shop | The Lone Catalog Arranger v1.0 Part 2 | .....

16 *Softkey* | Sensible Speller for ProDOS | Sideways | *Readers' Softkeys* | Rescue Raiders | Sheila | Basic Building Blocks | Artsci Programs | Crossfire | *Feature* | Secret Weapon: RAMcard | *Core* | The Controller Writer | A Fix For The Beyond Castle Wolfenstein Softkey | The Lone Catalog Arranger Part 1 | .....

13 *Softkeys* | Laf Pak | Beyond Castle Wolfenstein | Transylvania | The Quest | Electronic Arts | Snooper Troops (Case 2) | DLM Software | Learning With Leeper | TellStar | *Core* | CSaver: The Advanced Way to Store Super IOB Controllers | Adding New Commands to DOS 3.3 | Fixing ProDOS 1.0.1 BSAVE Bug | *Review* | Enhancing Your Apple | *Feature* | Locksmith 5.0 and Locksmith Programming Language | .....

11 *Softkeys* | Sensible Speller | Exodus: Ultima III | *Readers' Softkeys* | SoftPorn Adventure | The Einstein Compiler v5.3 | Mask of The Sun | *Features* | Copy][Plus (4.4C): Update Of An Old Friend | Parameter List For Essential Data Duplicator | *Core* | Ultimaker III | The Mapping of Ultima III | Ultima II...The Rest Of The Picture | .....

7 *Softkeys* | Zaxxon | Mask of the Sun | Crush | Crumble & Chomp | Snake Byte | DB Master | Mouskattack | *Features* | Making Liberated Backups That Retain Their Copy Protection | S-C Assembler: Review | Disk Directory Designer | *Core* | Corefiler: Part 1 | Upper & Lower Case Output for Zork | ....

4 *Softkeys* | Ultima II | Witness | Prisoner II | Pest Patrol | Adventure Tips for Ultima II & III | Copy II Plus PARMS Update | *Feature* | Ultima II Character Editor | .....

1 *Softkeys* | Data Reporter | Multiplan | Zork | *Features* | PARMS for Copy II Plus | No More Bugs | APT's for Choplifter & Cannonball Blitz | *'Copycard' Reviews* | Replay | Crackshot | Snapshot | Wildcard |

CORE 3 .....Games:
Constructing Your Own Joystick | Compiling Games | *GAME REVIEWS:* Over 30 of the latest and best | Pick Of The Pack: All-time TOP 20 games | Destructive Forces | EAMON | Graphics Magician and GraFORTH | Dragon Dungeon | .....

CORE 2 .....Utilites:
Dynamic Menu | High Res: Scroll Demo | GOTO Label: Replace | Line Find | Quick Copy: Copy | ..

CORE 1 ....Graphics:
Memory Map | Text Graphics: Marquee | Boxes | Jagged Scroller | Low Res: Color Character Chart | High Res: Screen Cruncher | The UFO Factory | Color | Vector Graphics:Shimmering Shapes | A Shape Table Mini-Editor | Block Graphics: Arcade Quality Graphics for BASIC Programmers | Animation | ....

Hardcore Computing 3 .....
HyperDOS Creator | Menu Hello | Zyphyr Wars | Vector Graphics | Review of Bit Copiers | Boot Code Tracing | Softkey IOB | Interview with 'Mike' Markkula | .....

For special savings, order our 'Core Special' and receive all three CORE magazines for only $10.00

