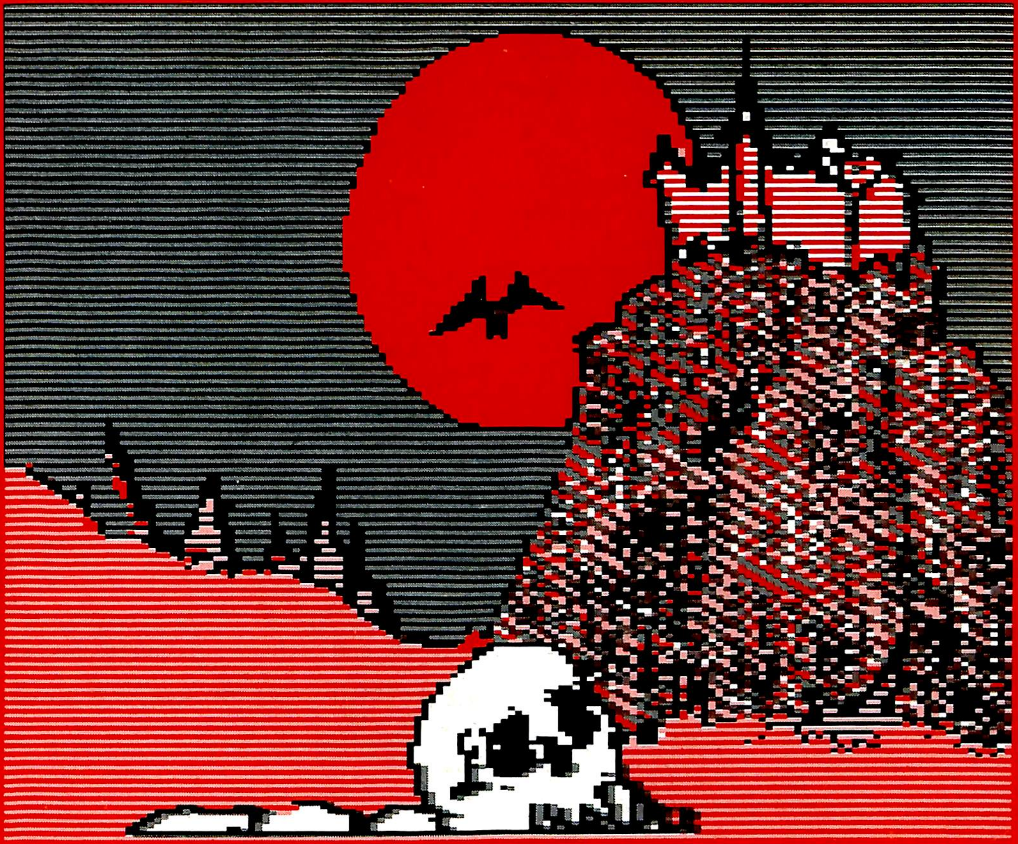
For The Serious User Of Apple][Computers

COMPUTIST

Issue No. 35 September 1986

USA $3.75
Canada/Mexico $7.00
All Others $13.25


(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$\boxed{\text{P}}$

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. 32. 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. 32 as a back issue or order Program Library Disk No. 32.

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 //c 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 (▵). 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 $\boxed{\text{Y}}$.

xxx.xxx$\boxed{\text{Y}}$

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.

Save On Software

Title	Publisher	Suggested Retail	Customer Cost	QTY	Total Cost
Recommended Literature:					
☐ Beneath Apple DOS (Book)	Quality Software	$19.95	$16.00	____	________
☐ Beneath Apple ProDOS (Book)	Quality Software	$19.95	$16.00	____	________
☐ Disk Edit (Book of Softkeys vol 1)	SoftKey		$12.95	____	________
Recommended Software:					
☐ Global Program Line Editor	Beagle Bros	$49.95	$35.25	____	________
☐ Super IOB (Issue No. 32 w/disk)	SoftKey		$10.95	____	________
☐ Magic Window // (specify][or //e)	Artsci	$149.95	$106.00	____	________
☐ Bag of Tricks II	Quality Software	$49.95	$39.75	____	________
Miscellaneous Bargains					
☐ Dazzle Draw	Broderbund	$59.95	$47.50	____	________
☐ F-15 Strike Eagle	Microprose	$34.95	$28.00	____	________
☐ The Print Shop	Broderbund	$49.95	$39.75	____	________
☐ Flight Simulator II	Sublogic	$49.95	$44.00	____	________
☐ Night Mission Pinball	Sublogic	$34.95	$30.75	____	________
☐ Exodus Ultima III	Origin Systems	$59.95	$47.75	____	________
☐ Hitchhiker's Guide to the Galaxy	Infocom	$39.95	$31.00	____	________
☐ Witness	Infocom	$39.95	$31.00	____	________
☐ Dino Eggs	Microlab	$40.00	$20.00	____	________
☐ Zork III	Infocom	$44.95	$35.00	____	________
			Subtotal		________
			Shipping*		________
			Total		________

*Domestic Shipping and Handling: $2.00 per item. Five or more items FREE.

Name____ID#__________

Address ____

City ____________________ State ____________ Zip ______________

Country ____________________ Phone ______________________

VISA MasterCard ____________-____________-____________-____________ Exp. ____________

Signature ____ CP35

To order, complete order form and mail to: COMPUTIST PO Box 110937-SOS Tacoma, WA 98411

Offer good while supplies last. Washington residents add 7.8% sales tax. Foreign orders inquire as to appropriate shipping and handling fees. Limited offer, expires December 31, 1986.

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 ______-______-______-______ Exp. ______
Signature ______ CP35

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 ______-______-______-______ Exp. ______
Signature______ CP35

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 ______-______-______-______ Exp. ______
Signature ______ CP35

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 ______-______-______-______ Exp. ______
Signature______ CP35

Attention COMPUTIST Collectors

The following back issues are practically gone. Once they have sold-out, they will no longer be available in magazine form to our readers.

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 | ....

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 | ....

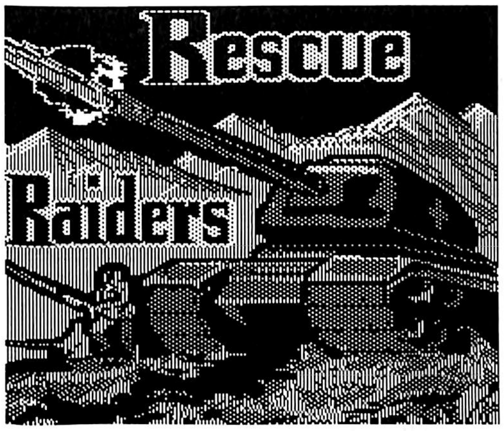
23 *Softkeys* | Choplifter | Mufplot | Flashcalc | Karateka | Newsroom | E-Z Draw | *Readers' Softkeys* | Gato | Dino Eggs | Pinball Construction Set | TAC | The Print Shop: Graphics Library | Death In The Caribbean | *Features* | Using A.R.D. To Softkey Mars Cars | How To Be The Writemaster | *Core*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

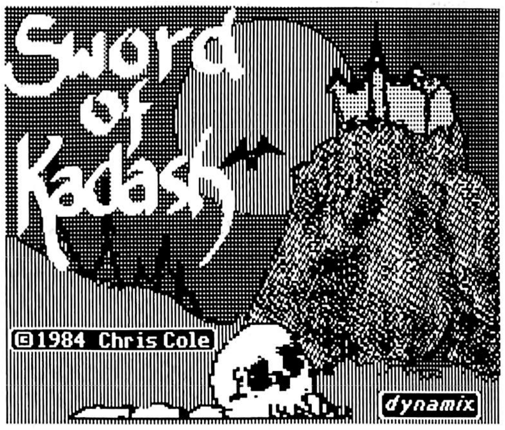
First Come, first served. Some issues may be slightly damaged. No reservations allowed, prepayment only with VISA/MC. To order, phone (206) 474-5750 or use the order form on page 31. For phone orders, please have subscriber ID number and ship-to address ready. Normal back issue rates apply to foreign orders. Washington State residents add 7.8% sales tax.

COMPUTIST

Issue 35 September 1986

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





This month's cover:
Graphics from Penguin's "Sword of Kadash."

Address all advertising inquiries to COMPUTIST, Advertising Department, PO Box 110816, Tacoma, WA 98411. Mail manuscripts to COMPUTIST, PO Box 110846-K, Tacoma, WA 98411.

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.

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.

Another Halley Project

The softkey for the Halley Project you presented a few issues ago didn't work when I tried it, but fortunately I was able to find a new one that worked on my copy.

RUN COPYA

Press **Reset** when the slot and drive questions appear. Now type:

CALL -151
B925:18 60
⌃C
199 POKE 47426,24
249 POKE 47426,56
259 POKE 47426,56
RUN

When finished, run your favorite disk editor and change track $11, sector $0, byte $9A to $1D (as in previous softkey). Done!

Also, for those of you who love to mess around with programs, the initial hi-res picture runs from track $0D, sector $0F through all of sectors $0E and $0F. Unfortunately, this picture is loaded in a very odd way. In even groups of eight lines, the two sets of four are switched! Fortunately, after a bit of sweating, I came up with a routine to fix it. Since the process is fairly simple, the routine is like a toggle. If you run it twice, it will return to its original format. Hard to explain. Just try it. Converts page 2 ($4000) picture to new format on page 1 ($2000).

```
6000:A0 00 84 06 84 08 A9 40
6008:EA EA EA EA 85 07 B2 06
6010:AA A5 07 38 E9 40 EA EA
6018:C9 10 B0 05 18 69 30 90
6020:03 18 69 10 85 09 84 08
6028:8A 92 08 C0 00 D0 05 A5
6030:07 C9 60 D0 01 60
```

1) First, use a Super IOB controller to copy tracks D, E, and F onto a normally initialized disk.

2) Then (this is the tricky part) build a TS list for this hi-res screen just like DOS would; increasing sectors, decreasing tracks. Also, make a catalog entry. Consult *Beneath Apple Dos* for details.

3) Now make and save your own screen on the same disk **under the same name**. This will insure that it falls on the same track.

4) Since DOS inserts 4 bytes at the beginning of each file, move the entire file ($2000-$3FFF) left 4 bytes.

5) Now mess it up with the subroutine above and save it again.

6) Copy it back to the working copy.

7) Using your finest optic abilities (or the little scrap of paper that has the top 4 bytes of the picture), reconstruct these 4 bytes and place them on track $F, sector $F, using your favorite sector editor. (Note: it's usually easier to do this if you intentionally leave the 8th line from the top blank because all you have to do then is just zero those bytes.)

P.S. Those NOP's don't mean a thing. I was revising the code and I didn't want to recalculate all the relative branches. Also, this code is heavily 65C02 dependent.

John Clements
San Francisco, CA

Copy][Plus 6.0 on a Franklin

I recently bought a copy of COPY][Plus Version 6. I have a Franklin Ace 1000 and could not boot this program. I had to modify the code in order to run it. Here's how to get ProDOS Version 1.1.1 or 1.0.1 running on your Franklin Ace 1000 or 1200:

1) Boot the ProDOS system disk. You will see a title screen with Apple's copyright notice.

2) When the system locks up, press the **Reset** key. You will jump into the monitor and see the prompt (*).

3) To start the drive and boot the disk under Version 1.1.1, type:

269E:EA EA
2000G

4) Under Version 1.0.1, type:

265B:EA EA
2000G

You can make a permanent patch to Version 1.1.1 by typing:

BLOAD PRODOS,A$2000,TSYS
CALL -151
269E:EA EA
BSAVE PRODOS,A$2000,L$3A00,TSYS

I hope this helps some of you Franklin Users.

Larry Altuna
Staten Island, NY

Another Blazing Paddles

Was I happy when I received COMPUTIST No. 31, and saw a softkey for Blazing Paddles, thinking I would finally get a good backup. Well guess what happened... that's right, I have a different version from the one you published. However, by using the information from Frank Carone's article and making some educated guesses, I was able to make a COPYA disk. Here's the procedure.

1) Copy the disk with COPYA.

2) Use your favorite sector editor to change:

TRACK	SECTOR	BYTE	FROM	TO
0	$B	13	D0	EA
		14	F7	EA
		2C	D0	EA
		2D	F7	EA

3) Write the change to the disk and you're done!

You have a great magazine - keep up the good work!

Steve Rodgers
Libby, MT

input

Dollars and Sense

Here's a softkey for Dollars and Sense by Monogram.

1) Copy the disk with COPYA.

2) Run a sector editor and go to track $Ø, sector $5. Enter the following values starting at byte $8C:

```
EA A5 Ø2 38 E9 4Ø 85 Ø4 A5 Ø3 E9 ØØ 85 Ø5
AØ 3F B1 Ø4 91 Ø2 88 1Ø F9
```

3) That's it.

Dan Agnew
Fairport, NY

ProntoDOS Procedure

I'm grateful to Tom Weishaar for putting the 65C02/ProntoDOS issue to bed and for letting us know about his newsletter (my check's in the mail).

I have a friend who read about the alleged incompatability of ProntoDOS and his new 65C02 and held off installing the chip - until I told him that there was no problem! Of course, it works fine.

And speaking of my favorite "fast DOS", ProntoDOS - in COMPUTIST No. 3Ø, you published a nice article by Phil Goetz, *Increasing Your Disk Capacity*. In the article, Phil told of a procedure to capture fifteen sectors from track two of a DOS 3.3 disk that might seem slightly messy for a beginner. There is a much easier way to do this if you want to use ProntoDOS. Since ProntoDOS only uses tracks zero, one and sector zero of track two, it is a natural to copy right onto a disk that has been INITed for thirty five tracks. Using this procedure would eliminate the need for step 2 in Phil's article. The procedure would be done after the new disk was INITed in step 5 and would read as follows:

6) Use Copy][Plus to copy ProntoDOS to the new disk. Or use any bit-copy program copying only tracks Ø-3.

The patch in step 3 of the article covers patching the VTOC to show that only sector zero on track two is in use (bytes 4Ø and 41 on track $11 sector $ØØ are changed from their normal values of $ØØ ØØ [no sectors free] to $FF FE [only sector zero in use]).

Anyone can have a disk with only the extra fifteen sectors on track two freed up by INITing a disk with ProntoDOS thusly:

1) Boot a disk with ProntoDOS installed on it.

2) Reset the computer to get the Applesoft prompt.

3) Put a blank disk in drive 1 and type INITHELLO to INIT the disk with a program called HELLO. Note the space after INIT and HELLO was purposely left out as it's optional! And that isn't in any Apple Company publication that I've ever read!

Another hint when using ProntoDOS - a lot of people run into the problem of looking at the Boot Program Name using COPY][Plus. Well, you won't see it, and you get the message, "NO BOOT PROGRAM TO CHANGE" because the Boot Program is not in the usual DOS 3.3 spot on track $1, sector $9 at byte $75. ProntoDOS stores its boot program name at track 1, sector 7, byte $75.

If you want to change the Boot Program Name, you've got two choices. Use a sector editor to change the name. Or, INIT a new disk with ProntoDOS using the desired name instead of HELLO, and copy over all your files from the old disk to the new one.

Finally, what do you do if the desired Boot Program isn't an Applesoft program but a binary program or a file you want to EXEC? Simple, according to *Beneath Apple DOS*, page 7-3, the byte at $9E42 in a 48K DOS 3.3 should be Ø6, 34 or 14 for Applesoft/Integer, binary or EXEC files respectively. This byte is found on the DOS 3.3 disk on track Ø, sector $ØD, byte $42 and on the ProntoDOS disk two sectors behind that at track Ø, sector $ØB, byte $42.

PLEASE publish more COMPUTIST!

Khalid Abu Kabbous
Kingdom of Saudi Arabia

About Rescue Raiders

First, I am very glad there is a magazine for apple users like COMPUTIST. It is the only computer magazine I find worth subscribing to and the only magazine I find myself always anxious to receive in the mail. Keep up the good work. Next, though I'm not much of a hacker, I find I can easily type in codes from your magazine. Here are a few questions that I have. On page 6 of COMPUTIST No. 16, it mentions that to access the cheat mode on Rescue Raiders, you have to type in POPPY. It also mentions that different versions of the game might require different passwords. (On page 4 of COMPUTIST No. 26 it lists the password as ZIPPY). In issue 16, it also says that to find the password for your version of the game, look on track $ØF, sector $9, bytes $A8 to $AC and read backwards. My friend and I both own copies of the game and when we read bytes $A8 to $AC on our disks we both get different garbage, though we both do have BOBBYPIN spelled backwards at bytes $7Ø to $77. But for both of our disks, when we typed the password in, none of the cheat features would work. The cheat features would only work if the disk was deprotected with the method used in issue 16. Then the cheat mode worked fine, but whenever you wanted to fire bullets to the right, the bullets would be pulled up into the air instead of coming down. Does any reader out there know how to fix this problem with their deprotected copy?

Also, I have a copy of Graphics Expander by Springboard that I cannot backup properly. I tried your softkey for Newsroom on page 18 of COMPUTIST No. 23 on it and the program seems to run fine (even the Newsroom demo on it works) until after it checks to see if there's a disk in drive 2. Then the program just reboots. Finally, I have four games for the Apple made by Atarisoft. Galaxian, Jungle Hunt, Gremlins, and Centipede. I've tried everything to copy them but nothing seems to work. Any suggestions?

Besides that, all the other softkeys I've tried from your magazine work. Again, keep up the good work!!!

A Khan
Canada

***Mr Kahn:** Most Atarisoft software can be copied by typing:*

B942:18

and RUNning COPYA.

Silent Service Stuff

I would like to see *Silent Service* by Microprose on your Most Wanted List, as it seems to be impossible to backup with EDD, Copy][Plus or Locksmith, or any other program I have access to.

I tried to deprotect it by making a copy with EDD TØ-T23 and using a sector editor to change bytes starting with CA: from 2Ø 3A DB to EA EA EA 6Ø on tracks 4 and 5 and sector 9 (both tracks) with partial success. This eliminated the message HARDWARE FAILURE but the message YOU HAVE RUN

input

INTO AN ENEMY MINE invariably occurs after about 5 to 1Ø minutes of play. I have never received this message while using the original disk.

It is easy to tell if you are going to get the ENEMY MINE message while the program is loading, (after selecting the level), as it loads with an audible track change.

My manual is marked Change 1, 15 Sept 85. The same as in your article on page 4 of COMPUTIST No. 3Ø. The method by Mr. Lemme does not work on my copy, or perhaps he did not use the copy long enough for the ENEMY MINE message to occur.

I consider your publication the best Apple magazine on the market, with Nibble running second and inCider in third place, and I look forward to each issue of COMPUTIST.

Sheldon M. Atterbury
Sierre Madre, CA

Apple Only?

Since no one seems to be arguing against retaining COMPUTIST as an Apple only magazine, I thought I would try and give a bit of an argument for the other side. My suggestion is that the magazine should be devoted to hackers (in the original sense of computer afficionados rather than vandals) and to open architecture computers. At the present time then this would mean that the magazine would be devoted mostly to Apple with small sections devoted to the IBM PC and clones and perhaps the Amiga.

Don't get me wrong, I like the Apple, and it will be around for a long time. But, nothing lasts forever. The Model T Ford was once the automotive equivalent of the Apple; however a magazine presently devoted to Model T Fords would have relatively limited circulation. People who are seriously interested in computers will naturally progress towards machines that have 16 or 32 bit processors that can access a lot of memory quickly.

I am convinced that before long we will have computers with gigabytes of memory both on optical discs and in RAM that will permit an explosion of innovative programming that will make today's programs seem childish by comparison. I don't believe that Apple will be part of this explosion. There is one further consideration. The Apple market is presently saturated with relatively good programs considering the limitations of the Apples slow processor and small amount of memory. However, the market for IBM PC programs is presently wide open. Anyone who uses a PC knows that there is a tremendous market out there for innovative games that aren't merely re-written from other comuters, and especially for utilities. The utilities presently available for the PC are few in number, greatly overpriced, difficult to use and come with documentation which is verbose, confusing and incomplete.

I might add that there is also a terrific market out there for people who can crack IBM programs so that, for instance, they may be installed on a hard disk. (Jeff (my brother) tells me that most IBM programs are pretty easy to crack.)

If the magazine were expanded by one double page for the IBM and otherwise remain the same size, I think that the interests of the die-hard Apple fanatics could be served as well as the interests of those of us who wish to move on to more powerful machines.

David W. Rivett
Canada

About Autoduel

This is probably a rather strange letter. It's a partial softkey for Autoduel by Origin Systems. The softkey for Autoduel is very similar to the Ultima IV softkey by Mike Roetman in COMPUTIST No. 28. Autoduel performs fine when you use this softkey except for when you wish to go to the Arena feature in the game. The disk just whirls on doing nothing. Maybe you or some other readers can solve this problem.

1) INITialize a disk with:

```
INIT HELLO
```

2) Run Super IOB v1.5 with the Ultima IV controller in COMPUTIST No. 28.

3) type in this program.

```
1Ø PRINT CHR$(4) "BRUN B□AT"
```

4) Save this as your hello program on copy of Autoduel.

```
SAVE HELLO
```

5) Make the following change to file called "S□AP"

```
BLOAD S□AP
CALL -151
11A1:EA EA EA
11A6:EA EA EA
11B8:B7
11BA:E8
BSAVE S□AP,A$8ØØ,L$CØØ
```

Dan Agnew
Fairport, NY

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

Kevin Sartorelli's softkey for...

Pooyan

Datasoft, Inc.
19808 Nordoff
Chatsworth, CA 91311

Requirements:
A blank disk
At least 48K

This method requires a RAM card to run the game as the title picture is stored there. If there is no RAM card the game will still run but there will be rubbish instead of the title picture. The softkey is given in a cookbook fashion to save space as the boot code trace is quite long.

The first boot stage starts off with an illegal opcode that ignores the first two bytes ($801 and $802). The rest of the boot trace you should be able to follow, should you feel like it, by liberal use of the monitor "L" command.

1) Boot a DOS 3.3 disk. Type **FP** to clear any BASIC program.

2) Insert a blank disk in the drive and type

```
INIT HELLO
```

to create a slave disk with a null (empty) greeting program (needed later in the softkey).

3) Go into the Monitor.

```
CALL -151
```

4) Insert the POOYAN disk into drive one.

5) Move the initial boot code from ROM to RAM.

```
6600<C600.C6FFM
```

6) Load the first boot stage from the disk.

```
66F8:0
6600G
```

7) Turn off the drive.

```
C0E8
```

8) Alter the boot code.

```
6659:20
66F8:4C
2000<800.8FFM
```

9) Change where the code is stored.

```
209D:CA
20A0:A
20B3:0
```

10) Decode the second boot stage to $A00.

```
2097G
```

11) Prepare to use our modified boot code.

```
89B:0A EA
85E:EA
```

12) Modify the boot code to jump to our routine.

```
A6C:4C 00 B7
```

13) Modify the next boot stage to jump back to our routine when finished.

```
B700:A9 4C 8D DA 04 A9 00 8D DB
B709:04 A9 B8 8D DC 04 4C 84 04
```

14) Add code to save the text page.

```
B800:A0 00 A2 04 B9 00 04 99
B808:00 34 C8 D0 F7 EE 06 B8
B810:EE 09 B8 CA D0 EE AD 82
B818:C0 4C 59 FF
```

15) Load in the main game and save the text page.

```
6600G
```

16) Turn off the drive again.

```
C0E8
```

17) Move some code to a safe place.

```
A000<2000.26FFM
```

18) Clear the screen and HOME the cursor.

```
FC58G
```

19) Load in the title picture.

```
400<3400.37FFM
2B:60
819CG
```

20) Move half the picture to $5000 and the other half up to $3000.

```
5000<3000.3FFFM
3000<2000.2FFFM
```

21) Restore the code moved in step 17, and pack other code into unused space.

```
2000<A000.A6FFM
2700<8D00.95FFM
4500<8A00.8CFFM
```

22) Type in some code to get the picture from the language card.

```
819C:AD 80 C0 A9 E0 8D AF 81
81A4:A9 20 8D B2 81 A2 20 A0
81AC:00 B9 00 E0 99 00 20 C8
81B4:D0 F7 EE AF 81 EE B2 81
81BC:CA D0 EE AD 82 C0 AD C4
81C4:C0 EE F4 03 60
```

23) Type in code to move the code to the correct places in memory and put the picture in the RAM card.

```
4800:AD 81 C0 AD 81 C0 A2 10
4808:A0 00 B9 00 30 99 00 E0
4810:C8 D0 F7 EE 0C 48 EE 0F
4818:48 CA D0 EE A2 10 B9 00
4820:50 99 00 F0 C8 D0 F7 EE
4828:20 48 EE 23 48 CA D0 EE
4830:AD 82 C0 A2 09 B9 00 27
4838:99 00 8D C8 D0 F7 EE 37
4840:48 EE 3A 48 CA D0 EE A2
4848:03 B9 00 45 99 00 8A C8
4850:D0 F7 EE 4B 48 EE 4E 48
4858:CA D0 EE AD 57 C0 AD 54
4860:C0 AD 52 C0 AD 50 C0 4C
4868:00 60
```

24) Add a jump to our routine.

```
CFD:4C 00 48
```

25) Boot the slave disk made at the beginning.

```
BSAVE POOYAN,A$CFD,L$7D03
```

26) Type **BRUN POOYAN** to play the game.

P. J. Thompson's softkey for...

The Perfect Score

Mindscape Software
P.O. Box 506
Northampton, MA 01601

Requirements:
Apple][
COPYA
A sector editor
Twelve blank disk sides

The Perfect Score is a huge program written in the FORTH language for high school students preparing for the SATs. The verbal sections are quite easy while the math is fairly difficult. It is reasonably priced and not heavily protected. Although a bit copier cannot copy it, modification of the protection code will wipe out any problems in backing up the program. Here it is step-by-step:

1) Load COPYA and change the RWTS to ignore errors:

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

2) Copy all twelve sides of The Perfect Score. This may take a while because the sectors are written in a funny order.

3) Start up your sector editor and make this patch on each disk:

```
Track  Sector  Byte  To:
-------------------------
 $0     $9     $40  $4C
 $0     $9     $41  $D4
 $0     $9     $42  $02
```

That's all! This skips the protection loaded upon booting and is used constantly throughout the program. Enjoy this program and hope for a perfect score.

Charles Taylor revisits...

Alice in Wonderland

Windham Classics
One Kendall Square
Cambridge, MA 02139

Requirements:
Super IOB v1.5
One blank two-sided disk

After reading Allan Migdal's softkey for Alice in Wonderland (COMPUTIST No. 29), I realized that I had a different version. The main difference is that all files are not CATALOGable by normal methods. Also, the boot program is HELLO, not STARTUP.

Step by Step

1) Install the Super IOB controller below and use it to copy both sides of Alice in Wonderland. The address and data epilogues have been changed from DE AA to FF FF.

2) Copy DOS from your DOS 3.3 system master to the boot side of the copy of Alice. Use Super IOB to copy tracks 0-2 or use something like Copy II Plus' "Copy DOS" option. That's all!

I would like to present two better methods of "BRUNning STARTUP" than the method used by Mr. Migdal in his softkey:

1) Change the "hello" program with Copy][Plus. This automatically makes the DOS changes necessary to BRUN programs on booting.