Back Issues & Library Disks order form

Issue	Mag $4.75	Disk $9.95	Both $12.95
34	☐	☐	☐
33	☐	☐	☐
32	☐	☐	☐
31	☐	☐	☐
30	☐	☐	☐
29	☐	☐	☐
28	☐	☐	☐
27	☐	☐	☐
26	☐	☐	☐
25	☐	☐	☐
24	☐	☐	☐
☆ 23	☐	☐	☐
22	☐	☐	☐
21	NA	☐	NA
20	☐	☐	☐
19	☐	☐	☐
18	NA	☐	NA
17	☐	☐	☐
16	☐	☐	☐
15	NA	☐	NA
14	NA	☐	NA
☆ 13	☐	☐	☐
12	NA	☐	NA
☆ 11	☐	☐	☐
10	NA	☐	NA
9	NA	☐	NA
8	NA	☐	NA
☆ 7	☐	☐	☐
6	NA	☐	NA
☆ 4	☐	☐	☐
3	NA	☐	NA
Core 2	☐	☐	☐
2	NA	☐	NA
1	☐	☐	☐
Core 1	☐	☐	☐
Core 3	☐	☐	☐
Computing 3	☐	NA	NA
Best of Hardcore Computing	NA	☐	NA
Core Special $10.00 (All three CORE magazines)	☐		

Special "Both" disk & magazine combination orders apply to one issue and its corresponding disk.

Some disks apply to more than one issue and are shown as taller boxes.

☆ We have a limited supply of these issues.

BACK ISSUES and LIBRARY DISKS

of COMPUTIST *(formerly Hardcore COMPUTIST)*
are still available, though some issues (marked NA) are sold out, library disks are available for ALL issues of COMPUTIST.

LIBRARY DISKS are perfect companions for COMPUTIST

Documentation for Library Disks is in the corresponding issue.

Send me the back issues and/or library disks indicated:

Name _____ ID# _____

Address _____

City _____ State _____ Zip _____

Country _____ Phone _____

VISA MasterCard _____-_____-_____-_____ Exp. _____

Signature _____ CP34

Send check or money order to: COMPUTIST PO Box 110846-T Tacoma, WA 98411. Most orders shipped UPS so please use street address. Offer good while supply lasts. In Washington state: add 7.8% sales tax.

Back Issue Rates For Foreign Orders

FOREIGN MAGAZINE ORDERS
Price for each magazine includes shipping.

	1 - 2 copies	3 to 4 copies	5 or more copies
Canada/Mexico	$8.00 each	$7.00 each	$6.00 each
Other Foreign	$14.25 each	$13.25 each	12.25 each

FOREIGN DISK ORDERS

Disks are $11.94 each (includes shipping). Special "Both" disk and magazine combinations shown do NOT apply to Foreign orders. US funds drawn on US banks. All foreign orders sent AIR RATES.

Writer's Guide

COMPUTIST

is a monthly magazine dedicated to the serious user of the Apple (or compatible) computer. COMPUTIST welcomes articles on a variety of subjects in all levels of technical difficulty but requires accurate data, technical competence, correct English usage, readable style, and fully defined jargon and buzzwords.

MANUSCRIPT MECHANICS

All manuscripts must be typed or printed on one side of the paper. Text should be double-spaced.

Printouts should use a non-compressed font with both upper and lower case. A letter quality mode is preferred, with each page torn at the perforation only. Pages need not be stapled together. The cover page of each manuscript should contain the following data:

TITLE OF WORK
FULL NAME OF AUTHOR
ADDRESS
PHONE NUMBER

Each page of the manuscript and program listing should include the author's name, the title of the work, and the page number in the upper right hand corner.

The article and any accompanying program **SHOULD BE SUBMITTED AS A STANDARD TEXT FILE ON A DOS 3.3 DISK.** Label the disk with the title of the work and the author's full name and address. **ON DISK, TEXT MUST BE SINGLE-SPACED ONLY.** Please identify your editing program.

Original disks are always returned as soon as possible. Other materials will be returned only when adequate return packaging and postage is enclosed. We are not responsible for unreturned submissions. We *will guarantee* the return of original commercial disks mailed to us for verification of an accompanying softkey.

You will be notified of the status of your submission within 4 to 6 weeks after it is received if the article is a softkey accompanied by an original disk. Please submit completed manuscripts directly; do not query first. Previously published material and simultaneous submissions are not accepted.