2) Or, boot the DOS 3.3 master. Then type

POKE 40514,52

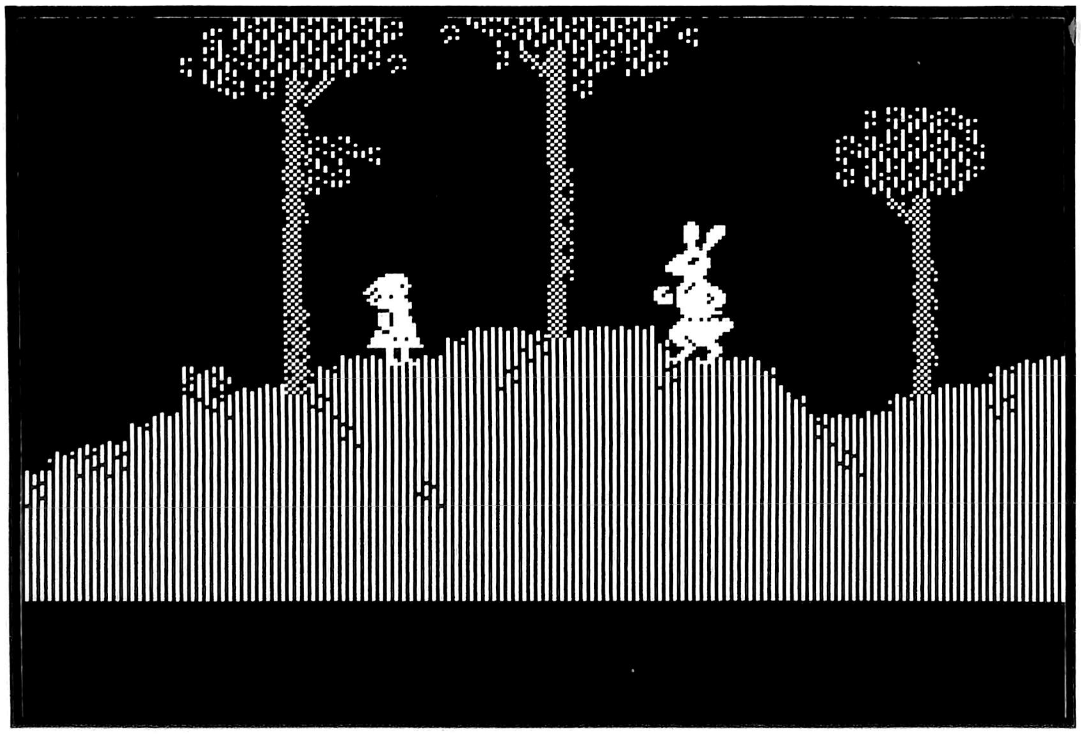
and INIT a disk with the name of the binary "hello" program. To EXEC a text file as the "hello" program, use POKE 40514,20.

controller

```
1000 REM ALICE CONTROLLER #2
1010 TK = 0 : LT = 35 : ST = 15 : LS = 15 : CD = WR : FAST
     = 1
1020 GOSUB 490 : RESTORE : GOSUB 170 : GOSUB 610
1030 GOSUB 490 : GOSUB 230 : GOSUB 610 : IF PEEK
     (TRK ) = LT THEN 1050
1040 TK = PEEK (TRK ) : ST = PEEK (SCT ) : GOTO 1020
1050 HOME : PRINT "COPYDONE" : END
1060 DATA 255 ,255 ,255 ,255
```

controller checksums

```
1000  - $356B     1040  - $44C8
1010  - $2544     1050  - $9B42
1020  - $F372     1060  - $BA53
1030  - $A259
```



P. J. Thompson's softkey for...

The Money Manager

Sterling Swift Publishing Co.
7901 South IH-35
Austin, TX 78744

Requirements:
Apple][computer
COPYA or equivalent
A sector editor
Two blank disks

The Money Manager is a simulation of business made for educating the user. Although the copy protection is not extensive, it isn't easy to bit copy without parms. It checks for a totally illegal byte between two of the address headers. If we can avoid this routine, the program would be unprotected.

1) Load COPYA or any other sector copier and copy both disks on to your backup.

2) Load up your sector editor and make the following patches to both disks:

```
Track  Sector  Byte  From  To:
------------------------------
$00    $0B     $58   $4C   $EA
$00    $0B     $59   $E5   $BD
$00    $0B     $5A   $BC   $8C
```

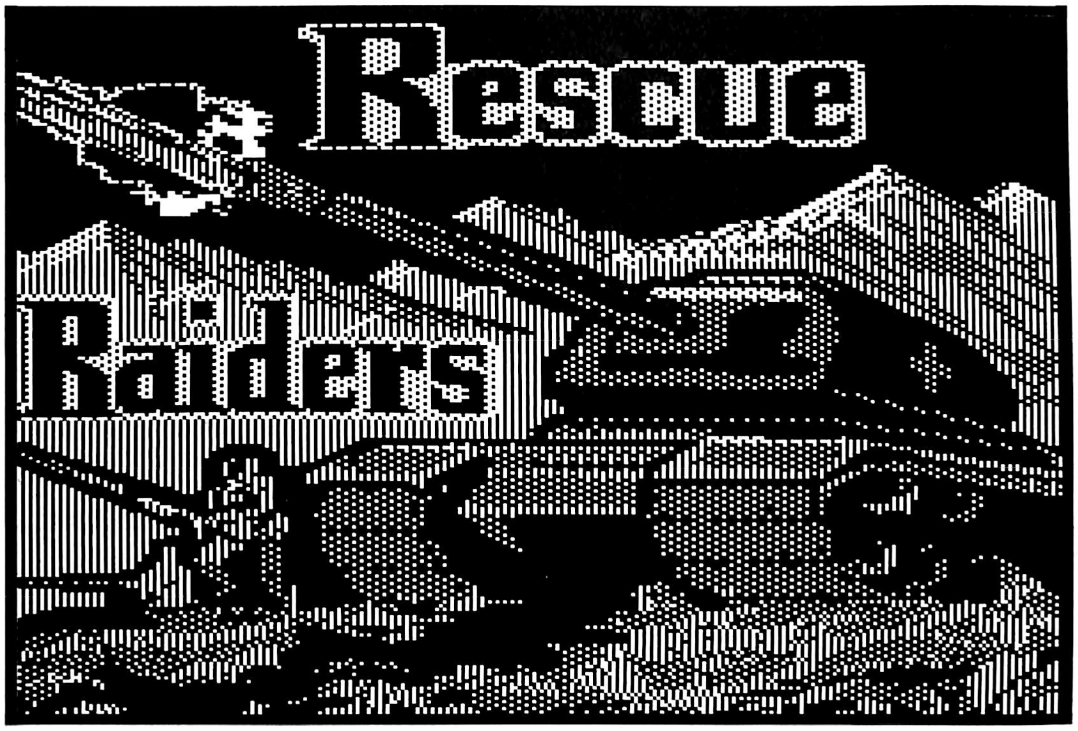
P. J. Thompson's softkey for...

Good Thinking!

Hoffman Educational Systems

Requirements:
64K Apple][
A sector editor
Twelve blank disk sides

readers' softkey & copy exchange



Good Thinking! is a new program written in Pascal designed to improve language among young kids. It can be copied normally but upon bootup the copy will not be accepted. After scanning the disk for a BD 8C C0 (LDA $C08C,X), which is a common instruction used to read bytes off the disk, I made a small patch which bypasses the protection. This softkey is for Level One programs.

Here it is step-by-step.

1) Load COPYA or any other full disk copier.

2) Copy all twelve sides of the Good Thinking! disks. If you want to save a little room you can double side your disks.

3) Boot up your sector editor.

4) On each disk make the following patch at byte $B6 from $4C to $60. This will be done on a different sector for each disk. Here is a list of the disks and the corresponding sector to modify:

Disk	Trk	Sect
CAUSE & EFFECT	$1C	$6
DRAWING CONCLUSIONS	$20	$8
PREDICTING OUTCOMES	$1F	$A
GENERALIZATIONS	$1E	$A
MAIN IDEA	$1F	$A
TOPIC SENTENCES	$20	$4
CLASSIFYING	$1E	$6
COMPARISON	$1F	$E
ANALOGIES	$1D	$6

If these sectors do not contain a 4C F8 00 (JMP $00F8) at byte $B6 then it is the wrong sector. Either check again or search your disk for a BD 8C C0. Make sure the sector in which you found those bytes are on a track greater than $3 and then do your patch. On the first two tracks these bytes are used constantly for the disk operating system. Hopefully you won't have to go through this mess but the softkey should work either way.

Steve and Rod Smith's patch for...

Mockingboard Rescue Raiders

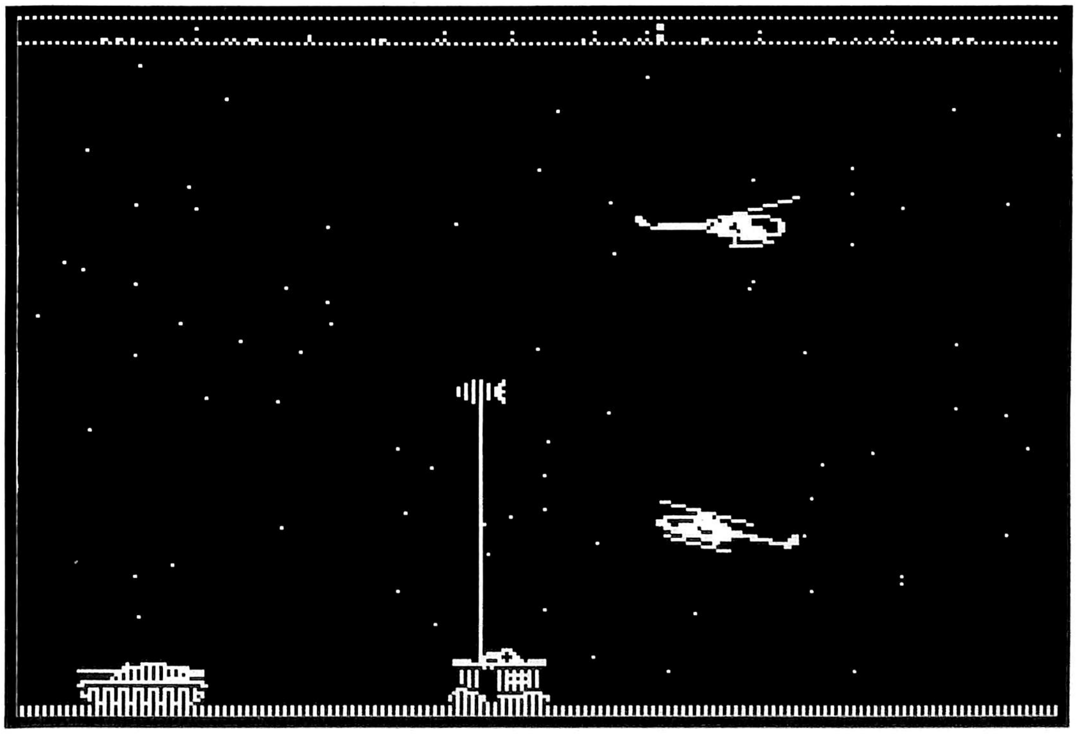
Requirements:
COPYAable version of Rescue Raiders
Sector Editor

Those of you who own Mockingboards may have been a bit frustrated when after following the softkey for Rescue Raiders in COMPUTIST No. 16. When I tried it, I found that the program froze at the main screen.

Well, as it turns out, the program checks for a Mockingboard, but does not do anything with it after it's found one, except hang. Here is an easy patch to apply to your backup that will eliminate the check and play the game without having to remove your Mockingboard.

With a sector editor, apply these changes to your backup:

Track	Sector	Byte	Change to
$15	$01	$02	$C0
$15	$01	$05	$60



Softkey for...

Hi-Res

by William Hinger

On-Line Systems

Requirements:
Hi-Res Cribbage
A blank disk

This game was released in 1980, and at that time there were a few holdouts who were still using DOS 3.2 in their machines. It is not surprising, therefore, that the disk was released in 13-sector format with a special boot sector for DOS 3.3 systems. In fact, the disk has a completely normal DOS, and if you interrupt the loading process with a Ⓒ C, you can catalog the disk and snoop through all the files there.

If you load and list the Applesoft file named CRIBBAGE, you will see that it first loads and runs a binary file named HIRES. This file turns on the hi-res graphics and draws the title page. This file is not necessary as it is later completely overwritten when the next file is loaded.

A binary file named BCRIBBAGE is then loaded at address $D00 with a length of $7FFF, which means that it extends in memory to $8CFF. (Locations $AA72 and $AA73 in DOS contain the start address of the most recently loaded binary file, and locations $AA60 and $AA61 contain the length of the most recently loaded program or file.)

Then a binary file called EXTRA is loaded at $8D00 with a length of $600. They obviously could have made these one file with a simple patch to the save length parameter in DOS (A964:FF lets you BSAVE more than 32K at a time), but they didn't.

The last file loaded is a binary file named IF which is loaded at $300 with a length of $A. After a couple of pokes, this file is called and starts the program. The first instruction in this file is a JSR $7D64. This routine simply clears all memory between $800 and $CFF to zeros. The next instruction, JSR $8700, is the one which prints the menu and executes the program.

Now, what we would like to do is save these files as one binary file which we can FID to another disk and execute with a BRUN. This will work as long as we have the original disk in the drive. But if we have transferred the files to another disk with a file transfer utility (such as MUFFIN to transfer the files to a 16 sector disk), the program will try to access the disk and die a fast death in the middle of the menu. So we need to look a little further.

If you look at the routine which starts at $8700, you can see that it repeats a cycle where it sets VTAB and CH and then calls a routine (at $7F27) which prints a line to the screen. This continues until just before it would print menu choice 5. Then instead of a JSR to $7F27, there is a JSR to $8FE0. This is the protection routine, such as it is. It first does a call to a subroutine at $8F00 which unscrambles the code at $9000, calls the code at $9000, rescrambles the code at $9000 with another call to $8F00, and then finally modifies its own code so that the next time the menu is printed, the call to $8FE0 encounters a JSR $7F27 and RTS.

Examination of the code at $9000 will show that the program first loads track 0, sector 0 into page 3 and then track 2, sector C into page 91. Since $9100 is in the middle of the EXTRA file, we can save this with the rest, but we need to make arrangements to save the code at $300 inside of our file. We can do this by moving it to $C00 and then writing an entry routine which will reposition this code before calling the program.

One more point needs to be addressed here. The menu choices which save a match and continue a previously saved match need to be eliminated. When I chose to update this program, I wanted it to be able to execute under either DOS 3.3 or ProDOS. These routines read and write sectors directly to disk using direct calls to DOS 3.2 RWTS routines, and since a match is less important to me than just playing the game, I chose to eliminate these routines entirely. This involved changing some of the ASCII in page $80 (memory $8000) so that these choices no longer show up, and then changing the program so that it will not accept these choices. This is where most of the code you will need to enter to unprotect this program will be placed.

I then wrote a short routine which moved the code to page 3, set the graphics display, set $4A and $4B in zero page to the values originally set by the Applesoft program, and then jumped to $300 to start the program. After saving the code as a binary file, I tried running it. Voila! When I ran the program, it looked great and ran fine on my old Apple][. But before considering the task to be complete, I usually try it on the kids' //c also. Good thing I did. The program died.

Now the only thing that I could think of that would allow the program to run on my antique, yet would not allow it to run on the new //c would normally involve direct calls to a part of the monitor ROM which was changed during the various incarnations of the Apple][family. So I traced through about 10 routines to find what I thought I was looking for. The culprit was found at $7E54 and started with an LDA $E006. This was used as an ID byte to determine what flavor of BASIC was resident. In my old machine, Applesoft would return $00 and Integer would return $85. Since I do have Applesoft ROMs in the machine, this would return a $00 and all would be fine. But $E006 on the //c would return a $89 and the program would mistake this for Integer BASIC and die. Since I would always be running the program on a machine containing Applesoft, I disabled

Cribbage

the choice by having the routine load $00 into the accumulator instead of loading an ID byte. It now worked fine on both of my machines. If by chance you are still using an Apple][with the Integer ROMs, you need to skip the step which changes the code at $7E54. The step by step process follows, so do and enjoy!

Step by Step

1) Initialize a blank DOS 3.3 disk with an empty HELLO program:

```
NEW
INIT HELLO
```

2) Insert your original Cribbage disk and boot it:

```
PR#6
```

3) When the program has displayed the entire menu and the disk drive has quit spinning, interrupt the program by pressing Reset.

4) Enter the monitor:

```
CALL -151
```

5) Protect page 3 of memory from the boot:

```
C00<300.3FFM
```

6) Now boot the slave disk you created in step 1:

```
6 OpenApple P
```

7) Modify the menu to delete the SAVE MATCH and CONTINUE MATCH options:

```
CALL -151
8088:CF D0 C5 D2 C1 D4 C9 CE
8090:C7 A0 C9 CE D3 D4 D2 D5
8098:C3 D4 C9 CF CE D3 A0 A0
80A0:A0 8D B3 A0 AD A0 C5 CE
80A8:C4 A0 A0 A0 A0 A0 A0 A0
80B0:A0 A0 A0 A0 A0 A0 A0 A0
80B8:A0 A0 A0 A0 8D A0 A0 A0
80C0:A0 A0 A0 A0 A0 A0 A0 A0
80C8:A0 A0 A0 A0 A0 A0 A0 A0
80D0:A0 A0 A0 A0 A0 A0 A0 8D
80D8:A0 A0 A0 A0 A0 A0 A0 8D
```

8) Delete the jump to the protection routine:

```
87BA:27 7F
```

9) Modify the choice subroutine to reflect the new menu:

```
87C4:B4
87E0:FF
87E7:FF
87EE:02
8808:BF 9D
```

10) Enter a short startup routine:

```
BC0:8D 50 C0 8D 52 C0 8D 54
BC8:C0 8D 57 C0 A9 0C 85 00
BD0:85 01 85 02 C6 00 D0 FC
BD8:C6 01 D0 F8 C6 02 D0 F4
BE0:A2 00 BD 00 0C 9D 00 03
BE8:CA D0 F7 A9 00 85 4A A9
BF0:08 85 4B 4C 00 03 00 00
```

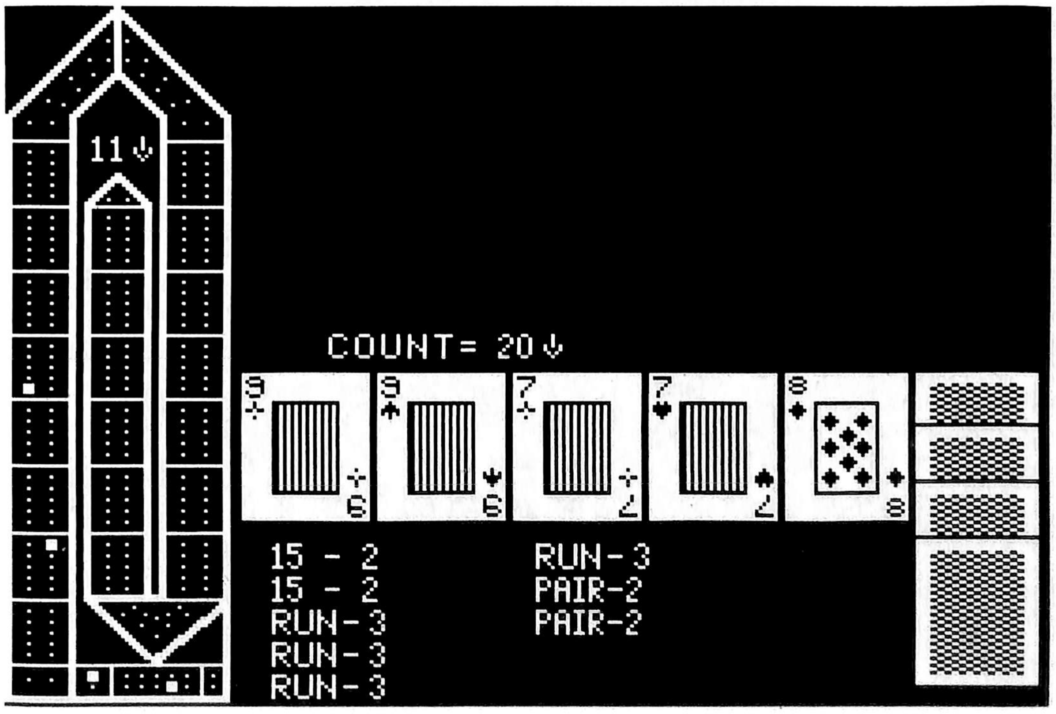
11) If your machine is a][Plus or newer, or if you have replaced the Integer ROMs in your][with FP BASIC ROMs, make the following change:

```
7E54:EA A9 00
```

12) Save the new file:

```
A964:FF
3D0G
BSAVE CRIBBAGE,A$BC0,L$8740
```

That's all there is to it. This 138 sector binary file can be moved using FID from your DOS 3.3 System Master.



softkey for...

Olympic

by Marc Lirette

Microsoft Corp.
10700 Northop Way
POB 97200
Bellevue, WA 98009

Requirements:
Super IOB v1.5
Apple][Plus or equivalent
One disk drive (two are preferred)

After several attempts at trying to unprotect my Olympic Decathlon disk I was only able to unprotect it last week. The main reason why I couldn't unprotect Olympic Decathlon was because is uses an old system of storing the information on the disk. After boot tracing and examining the read routine, I noticed that the tracks on the disk were written with 4 & 4 encoding. Standard DOS 3.3 and ProDOS uses 6 & 2 encoding. Refer to Beneath Apple DOS or ProDOS by Don Worth and Peter Lechner for a discussion of 4 & 4 encoding versus 6 & 2.

Because the disk uses 4 & 4 encoding, it can only store 11 sectors per track.

The method of unprotecting the disk is to first tranfer the disk to a DOS 3.3 format (6 & 2 encoding) and then change its read routine to use RWTS instead of reading 4 & 4. First I used the good old boot trace to capture the Olympic Decathlon read routine. Then I wrote a Super IOB v1.5 controller that uses the Olympic Decathlon's own read routine and normal DOS for writes. The resulting disk would not boot.

After examining the boot process of the original disk some more I decided that I would have to write my own loader routine and modify the read routine on the copy of Olympic Decathlon to use standard RWTS. So after some revisions to my controller and the creation of my LOADER file, it worked!

The Cook Book

1) Key in the LOADER hexdump and the Super IOB controller using the instructions on the inside front cover of this magazine and save them with:

```
BSAVE LOADER,A$9000,L$6D
SAVE CON.OLYMPIC
```

2) We will now boot code trace your original Olympic Decathlon disk to capture its read routine. First write protect your original and enter the monitor.

```
CALL-151
```

3) Cpoy the disk controller card ROM in slot 6 to RAM.

```
9600<C600.C6FFM
```

4) Change this relocated boot ROM to put us into the monitor instead of executing the next stage of the boot.

```
96F8:4C 59 FF
```

5) Execute this modified Boot 0 and turn off the drive.

```
9600G
C0E8
```

6) The disk drive will start up and clank. The cursor will then reappear. Now track 0, sector 0 is loaded into memory. Change Boot 1 at $0800 to put us into the monitor instead of executing Boot 2.

```
826:4C 59 FF
```

7) Change Boot 0 to load track 0, sector 0 into $9800 and execute Boot 1 at $0800.

```
9659:98
96F8:4C 01 08
```

8) Execute Boot 0 and 1 and turn off the drive.

```
9600G
C0E8
```

9) Now Boot 2 is loaded into memory. This is Olympic's read routine. Boot your Super IOB disk and save this read routine.

```
C600G
BSAVE RWTS.OLYMPIC ,A$8900,L$900
```

10) We will now create a slightly modified boot strap code to load in Olympic Decathlon. Enter

Decathlon

the monitor and move the boot strap code to $3600.

```
CALL -151
3600<B600.BFFFM
```

11) Make the following modifications to this code so that it will load Olympic Decathlon correctly.

```
36FE:B6
3715:01
371A:0F
373F:81 C0 20 93 FE
3748:00 89
37A0:C0 0A D0 05 A0 0F
37AC:EE
37E0:09
37E7:8A
37EB:00
37F0:00
```

12) Return to Applesoft and save this modified boot strap to your Super IOB disk.

```
BSAVE RWTS.NORMAL ,A$3600,L$A00
```

13) Use the Super IOB controller to copy Olympic Decathlon. Note that after track $22 is written, you will be asked to insert your Super IOB disk. This is so the controller can BLOAD the files you just created and write them to your deprotected copy.

LOADER

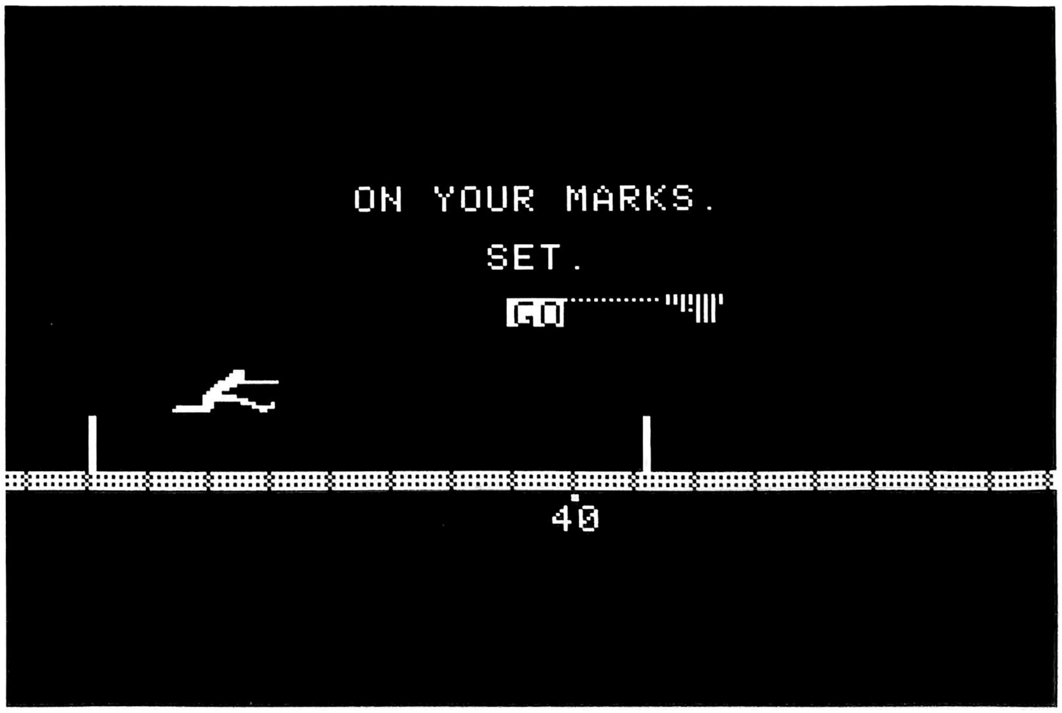
```
9000: A5 0C 4A 8D EC B7 A5 0D    $18DD
9008: 8D ED B7 A5 08 8D F1 B7    $DB8F
9010: A9 00 8D F0 B7 A5 03 85    $53AB
9018: 02 E6 03 EE F1 B7 EE ED    $04DC
9020: B7 AD ED B7 C9 0B D0 08    $980D
9028: A9 00 8D ED B7 EE EC B7    $9F92
9030: C6 02 A5 02 D0 E5 AD EC    $73A4
9038: B7 85 05 AD ED B7 85 06    $D544
9040: A9 01 8D F4 B7 A9 B7 A0    $8FA0
9048: E8 20 B5 B7 AC ED B7 88    $76B1

9050: 10 05 A0 0A CE EC B7 8C    $B7E7
9058: ED B7 CE F1 B7 C6 03 A5    $47D2
9060: 03 D0 E2 A5 05 0A 85 0C    $B8BC
9068: A5 06 85 0D 60             $5BE7
```

controller

```
1000 REM OLYMPIC DECATHLON
1010 FOR A = 936 TO 960 : READ X : POKE A , X : NEXT
1020 TK = 1 : ST = 10 : LT = 35 : LS = 10 : CD = WR : POKE
     900 , 10 : POKE 907 , 206 : POKE 916 , 144
1030 GOSUB 490 : POKE 955 , TK * 2 : POKE 941 , 22
     + 66 * (TK < 33 ) : CALL 936
1040 IF TK = 9 THEN FOR A = 91 TO 93 : POKE 10752
     + A , 234 : NEXT : POKE 10850 , 0
1050 GOSUB 490 : FOR A = 0 TO 7 : POKE BUF , 49 + 11
     * A : MB = 39 + 11 * A
1060 GOSUB 610 : TK = TK + 1 : IF TK < LT THEN NEXT
     : GOTO 1030
1070 A$ = CHR$ (7 ) + "INSERT^ SUPER^ IOB^ DISK"
     : GOSUB 470 : HOME : PRINT
1080 PRINT CHR$ (4 ) "BLOAD^
     RWTS.OLYMPIC.A$2700" : PRINT CHR$ (4 )
     "BLOAD^ LOADER.A$2E00"
1090 PRINT CHR$ (4 ) "BLOAD^
     RWTS.NORMAL.A$3000"
1100 A$ = "INSERT^ TARGET^ DISK^ IN^ DRIVE^ " +
     STR$ (D2 ) : GOSUB 470 : HOME
1110 ST = 9 : TK = 0 : POKE BUF , 57 : MB = 48 : GOSUB
     610 : POKE 907 , 238
1120 POKE 916 , 176 : ST = 15 : POKE BUF , 44 : MB =
     48 : GOSUB 610 : TK = 1 : POKE BUF , 39 : MB =
     44 : GOSUB 610
1130 HOME : PRINT "THAT'S^ ALL^ FOLKS!" : END
5000 DATA 169 , 96 , 133 , 11 , 169 , 77 , 133 , 3 , 169
     , 39 , 133
5010 DATA 8 , 169 , 0 , 133 , 7 , 133 , 13 , 169 , 0 , 133
     , 12 , 76 , 0 , 144
10010 PRINT CHR$ (4 ) "BLOAD^ RWTS.OLYMPIC"
```

controller checksums

```
1000  - $356B      1090  - $DE25
1010  - $EC74      1100  - $643E
1020  - $F80E      1110  - $035A
1030  - $EAA7      1120  - $24A3
1040  - $5610      1130  - $4697
1050  - $DC04      5000  - $E913
1060  - $FB0F      5010  - $A9BE
1070  - $E857      10010 - $CDE4
1080  - $730A
```