SUBJECTS

We prefer material on these topics:

1) Original program/article combinations
2) General articles (Apple computing)
3) Softkeys
4) Advanced Playing Techniques (APT's)
5) Hardware modifications
6) DOS modifications
7) Product reviews (hardware and software)
8) Utilities
9) Bit Copy Parameters

WRITING YOUR ARTICLE

Observe the following points of style:

A. Always assume that your reader is a novice and explain all buzzwords and technical jargon. Pay special attention to grammar and punctuation; we require technical competence but also good, readable style.

B. Whenever appropriate, a list of hardware and software requirements should be included at the beginning of the manuscript. When published, this list will be offset from the main text.

C. Include the name and address of the manufacturer and the price when a commercial program is mentioned. This is of particular importance in PRODUCT REVIEWS.

D. When submitting programs, first introduce the purpose of the program and features of special interest. Include background information describing its use. Tips for advanced uses, program modifications, and utilities can also be included. Avoid long print statements and use TABs instead of spaces.

Remember: A beginner should be able to type the program with ease.

E. A PROGRAM is not accepted for publication without an accompanying article. These articles, as well as articles on **hardware** and **DOS modifications** MUST summarize the action of the main routines and include a fully remarked listing.

F. GENERAL ARTICLES may include advanced tips, tutorials, and explorations of a particular aspect of Apple computing.

G. SOFTKEYS of any length are acceptable and must contain detailed step-by-step procedures. For each softkey, first introduce the locking technique used and then give precise steps to unlock the copy-protected program. Number each step whenever possible. We accept articles which explain locking techniques used in several programs published by the same company.

H. When altering game programs, the changes made are sometimes extensive enough to warrant the title of ADVANCED PLAYING TECHNIQUE (APT). APTs can deal with alterations to a program, deleting annoying sounds, acquiring more points in play and avoiding hazards. Again, provide step-by-step instructions to complete each APT and explain each step's function. APT's of 100 words or more are preferred.

AUTHOR'S RIGHTS

Each article is published under the author's byline. As a rule, all rights, as well as one-time reprint rights are purchased. Purchase of exclusive rights to programs is required; however, alternate arrangements may be made with individual authors depending on the merit of the contribution.

PAYMENTS

COMPUTIST pays upon publication. Rate of payment depends on the amount of editing required and the length of the article. Payment ranges from $20 to $50 per typeset page for an article. We also pay $10 to $20 for short softkeys and APT's. A fully explained softkey accompanied by the commercial disk for verification may earn up to $50 per typeset page.

Please mail your submissions to:

COMPUTIST
Editorial Department
PO Box 110846-T
Tacoma, WA 98411

big deal.

We really mean it. This is truly a big deal. We want to sell you a book or two. Need we say more?

The Book Of Softkeys Volume II is here.

At long last, The second volume in our series of compilations is ready. Once again, we have combined several issues of (Hardcore) COMPUTIST into one compact book. Volume II of the Book Of Softkeys contains articles from issues 6 through 10.

The Big Deal is, Volume II has a lower price than Volume I originally had. Not only that, but the price of Volume I has been massively reduced. The two books make an economical alternative to those rare (and unavailable) back issues of Hardcore COMPUTIST.

Volume II ($17.95)

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 5.0 | 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 4.0, 4.0c, 4.1c | the Spy Strikes Back | Time Zone v1.1 | Visible Computer: 6502 | Visidex | Visiterm | Zaxxon | Hayden Software | Sierra Online Software | PLUS the complete listing of the ultimate cracking program...Super IOB 1.5 | and more!

Volume I ($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.

To Order: Send $17.95 + Shipping and Handling for Volume II and/or $12.95 + S&H for Volume I. Shipping and handling is $2.00 per book for US orders, $5.00 per book for foreign orders. U.S. funds drawn on U.S. banks only. Washington State orders add 7.8% sales tax. Send your orders to: **SoftKey Publishing, PO Box 110937-BK, Tacoma, WA 98411**