F-15

by Jim Wallace

MicroProse
12Ø Lakefront Drive
Hunt Valley, MD 21Ø3Ø

Requirements:
A blank disk
Super IOB 1.5
Sector editor (optional)
A copy program which will copy specific tracks (optional)

As Michael Ferreira and Ken Burnell (Input, COMPUTIST Nos. 28 and 29) indicate, there are other versions of F-15 Strike Eagle released which are different from Larry Jasonowicz's (COMPUTIST No. 24). I also tried Larry's softkey procedure with no success. His article was helpful, however, in softkeying the F-15 I have. Perhaps some of my findings will help in softkeying still other versions.

A quick check of the disk using CIA revealed a normal DOS 3.3 format except track $22 which appeared to be empty anyway. So I made a copy of F-15, tracks $ØØ through $21. I used this copy during my investigation and used the original for the final softkey.

I decided to look for a catalog track, whose location is usually indicated in normal DOS 3.3 on track $Ø1, sector $ØB, byte $Ø1. Finding the value $15, I initialized a blank disk with DOS 3.3, used a sector editor to change byte $Ø1 of sector $ØB, track $Ø1, to $15 and then rebooted this disk. The F-15 copy could then be CATALOGed and the files loaded.

I was interested in examining any file which loaded into page $Ø2, assuming that the protection would at least be similar to Larry's version. The file "AS", located on track $1Ø, sectors $ØC and $ØD, loaded into page $Ø2. After loading it, I found that it also stored $CØ at $Ø2Ø1 if a copy of the disk is detected and $DB if no error is encountered. To defeat this check, I simply changed the $CØ to $DB (two locations (track $1Ø, sector $ØC, bytes $9Ø and $9E) on the copy of F-15 which I had just made).

Having made these changes, the game would progress up to the point of requesting you to enter a code number (explained in the F-15 manual). After the code was entered, the program bombed with a "HARDWARE FAILURE!" message. Noticing that disk access occurred after the code was entered, I suspected a nibble count or some unique signature on that problem track $22. So I rebooted and interrupted the process with a modified ROM (see previous issues of COMPUTIST) to search for disk access code (some reference to location $CØ8C). This led me to the actual nibble count subroutine located from $53Ø3 through $537B. (I later checked the catalog and found file "SCN.DTA" loaded into pages $5Ø through $53. However, don't bother trying to load and read it (it's encoded and gets Exclusive-OR'ed after loading to reveal sensible data)). Backtracking, I found the process begins at $5ØØØ. Without any further analysis of the code, I simply gave it a shot of "18 6Ø" (encoded, of course) to indicate "no errors found" and return to the caller. No luck. Apparently there was something the subroutine generated which the caller looked for upon return.

Looking through the routine, I found a couple of places where $21 is loaded and then the program jumps to what appeared to be its own SEEK subroutine (to move the disk arm to a specific track. See COMPUTIST No. 22, Super IOB 1.5 "Half-Tracks" for more on this subject).

```
$5ØØØ- 18          CLC
$5ØØ1- 9Ø ØF       BCC $5Ø12
$5ØØ3- 9A           ..
$5ØØ4- 96 21        ..
$5ØØ6- Ø1 6Ø        ..
$5ØØ8- D5 AA        ..
$5ØØA- 96 DE        ..
$5ØØC- AA           ..
$5ØØD- D5 AA        ..
$5ØØF- AD DE AA     ..
.
.
.
$5142- AD Ø5 ØØ    LDA $5ØØ5
$5145- 2Ø 89 52    JSR $5289
$5148- AD Ø5 5Ø    LDA $5ØØ5
$514B- 2Ø 9B 51    JSR $519B
$514E- 2Ø 7E 51    JSR $517E
```

When a program uses a SEEK subroutine, 9.5 times out of 1Ø (decimal--not hex) it's using half-tracks. A quick check of the original disk revealed a fully formatted track $21.5 with seemingly empty sectors. As a matter of fact, tracks $21, $21.5, and $22 appear identical--and I do mean identical! Track $22 is formatted with a track ID of $21!

After locating these tracks, the actual nibble count subroutine counts a specific number of bytes, then looks for a "9A 96 xx DE AA" sequence.

```
$532A- AØ 54       LDY #$54
$532C- A9 ØØ       LDA #$ØØ
$532E- BD 8C CØ    LDA $CØ8C.X
$5331- 1Ø FB       BPL $532E
.... Read $54 + $FF bytes:
$5333- EA          NOP
$5334- 88          DEY
$5335- DØ F7       BNE $532E
$5337- BD 8C CØ    LDA $CØ8C.X
$533A- 1Ø FB       BPL $5337
$533C- EA          NOP
$533D- C8          INY
$533E- DØ F7       BNE $5337
$534Ø- BD 8C CØ    LDA $CØ8C.X
$5343- 1Ø FB       BPL $534Ø
$5345- A4 2A       LDY $2A
$5347- DØ Ø5       BNE $534E
$5349- CD Ø3 5Ø    CMP $5ØØ3
$534C- DØ 2C       BNE $537A
$534E- BD 8C CØ    LDA $CØ8C.X
$5351- 1Ø FB       BPL $534E
```

Strike Eagle

```
$5353- A4 2A       LDY $2A
$5355- D0 05       BNE $535C
$5357- CD 04 05    CMP $5004
.... Read and compare the next five
     bytes to "9A 96 xx DE AA":
$535A- D0 1E       BNE $537A
$535C- BD 8C C0    LDA C08C,X
$535F- 10 FB       BPL $535C
$5361- EA          NOP
$5362- EA          NOP
$5363- EA          NOP
$5364- BD 8C C0    LDA C08C,X
$5367- 10 FB       BPL $5364
$5369- CD 10 50    CMP $5010
$536C- D0 0C       BNE $537A
$536E- BD 8C C0    LDA $C08C
$5371- 10 FB       BPL $536E
$5373- CD 11 50    CMP $5011
$5376- D0 02       BNE $537A
$5378- 18          CLC
$5379- 60          RTS
$537A- 38          SEC
$537B- 60          RTS
```

Hmmmm. The DE AA is the epilog of a normal DOS 3.3 sector. Checking the sectors on these tracks again with CIA Linguist revealed the "9A 96 9A DE AA" at the end of the sector $00 on each of these similar tracks. It's easily overlooked in the sea of "96's" unless you're looking for it.

Well, if I couldn't ignore the whole disk check routine, at least I could skip the actual nibble count. At $5151 the program branches to $51EE if a good nibble count has occurred and steps are executed to indicate no error. The sole indication of no error is that the A register must contain $00 when returning to the caller. The remaining code between $5165 through $517A will set up other conditions needed upon return whether an error was detected or not.

With this information, I decided to change the code on page $51 as needed to skip the actual nibble count. As mentioned earlier, this routine is encoded on the disk. Each page of the routine must be EOR'ed with a different value for decoding. In case any of you want to experiment further, they are listed below. CAUTION: The bytes you see in RAM are offset by a factor of +4 when using your sector editor. For example, when you read in page $50 from track $0C, sector $07, byte $5020 that you see in RAM will be byte number $24 of your sector editor--not byte number $20. When DOS 3.3 stores binary files to the disk, it precedes the actual data with a two-byte address and a two-byte length. This "moves" the rest of the data four bytes down the file.

Page	Value
$50	$A5
$51	$75
$52	$39
$53	$6E

(One item I didn't follow through on was figuring out why immediately after decoding, page numbers in the code are $60 through $63, then changed to $50 through $53 prior to execution. Another Caution: Take heed, ye who want to experiment further!)

Making The Copy

The sector editor method for making a F-15 backup is faster. Nearly everyone has a sector editor, but for anyone who does not, use the Super IOB method described later.

1) Copy the F-15 disk, tracks $00 through $21, or, if you can't specify tracks, ignore errors on track $22.

2) Use your sector editor to make the following changes:

Track	Sector	Byte	From	To
$C	$06	$31	$55	$39
		$32	$87	$2D
		$5C	$55	$6D
		$5D	$53	$DC
		$5E	$17	$6E
		$5F	$39	$F8
		$60	$37	$93
$10	$0C	$90	$C0	$DB
		$9E	$C0	$DB

3) Be sure to write each sector back after making the changes.

The Super IOB Method

1) Initialize a blank disk with DOS 3.3.

2) Install the F-15 controller at the end of the article into Super IOB and use it to copy tracks $00 through $21 of the original disk. Super IOB will make the necessary sector edits on tracks $0C and $10.

controller

```
1000 REM F-15 STRIKE EAGLE CONT.
1010 TK = 0 : LT = 34 : ST = 15 : LS = 15 : CD = WR : FAST
     = 1
1020 GOSUB 490 : GOSUB 610 : T1 = TK : TK = PEEK (TRK
     ) - 1 : RESTORE : GOSUB 310 : TK = T1
1030 GOSUB 490 : GOSUB 610 : IF PEEK (TRK ) = LT
     THEN 1050
1040 TK = PEEK (TRK ) : ST = PEEK (SCT ) : GOTO 1020
1050 HOME : PRINT "DONE." : END
5000 DATA 11^ CHANGES
5010 DATA 12 ,6 ,49 ,57 ,12 ,6 ,50 ,45
5020 DATA 12 ,6 ,51 ,20 ,12 ,6 ,92 ,109
5030 DATA 12 ,6 ,93 ,220 ,12 ,6 ,94 ,110
5040 DATA 12 ,6 ,95 ,248 ,12 ,6 ,96 ,147
5050 DATA 12 ,6 ,97 ,20 ,16 ,12 ,144 ,219
5060 DATA 16 ,12 ,158 ,219
```

controller checksums

```
1000  - $356B      5010  - $9E8E
1010  - $0715      5020  - $6508
1020  - $CE56      5030  - $D7AB
1030  - $7CE8      5040  - $1CA8
1040  - $D1CC      5050  - $3F5D
1050  - $33EF      5060  - $8768
5000  - $E34A
```

Exploring ProDOS by Installing a...

CPS Clock Driver

by Paul Blumstein

If you own a Mountain Computer CPS clock card, this article will show you how you can use it as a ProDOS compatible clock. If you do not have a CPS clock, this article will provide you with insight into some of the inner workings of ProDOS that you will find useful to you whether you wish to write assembly language programs or as a starting point for further exploration. We shall explore ProDOS's Machine Language Interface and its Global Pages and we will discover how a system program is called and relocated.

Background

About four years ago, Mountain Computer Incorporated revolutionized the Apple compatible peripheral card industry by introducing the "CPS Multifunction Card." This card contained a bi-directional serial interface, a parallel interface and a clock/calendar. Its name, CPS, stands for its three functions: Clock, Parallel and Serial. This card was popular for two years until ProDOS arrived, making the clock function of little value. Many people still are using these cards to drive their modem or printer or both.

Why A Clock?

An excellent reason for owning a ProDOS compatible clock is that ProDOS will automatically place the date and time in directory entries. That information tells you when each file was first created and when it was last modified. Have you ever had two or more files with the same name on two different disks and wondered which one was the latest version? Or have you ever needed to know whether your backup files were current? Having a clock solves these problems.

Another good reason to own a clock is that some applications will use your clock to make your life a little easier. For example, Apple's AppleWorks program initially prompts you for a date. If you have a clock, it defaults to the current date. If you don't have a clock, then it defaults to the last time you manually entered a date. Without a clock, this means that you will have to re-enter a new date every time you boot on a different day. With a clock, AppleWorks will also display the time that your files were last updated in addition to the date.

Word Juggler, a word processor by Quark Incorporated, is another good example. It has functions that automatically print the date and time on your documents. When you use these functions without a clock, you must manually enter the date and time each time you boot. When you do this, the time never changes. More and more ProDOS applications are being written to make use of a clock.

It is especially frustrating when you have a clock inside your computer that you can't use. This frustration led me to create a clock driver that would access a CPS clock and fool ProDOS into thinking that I had a ProDOS compatible clock. By the way, you probably have noticed the name Thunderclock mentioned in ProDOS-related literature. That is because Apple designed ProDOS to be compatible with the Thunderclock clock card made by Thunderware, Inc. All of the clock cards since that time have been designed to be Thunderclock compatible.

How a Program Accesses the Date

The creators of ProDOS decided to create subroutines that could be accessed in a way that would be compatible across future versions. One of the problems with DOS 3.3 was that programmers accessed its subroutines directly. That meant that Apple could not make any real modifications to DOS since everyone expected that everything would always be in the same place. ProDOS takes two steps to solve this problem: the Machine Language Interface (MLI), and Global Pages.

The MLI is a subroutine that has only one entry location: $BF00. An assembly language programmer executes a JSR instruction (similar to a BASIC GOSUB or CALL statement) to this location and passes parameters indicating what he would like ProDOS to do for him. For example, each of the ProDOS commands (like OPEN) have equivalent MLI calls. There are additional system functions that are only available through the MLI. The MLI call to get the system date and time (called GET__TIME), which we are about to explore, is one example of this.

The System Global Page is page $BF (locations $BF00 through $BFFF). This page contains values, such as file buffer addresses, and JMP (jump) instructions (which are equivalent to BASIC's GOTO statements). These instructions jump to various areas of ProDOS. The BASIC Interpreter (which is the interface between ProDOS and Applesoft) also has a Global Page (page $BE). By having fixed locations on these Global Pages, both ProDOS and the BASIC Interpreter can come out with new versions without and any program that does not violate the rules will not be affected. New versions of ProDOS and the BASIC Interpreter merely change the values stored in these locations, but the locations, themselves, never change. If you are interested in more details on the MLI and these Global Pages, I recommend Beneath Apple ProDOS by Don Worth and Pieter Lechner (Brady Communications, Inc., 1985, $19.95).

That brings us to the clock. When any ProDOS program wishes to retrieve the date and/or time, it does it by issuing an MLI call (JSR $BF00). By convention, this call is followed by a one byte function code and a two byte address that points to the location of a parameter list. When ProDOS is done with the call, program execution continues at the address immediately following the three bytes. The GET_TIME function code is $82. Since this call does not have any parameters, the two byte address after the function code has a value of zero. The assembly language code to get the system date and time is simply:

```
JSR $BF00     call the MLI
.HS 82        GET.TIME code
.HS 0000      address of parmlist
LDA...        continue program
```

ProDOS responds by placing the current date and time, in binary, in locations $BF90 through $BF93. The time is stored in a 24-hour clock format with hours placed in location $BF93 and minutes placed in location $BF92. The date is more complex. The 5 right-most bits (known as the Least Significant Bits or LSBs) of location $BF90 contain the day of the month. Since we number bits from 0 (on the right) to 7 (on the left), this would be bits 0 through 4. The number of years since 1900 (i.e., 1986=86) occupy bits 1 through 7 of location $BF91. That leaves bits 5 through 7 of $BF90 to contain the LSBs of the month and bit 0 of $BF91 to contain the Most Significant Bit (MSB) of the month.

Confused? Let's try an example using September 15, 1986 as today's date and 11:45 pm as the current time and figure out how this is stored (see Table 1). If you were to examine locations $BF90 and $BF91 with the monitor, you would see that it contains the values $2F and $AD, respectively, which is consistent with the bit pattern above. If the system doesn't contain a compatible clock, the values at these locations do not change. They are initially zero and are changed whenever you "set the clock". That is why you can set a non-existent clock and retrieve the non-changing date and time until you re-boot.

The Clock Driver

Since different ProDOS versions can have the clock driver code in different locations, it needs a way of knowing where the driver currently is. Remember the Global Page? Locations $BF07 and $BF08 contain the current address of the clock driver. (Always remember that an address has the low-value byte first). When ProDOS first boots, it looks for a compatible clock. If it finds one, it does two things. First, it places a $4C (JMP instruction) at location $BF06. Then it modifies the driver so that the clock addresses within the driver point to the correct slot. If a compatible clock isn't found, it places a $60 (RTS instruction) at $BF06 instead.

When a program (or ProDOS itself) calls the MLI (at $BF00) to retrieve the date, it uses a function code of $82. This tells it that it needs to execute the clock driver code. The MLI now does a JSR (Jump to SubRoutine) to $BF06. If a clock isn't present, the RTS (ReTurn from Subroutine) instruction is executed and execution passes back to the MLI. If a clock is present, then the JMP (JuMP) instruction directs the processor to the driver code. The driver code accesses the clock, fills in locations $BF90 through $BF93, and performs an RTS which passes execution back to the MLI. The MLI then returns to the program.

Solving the Big Problem

Those of you that have a CPS Clock realize that ProDOS doesn't recognize it. Even if it thought that the clock was compatible, the driver wouldn't work. That means that we must derive a way to fool ProDOS into thinking that we have a compatible clock and make it use a different driver. The problem wouldn't be too hard to solve if we were only going to use the clock for our own applications. In that case, all we would need to do is BLOAD our own driver into a location that know we is safe and change locations $BF06 through $BF08 to be a JMP to our driver. Then, anytime ProDOS or our programs needed the time, it would use our driver to access our clock and everything would work just fine.

But, we also want our store-bought programs to use the clock. This is tricky for two reasons. First, we never know what parts of memory the application will use, perhaps all of it. Second, most applications do not have a BASIC Interpreter and do not allow us to leave the application itself.

There is a solution! After ProDOS boots, it checks the catalog for the first system program with a name ending with ".SYSTEM". By the way, a system program is one whose file type is "SYS". It is always a binary file that will run at $2000. At the end of the boot, ProDOS loads this program and passes execution to it with the JMP command. On our ProDOS User's Disk, this program is called BASIC.SYSTEM. On an application program, it could be called anything. The solution is to rename the .SYSTEM program so that ProDOS won't find it and will find our own program, CLOCK.SYSTEM, instead.

CLOCK.SYSTEM contains our homebrew clock driver and code to install it. The installation code first changes the RTS instruction at $BF06 to be a JMP instruction so that ProDOS knows that a clock exists. Next, it changes all of the absolute addresses within the driver to be consistent with where we are going to move it. Then, it relocates the driver to the only safe place that we can move it: over the area where the old driver was.

The final thing that we will do is execute the system program that would normally be executed right after ProDOS booted: the one we renamed. This gets more complex. CLOCK.SYSTEM would have to relocate itself somewhere else since the other system program also executes at $2000 and we would overwrite ourselves if we loaded it. Also, BLOAD and BRUN aren't currently available for us to use. That would mean that we would have to write code to manually search the catalog and manually load it. All of this could be done, but would require a lot of extra work.

Luckily, there is an easier solution. ProDOS contains a section of code called QUIT which is accessed through the MLI. The QUIT code is designed to be used at the end of a system program to let the computer user execute another system program. By letting ProDOS do the work, we can save ourselves a lot of trouble. There is one problem though. The QUIT code requires you to enter the name of the next system program that you want to execute. That means a few extra keystrokes everytime you boot. Whenever I rename the .SYSTEM program, I call it "GO". That cuts down on the number of keystrokes. Also, by always using the same name, I do not have to remember what I called it on each disk.

A Guided Tour

The source code listing puts into practice everything that we have talked about. Most people place their CPS Cards into either slot 1 or 2. This listing is assembled for slot 2. If your card is in slot 1 (or any other slot, for that matter) a few minor changes are needed. If you are using an assembler, simply change the "SLOT .EQ 2" in the source code listing (next page) from a 2 to a 1 (or whatever the slot number is) and everything will assemble correctly. If you are going to type the program in through the monitor, change the $C2 to $C1 (or C followed by the slot number) at the following hex locations: 205E, 209D, 20A4, 20A7, 20BA, and 20BD.

Since this is a system program, it must reside at address $2000. The driver code follows the installation code which is executed at boot time.

There are several ways to access the clock through assembly language. I chose the most direct method, which bypasses the built-in ROM program on the card and accesses the clock directly. The clock is accessed by writing commands to its address location and reading data from (or writing data to) its data location. These locations are $CxFE and $CxF9, respectively, where x is the slot number.

Since time is always moving, we must hold the clock briefly while we read it. If we didn't and it was, say, 2:59, we may retrieve the minutes and the clock may then become 3:00 just before we get the hours. We would then think that the time was 3:59. This hold function is performed at $205A and after completion, the clock is released at $2099. The rest of the program gets each part of the date that it needs and stores it at the previously mentioned locations.

The RDCLK subroutine retrieves the tens place of each field, multiplies it by ten, then retrieves the ones place and adds it to the tens place. The multiplication is done by shifting two places to the left, adding back in the original number and shifting the result one place to the left. For example, if we are working on the minutes field and it is now 2:59, we will retrieve a 5 for the tens place. Shifting to the left is the equivalent of multiplying by two since we have a binary machine. The first shift turns the 5 into a 1Ø ($A). The second shift turns it into a 2Ø ($14). Adding back in the original number, 5, gives us 25 ($19) and the last shift turns this into a 5Ø ($32).

We then retrieve the ones position, which is a 9, and add it to the 5Ø to give us 59 ($3B). Each value that we retrieve is returned from the clock in the rightmost nibble, except for the tens place of hours, where we only need the two LSBs. In that case we perform an AND instruction to mask out those bits.

The installation code changes $BFØ6 to be a JuMP instruction and then calculates where RDCLK will reside after relocation. It then changes each of the JSRs to RDCLK in the driver so that they will point to its new location. The next step is to bank in the language card RAM and overlay the new clock driver on top of the old driver. Finally, the installation code calls QUIT through the MLI (at $2Ø4C).

Installing Our Masterpiece

Either type in the machine code through the monitor or type in the assembly code (this is in S-C Assembler format). If you use an assembler under DOS 3.3 simply convert the resultant binary file with the ProDOS CONVERT Utility. To make life simpler, call your program something other than CLOCK.SYSTEM (such as CPS.DRIVER) because it is not yet a system program. (ProDOS won't let you BLOAD a system program so you would want to keep a binary version of it around for future use). Put your program onto your ProDOS User's Disk and place your target disk into drive 2. (You can also do this with one drive). Rename the current .SYSTEM program and save your program as a .SYSTEM program. For example:

1) Make a backup copy of the disk you intend to install the clock driver on. Put away the original. NEVER modify your original disk.

2) Boot ProDOS and enter BASIC.

3) Enter the monitor with CALL -151 and type in the hexdump (if you don't assemble it).

4) Save the program to disk.

BSAVE CPS.DRIVER,A$2ØØØ,L$C3

5) Locate the .SYSTEM program and rename it "GO."

RENAME BASIC.SYSTEM,GO

Table 1: Time/Date Storage

FIELD	DECIMAL	HEX	# BITS	BINARY	COMMENT
Month	9	9	4	1ØØ1	see below
Day	15	F	5	Ø1111	see below
Year	86	56	7	1Ø1Ø11Ø	see below
Hour	23	17	8	ØØØ1Ø111	Placed at $BF93
Minute	45	2D	8	ØØ1Ø11Ø1	Placed at $BF92

```
<----$BF91----> <----$BF9Ø---->    ADDRESS
7 6 5 4 3 2 1 Ø 7 6 5 4 3 2 1 Ø    BIT #
1 Ø 1 Ø 1 1 Ø 1 Ø Ø 1 Ø 1 1 1 1    BIT VALUE
<---YEAR----> <-MON-> <--DAY-->     FIELD NAME
```

Source Code For CLOCK.SYSTEM

```
*            CLOCK.SYSTEM -- USE THE MCS CLOCK WITH PRODOS           *
*             REMEMBER TO RENAME THE ORIGINAL .SYSTEM FILE           *
*                       P. BLUMSTEIN -- 1/86                         *
                   .OR $2ØØØ
ØØ3A- TENS         .EQ $3A      KEEP A WORKSPACE LOC
ØØ1Ø- READ         .EQ $1Ø      CLOCK READ COMMAND
ØØØ2- SLOT         .EQ 2
C2FE- CLKADR       .EQ SLOT*$1ØØ+$CØFE
C2F9- CLKDATA      .EQ SLOT*$1ØØ+$CØF9
ØØ3A- TEMP         .EQ $3A
BFØØ- MLI          .EQ $BFØØ    MLI ENTRY POINT
      START
2ØØØ: A9 4C                LDA #$4C     SETUP GLOBAL JUMP
2ØØ2: 8D Ø6 BF             STA $BFØ6
2ØØ5: 18                   CLC
2ØØ6: AD Ø7 BF             LDA $BFØ7    CALC ADDRESS OF RDCLK
2ØØ9: 69 46                ADC #RDCLK-CLKSTART
2ØØB: 85 3A                STA TEMP
2ØØD: AD Ø8 BF             LDA $BFØ8
2Ø1Ø: 69 ØØ                ADC #Ø
2Ø12: 8D 63 2Ø             STA JSR1+2   SET UP RELOC INFO
2Ø15: 8D 6E 2Ø             STA JSR2+2
2Ø18: 8D 77 2Ø             STA JSR3+2
2Ø1B: 8D 7F 2Ø             STA JSR4+2
2Ø1E: 8D 92 2Ø             STA JSR5+2
2Ø21: A5 3A                LDA TEMP
2Ø23: 8D 62 2Ø             STA JSR1+1
2Ø26: 8D 6D 2Ø             STA JSR2+1
2Ø29: 8D 76 2Ø             STA JSR3+1
2Ø2C: 8D 7E 2Ø             STA JSR4+1
2Ø2F: 8D 91 2Ø             STA JSR5+1
2Ø32: AD Ø7 BF             LDA $BFØ7
2Ø35: 85 3A                STA TEMP
2Ø37: AD Ø8 BF             LDA $BFØ8
2Ø3A: 85 3B                STA TEMP+1
2Ø3C: AD 8B CØ             LDA $CØ8B    ALLOW WRITE TO RAM
2Ø3F: AD 8B CØ             LDA $CØ8B
2Ø42: AØ 69                LDY #CLKEND-CLKSTART #BYTES TO MOVE-1
      LOOP
2Ø44: B9 59 2Ø             LDA CLKSTART.Y OVERLAY CLOCK DRIVER
2Ø47: 91 3A                STA (TEMP).Y
2Ø49: 88                   DEY
```

```
204A: 10 F8              BPL LOOP
204C: 20 00 BF           JSR MLI
204F: 65                 .HS 65      EXECUTE QUIT CODE
2050: 52 20              .DA PARMS
      * * * END OF DRIVER INSTALLATION CODE * * *
2052: 04        PARMS    .HS 04      INDICATE # PARMS
2053: 00                 .HS 00
2054: 00 00              .HS 0000
2056: 00                 .HS 00
2057: 00 00              .HS 0000
      * * * BEGINNING OF DRIVER * * *
      CLKSTART
2059: B8                 CLV
205A: A9 40              LDA #$40    HOLD THE CLOCK
205C: 8D FE C2           STA CLKADR
205F: A9 03              LDA #3      MINUTE ADDRESS
2061: 20 9F 20  JSR1     JSR RDCLK
2064: 8D 92 BF           STA $BF92   STORE MINUTES DIRECTLY
2067: A9 05              LDA #5      HOURS ADDRESS
2069: 2C 06 BF           BIT $BF06   SET V FLAG...
206C: 20 9F 20  JSR2     JSR RDCLK   ...HOURS GET SPEC HANDLING
206F: 8D 93 BF           STA $BF93
2072: B8                 CLV
2073: A9 0C              LDA #12     YEAR ADDRESS
2075: 20 9F 20  JSR3     JSR RDCLK
2078: 8D 91 BF           STA $BF91   STORE IT UNSHIFTED
207B: A9 0A              LDA #10     MONTH ADDRESS
207D: 20 9F 20  JSR4     JSR RDCLK
2080: 4A                 LSR         SHIFT INTO PLACE
2081: 6A                 ROR
2082: 6A                 ROR
2083: 6A                 ROR
2084: 8D 90 BF           STA $BF90   PUT WHATS LEFT WHERE
2087: AD 91 BF           LDA $BF91   GET THE YEAR BACK
208A: 2A                 ROL         ...& STICK IN THE MONTH'S
208B: 8D 91 BF           STA $BF91   ...& PUT IT BACK
208E: A9 08              LDA #8      ADDRESS OF DAY
2090: 20 9F 20  JSR5     JSR RDCLK
2093: 0D 90 BF           ORA $BF90   PUT THE MONTH IN & ...
2096: 8D 90 BF           STA $BF90   ...PUT IT BACK
2099: A9 00              LDA #0      TAKE CLOCK OFF HOLD
209B: 8D FE C2           STA CLKADR
209E: 60                 RTS
      * * * RDCLK--ON ENTRY 'A' CONTAINS A CLOCK FIELD ADDRESS
*                      ON EXIT 'A' CONTAINS ITS BINARY VALUE              *
      RDCLK
209F: 09 10              ORA #READ   PUT READ FLAG INTO ADDRESS
20A1: AA                 TAX         HOLD THE ADDRESS
20A2: 8D FE C2           STA CLKADR  GET TENS PLACE
20A5: AD F9 C2           LDA CLKDATA
20A8: 50 02              BVC NORM    SKIP THIS MASK EXCEPT..
20AA: 29 03              AND #3      ..HOURS NEEDS 2 BITS
      NORM
20AC: 29 0F              AND #$F     OTHERWISE, 1 NIBBLE
20AE: 85 3A              STA TENS
20B0: 0A                 ASL         MULTIPLY BY TEN
20B1: 0A                 ASL
20B2: 65 3A              ADC TENS
20B4: 0A                 ASL
20B5: 85 3A              STA TENS
20B7: CA                 DEX         POINT TO ONES PLACE
20B8: 8E FE C2           STX CLKADR  AND GET IT
20BB: AD F9 C2           LDA CLKDATA
20BE: 29 0F              AND #$F     KEEP A NIBBLE
20C0: 65 3A              ADC TENS
20C2: 60        CLKEND   RTS
```

6) Install the new clock driver as a .SYSTEM file. The TSYS command tells ProDOS that the file type we're working with is .SYS ("Type SYStem"). This will keep ProDOS from complaining.

BLOAD CPS.DRIVER
CREATE CLOCK.SYSTEM,TSYS,D2
BSAVE CLOCK.SYSTEM,TSYS,A$2000,L$C3

The hexdump contains the program in binary with the checksums created by CHECKBIN. You can run CHECKBIN under ProDOS by using the ProDOS CONVERT Utility to make it a ProDOS file.

Always perform the above on a backup disk only, never on the original. When you boot the modified disk, it will prompt you: ENTER PREFIX. Hit RETURN at this point. It will then prompt you for the program name. Type in GO (in upper or lower case) and hit RETURN. The program will then proceed as it always did before. If you mistype the program name, the QUIT code will say FILE/PATH NOT FOUND and stop. Don't panic. Simply hit Reset and it returns to the ENTER PREFIX prompt.

Advanced Improvements

Two additional features can be added to this program. If you need to make the installation code work with the clock in any slot, you can add code to find the slot that your clock is currently in. Your CPS Setup Disk has a BASIC program called CPS SLOT FINDER that can easily be converted to assembly language. To see this program, boot the Setup disk, quit into BASIC, type in NEW, then EXEC CPS SLOT FINDER and LIST.

If you do not like to type GO every time you boot, you can add code to relocate the installation code, load the GO program at location $2000 and JMP to it. Personally, I find both of these features more trouble than they are worth.

CLOCK.SYSTEM Hexdump

```
 2000: A9 4C 8D 06 BF 18 AD 07   $C6BE
 2008: BF 69 46 85 3A AD 08 BF   $3DAE
 2010: 69 00 8D 63 20 8D 6E 20   $2409
 2018: 8D 77 20 8D 7F 20 8D 92   $2F9E
 2020: 20 A5 3A 8D 62 20 8D 6D   $DE61
 2028: 20 8D 76 20 8D 7E 20 8D   $0763
 2030: 91 20 AD 07 BF 85 3A AD   $92C5
 2038: 08 BF 85 3B AD 8B C0 AD   $6E88
 2040: 8B C0 A0 69 B9 59 20 91   $36A6
 2048: 3A 88 10 F8 20 00 BF 65   $BF93

 2050: 52 20 04 00 00 00 00 00   $5437
 2058: 00 B8 A9 40 8D FE C2 A9   $60A6
 2060: 03 20 9F 20 8D 92 BF A9   $3A0F
 2068: 05 2C 06 BF 20 9F 20 8D   $F0B2
 2070: 93 BF B8 A9 0C 20 9F 20   $3615
 2078: 8D 91 BF A9 0A 20 9F 20   $C14E
 2080: 4A 6A 6A 6A 8D 90 BF AD   $31C4
 2088: 91 BF 2A 8D 91 BF A9 08   $B76D
 2090: 20 9F 20 0D 90 BF 8D 90   $56D2
 2098: BF A9 00 8D FE C2 60 09   $D3DA

 20A0: 10 AA 8D FE C2 AD F9 C2   $BD89
 20A8: 50 02 29 03 29 0F 85 3A   $870C
 20B0: 0A 0A 65 3A 0A 85 3A CA   $7F6E
 20B8: 8E FE C2 AD F9 C2 29 0F   $75EB
 20C0: 65 3A 60                  $F385
```

Putting a New F8 On Your Language Card

by Ken Burnell

Requirements:
Apple Language Card
Low wattage soldering iron
Small diameter solder
Small diameter wire strap
Apple][Plus or equivalent computer (not //e or //c)
2716 EPROM
Exacto knife or equivalent
Access to an EPROM burner

Note: The following procedure involes modification of your motherboard and/or your Language Card. COMPUTIST will not be held responsible for any damages incurred while following this procedure.

Ever since I read an article in "Computers & Electronics" on custom programming Apple EPROMs (reference #1), and bought an inexpensive EPROM burner, I have been burning custom F8 EPROM's for my Apple, and enjoying it more. (I had previously built a ROM/EPROM adaptor socket for the Apple motherboard (reference # 2)). Therefore, when I read Ray Darrah's article in COMPUTIST No. 19 (reference # 3) I happily set out to round up a 2732 EPROM and miscellaneous small parts and program up a new, improved, expanded super F8. But, after re-reading Ray's article completely through again, I realized that I didn't really want to fool around with adding an extra chip to my virgin motherboard, cutting traces, etc.

So, I shelved the project (with several others) until I could find a better way. The procedure I finally came up with assumes that you have an Apple Language Card, APPLE Product #A2B0006 or equivalent. The way to spot this card is that it has a short sixteen-wire cable extending from a DIP socket on the card to a RAM socket on the motherboards (Apple says to use the RAM socket at location E8 but replacing other RAM chips in other locations will also work). Most important of all, this card has an an F8 ROM chip (usually a 9316) mounted near the rear of that card.

There are a few things you should know about the Apple Language Card and its ROM chip. First, the ROM on the card effectively replaces the F8 ROM on the motherboard. In fact, with the Language Card installed, you can remove the F8-ROM altogether on the motherboard (with the power turned OFF, of course!) and the computer will still work just fine. But the second, and more important fact is that the Apple Language Card, like the Apple Integer Card (remember those?), has two sets of solder pads. When these pads are modified, the substitution of a programmable 2716 EPROM for the standard 9316 non-programmable ROM supplied with the card is allowed.

The Way

The procedure, which references the sketch on page 21, requires the use of an exacto knife or equivalent, a small (1/4 inch or so) piece of wire (28 to 30 gauge, solid copper if you have it) and a low temperature soldering iron. *Note: Using a standard "soldering gun" type iron on any printed circuit board could damage the board.*

For printed circuit (PC) work, I use a fifteen to thirty watt iron, maximum, with small diameter resin core solder. I have found that solder diameter size # 30 works fine for the what we will be doing.

The procedure for modifying the Language Card to accept EPROMS instead of ROMs is as follows:

1) Using the knife, cut the copper circuit trace at the X-shaped pads marked "2716" as shown on the sketch at point A. Only a small, deep, cut is needed to sever the trace. If you have a continuity tester (or VOM meter), place it on a low-resistance setting and use it to test for no continuity between the cut traces after performing the surgery.

2) Place a short piece of small diameter copper wire across the O-shaped pads marked "2716" at point B on the sketch. Solder the jumper across the pads, joining the pads.

3) Install your EPROM being sure that the notch is pointed in the same direction (usually up) as the ROM you're replacing.

4) Re-install the card and its jumper cable back onto the motherboard, and you're done. The next time you boot up your computer, it will boot from your new EPROM on the Language Card.

Burning EPROMs

At the end of this article, I've listed several references to various articles and published letters previously written about burning EPROM's for Apple][computers. One of the problems I had, initially, with this process was that the authors told how to modify the

EPROM's code, and how to write binary programs of them, but not how to get the binary code into the EPROM's!

Fortunately, the BSAVE and BLOAD commands of DOS helps to make this job easy. For example, to save the contents of the F8 ROM ("uploading" in computer talk), you would type

```
BSAVE STDF8,$AF800,L$800
```

where "STDF8" is the file name I chose, $F800 is the beginning address in memory and $800 is the length of the file ($F800 through $FFFF in the ROM). This would create a binary file of the code in the F8 ROM (which, by the way is printed in Apple's reference manual in an appendix (reference # D)).

Later, when you want to transfer the binary file on disk to an EPROM in your EPROM burner, it is first necessary to select an address in RAM for the file to go. I usually use address $1000 for ease of making address calculations. Then, transfer the file from disk to the memory of the computer ("downloading"). The command to use here is BLOAD and it would look like this:

```
BLOAD STDF8,$A1000
```

Now, you would typically transfer control to your EPROM burner and instruct it to burn an EPROM using the data starting at $1000.

EPROM's And Special Keyboards

There are several keyboards around that are claimed to be exactly "plug compatible" with the standard Apple keyboard - that is, you can unplug your Apple keyboard at its sixteen-pin plug (better note the orientation of that plug or bad things may happen) without using a special adaptor board. But, if the replacement keyboard has lower case capability, you probably won't see the lower case characters on your monitor! This is because the Apple's F8 ROM converts the lowercase characters to uppercase before putting them on the screen.

Well, we can't have that can we? In Mr. Mitchell's article (reference # 1) he told how to fix this problem by modifying the code in a new F8 ROM, and he also told how to do some other interesting things like having your name displayed on the computer's monitor when it boots up instead of Apple's, etc. I strongly recommend Mr. Mitchell's article to anyone wishing to customize ROM's in his Apple][. In COMPUTIST No. 9, (reference # 4) an the author told how to modify ProDOS 1.0.1 to run on Franklin Ace and other Apple clones. Recently, Beagle Brothers (reference # 5) gave a tip on how to modify ProDOS 1.1.1 in much the same manner.

Some software checks for a code at location $FD83 in the F8 ROM. This code indicates that a particular computer (][+, //e, Franklin, other) is being used and then bombs if it doesn't like the answer. Likewise, the software may checksum-check the whole ROM. One solution to this problem is to use Ray Darrah's 2732-chip-with-a-switch trick (reference # 3) and switch between a standard F8 image and your custom one in either half of the 2732 - mounted on your motherboard, Integer Card, Language Card or... The variations are limited only by your imagination and pocket book.

The Integer Card

I mentioned earlier that the Apple Integer Card might also be modified to take 2716 EPROM's. Well, it can, in much the same way as the Language Card - by changing the wiring on two sets of O-shaped and X-shaped pads as above. Then the five ROM's - D0, D8, E0, E8, F0 and F8 can all be replaced with 2716 EPROM's.

Now, suppose you want to program often-used utility software into memory locations $D000 through $F7FF, ordinarily taken up by Applesoft in the][+. Do you suppose I could break up my favorite binary program into 2,048 word (2K) blocks, store them on disk as binary files, download them to 2K 2716 EPROM's in my EPROM burner, replace the D0-F0 Applesoft EPROM's (that I previously replaced the 9316's with), flip that magic red switch on the back of the Integer Card, and have, say the Copy][Plus utility programs up and running at the flip of a switch? I'm sure it can be done but I haven't tried it myself.

About EPROM Burners

EPROM burners can be expensive. I've seen them from thirty five dollars (the card fits into a slot in the Apple) to thousands of dollars. And EPROM erasers ain't cheap, either - they can run from about $ 50 to $ 80 and up. But, EPROM's are erased by UV (ultraviolet) rays like those given off by the sun and many fluorescent tubes. So, they say it's possible to tape an EPROM to a fluorescent tube or set it out in the sun for a week or so, and it should erase. Or, you may want to get together with a few friends and share the cost of an inexpensive burner, and, maybe an eraser.

The price of 2716 chips is down to a few dollars each. You don't need the "fast" 250 ms jobs. The slower, less expensive ones at 450 ms or so will work just fine. A good source of all this hardware is advertisements in the back of Byte, Computers & Electronics or COMPUTIST magazines.

Have fun burning.

References:

1) Computers & Electronics, November 1983, Pages 58-60, *Customize Your Apple With an EPROM Plug* by S. F. Mitchell

2) COMPUTIST No. 6, Pages 14-16, *Modified ROMs* by Ernie Young (see COMPUTIST No. 8 page 26)

3) COMPUTIST No. 19, Pages 9-11, *Double Your ROM Space* by Ray Darrah

4) COMPUTIST No. 9, Page 18, *Using ProDOS On A Franklin ACE*

5) *Pro-Byter* by Beagle Bros, page 78

6) COMPUTIST No. 8, Page 3, *Reading Apple ROM's*

7) The Best of Hardcore Computing, Page 46 *Curing Those Auto-Start Blues* by Charles R. Haight

8) COMPUTIST No. 9, Page 5, *Mod2 ROMs*

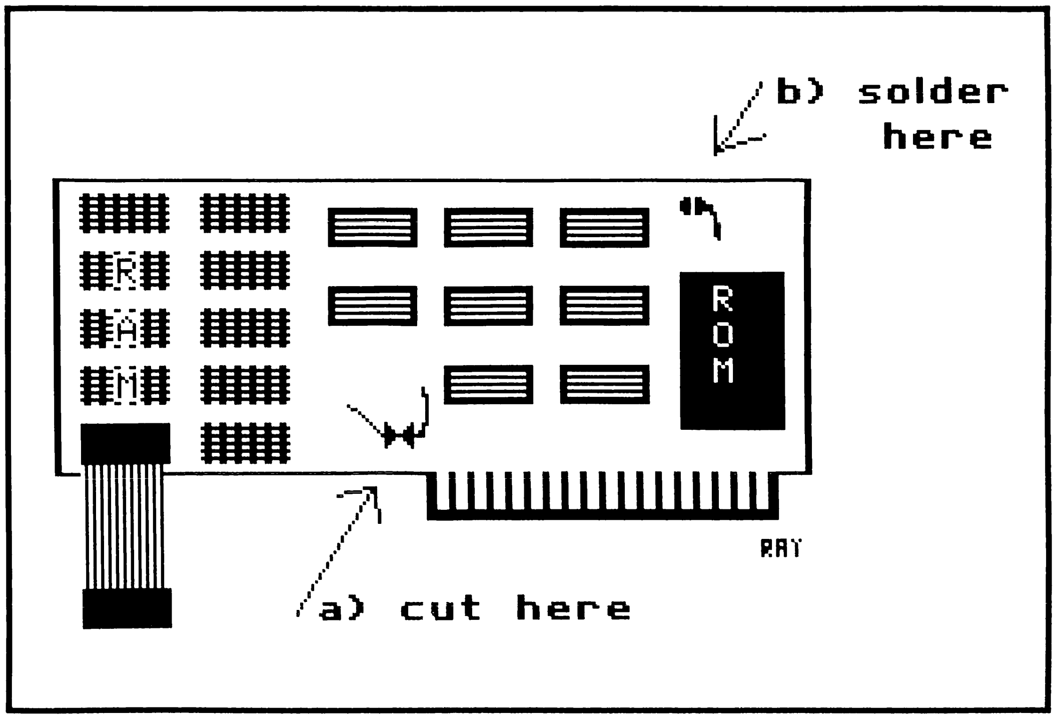
9) *COMPUTIST No. 12, Pages 24-26, Pseudo-ROMs On The Franklin ACE by Ken Stutzman (see COMPUTIST No. 14, page 9)*

A) *COMPUTIST No. 19, Pages 18-24, Towards A Better F8 ROM by Earl Taylor*

B) *COMPUTIST No. 22, Pages 4-5, Modified PROMs On The //e*

C) *COMPUTIST No. 24, Page 5, Double Those ROMs*

D) *Apple][Reference Manual, APPLE Product #A2L0001A, Appendix C*



A Review of...

The Senior PROM

by Robert Knowles

Cutting Edge Enterprises
Box 43234 Ren Cen Station
Detroit, MI 48243
$79.95

So you want to take control of your Apple //e or //c? You've decided you need: a way to reset directly into the Monitor, an NMI & copy card, a sector editor, and some machine language utilites, but you only have 80 bucks to spend? A new, low-cost device consisting of 32K of ROM and a pair of switches attached by a ribbon cable, going by the name of the *Senior PROM*, may be just what you've been looking for.

Description

The Senior PROM (SP) is a printed circuit (PC) card about two inches square, that replaces the 8K (2764) CD and EF ROMs on the motherboard with a pair of 16K (27128) EPROMs. These contain an image of the original Apple ROMs in one half and the SP firmware in the other half. At the end of the two foot long ribbon cable is a second PC card (2" by 1") with a pushbutton switch for performing Non-Maskable Interrupts (NMI's) and a three-position toggle switch to select and deselect SP. A small micro-probe with just enough wire to reach from the SP card to pin 6 of the 6502 provides the hookup for the NMI. A companion disk (described below) is included.

Installation

Installation is slightly more complex than just plugging in a card. You must remove the CD and EF ROMs, observing the usual precautions, insert the Senior PROM into the empty sockets, and attach the NMI wire to the processor. Although I have not tried the SP on a //c, I'm sure that installation in the //c will be a bit trickier, as the case "snaps" together.

Documentation

The documentation included with the SP is comprised of three stacks of stapled-together pages, all of which have clear, easy to read print. The thickest stack (54 pages) contains installation instructions for both the //e and //c and gives a thorough description of all the SP's features. In many cases, the instructions explain why a feature is provided and possible uses for it are given.

The smallest stack contains a keyboard overlay (on ordinary paper in two black and white halves that must be cut out and taped together) for quick reference to the options that become available directly after pressing Reset. This stack also contains diagrams for opening your computer and installing the SP.

The third stack is entitled "Deprotection Methods and Techniques Using the Senior PROM." This one gives an introduction to copy protection and indicates things to look for when identifying a protection scheme, with some ways to get around them. This is a very general manual (not much detail does 33 pages make), but it should provide a good starting point for the novice, assuming he knows assembly language.

The Switches

The SP firmware is inactive when the toggle switch is toward the rear of the computer. In this mode your Apple works like normal, ROMS and all. When you switch it to the forward (activated) position, Applesoft BASIC disappears (this could cause bad things to happen if you happen to be running an Applesoft program) and the only means of talking to your computer is through the SP's enhanced monitor. Once here, you have several utilities at your disposal including a sector editor, RWTS menu (with a disk copier) and Memory Management.

However, the real power is available when you press Reset or the NMI button. If the SP is in the "activated" or "transparent" (middle) position, the computer stops everything, responds with a distinctive beep, and sits in the "Wait Mode." At this point there are a number of options available. You can drop directly into the Monitor, reboot without destroying memory, or go directly to the built-in RWTS utilities. You can copy all of the main RAM to the auxiliary RAM (assuming of course you have at least 128K RAM) for saving; swap main and auxiliary; see where the program was when you interrupted it (if you used NMI); restart a program from the last time you interrupted and saved it; dump the text screen to the printer; and a few other things. All these options appear on the keyboard overlay so you can quickly choose the one you want. If however, you refuse to tape the keyboard overlay to your computer, you will be forced to memorize several arbitrarily assigned keys.

The Menus

The RWTS Utilities, also called the Main Menu, appears on the screen after pressing the "0" key from the wait mode. From here you can format a blank disk without DOS, copy a disk, alter the prologs and/or ignore the epilogs when reading a disk, or ignore most other RWTS errors (via the handy-dandy **B942:18**). The sector editor and disk copier can use either the RWTS kept in ROM or use an RWTS captured from a program you wish to examine or deprotect. From the Main Menu, one can also enter the sector editor or go to the Memory Management Menu.

The built-in sector editor is not the greatest ever, but it has many features. The standard features like switching between ASCII and hex display, reading incrementing sectors, editing bytes, and disassembling the sector are there as well as multiple sector buffering, disk and memory searching, reading sequential sectors into sequential memory, and nibble viewing of a track. In addition to all that, viewing the catalog, free sector map, disk comparison and reading half tracks are also supported.

The Memory Management Menu is a repeat of some of the memory move features available through the Wait Mode with the addition of copying sections of auxiliary RAM to useable main RAM for examination. Another option writes a test pattern into main memory for tracking what memory a program takes up. This menu could have been easily incorporated into the Wait Mode, but wasn't for some reason.

A New Monitor

The modified monitor will be a welcome addition for those who have 65C02s and want to use the extra opcodes available. The disassembler and the mini-assembler are both fully 65C02 compatible (a function not available even in the enhanced //e ROMs). The Step and Trace commands have been resurrected from the grave Apple put them into and may come in handy.

The hexdump and disassembler have an ASCII display along the right side of the screen, useful for searching for text strings. This part has a harmless bug in which the ASCII is only printed at the bottom of the screen. It scrolls up normally, but if you start disassembly at the top of the screen the ASCII does not get printed on the same line until the display does start scrolling. I hope this will be corrected in later versions.

The Fun Part

One of the features that the SP's advertisement is trying to push is the ability "to halt a program and instantly swap between two different 64K programs." It usually can. The catch is, you have to be able to break the program with an NMI so that it knows where to restart the program, or else know the entry point. I wanted to try this with my word processor and Dino Eggs (a classic!), for example. Dino Eggs works just fine and I could restart it any time I wanted. I could do most of the tricks like freezing the game before a level and retrying that level until I finished it, and even playing two games at once. However, things got tricky when programs took up 64K, or just played with the "language card". I was unable to restart Flight Simulator II at all and Magic Window //e was extremely balky, but it can be done with most 48K programs. Some programs are simply hopeless, but this is due more to the //e's hardware than to a fault in the Senior PROM.

The Disk

The disk contains utilities to save the contents of memory on disk in either 48K, 64K or 128K versions, depending on the program requirements. This allows you to restart the program cold from a DOS 3.3 environment. Also included (free unless you use them) are Diversi-DOS and Diversi-Copy from DSR. If you do decide to use them, DSR requests that you pay for them. You can also find COPYB, a COPYA modification that reads and copies protected disks with their own RWTS (like Super IOB's swap controller).

Gripes

Four out of five hackers surveyed prefer a cursor-based sector editor so they know exactly which byte they are dealing with. In the SP's sector editor, you must specify the byte number from a prompt at the top of the screen, then type in the bytes with only an address to tell you where you are. I prefer a visual indication of what I'm doing in cases like this.

The sector editor's search function is not like I expected. First you have to specify the string you want to find, then it **always** (whether you wanted it to or not) searches memory for all occurances of the string or sequence. Oh, you wanted to find it on *disk*? After wading through the memory search, press "L" for Locate. Now, tell the sector editor to search through the sectors and it will beep for each one it finds and stop. The catch is, it doesn't tell you *where* in the sector it found the string. This forces you to visually scan 256 not-so-readable, jam packed together bytes for the search string.

It would have been nice to have a screen-to-printer function available for all the features in the sector editor, like the disassembly or the memory searcher. But Noooo! If you want a screen dump, you have to hit the NMI button and print the screen from the wait mode. You CAN dump the sector display by pressing $\boxed{\text{☐S}}$, which (oddly) is $\boxed{\text{☐P}}$ in the wait mode. Pressing $\boxed{\text{☐P}}$ in the sector editor changes the disk drive slot (P for Peripheral? Give me a break).

The documentation mentions a ProDOS mode for the sector editor while talking about changing buffer pages, but no mention is made of how to activate it, if it even exists.

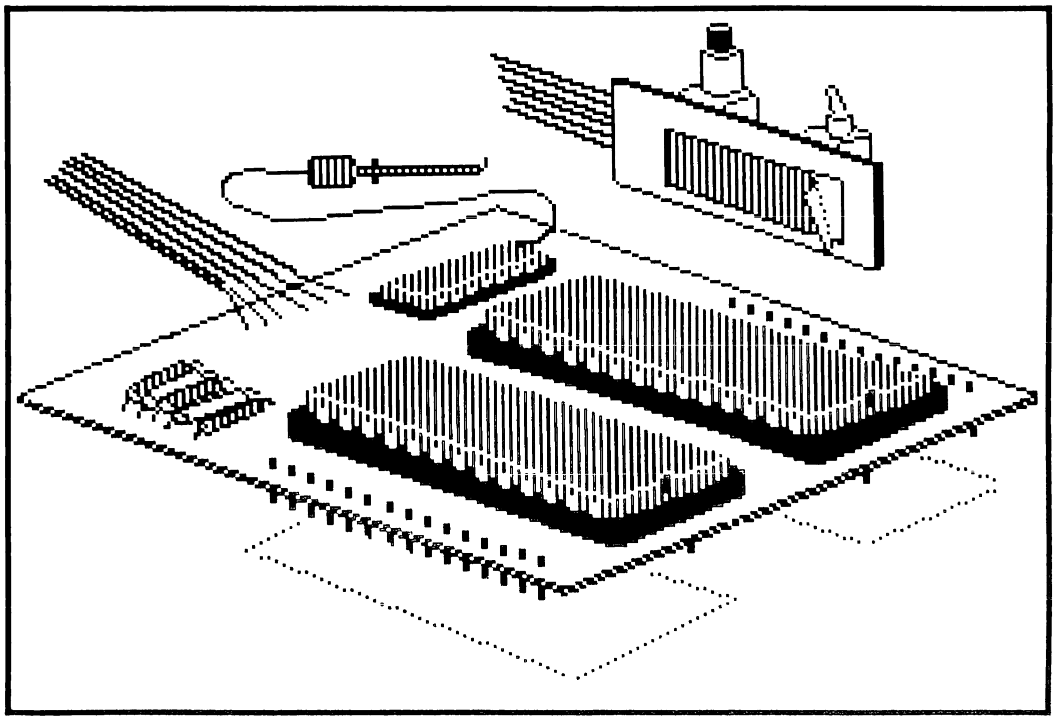
Methods of movement between features are inconsistent. To get into the Monitor, you press ESCape from the sector editor or menus and Delete from the wait mode. In the wait mode, ESCape selects the mixed or full screen graphics. To get to the sector editor, you press "8" from the wait mode, "3" from the RWTS menu. Things like that made learning to use the Senior Prom a bit tougher.

The Senior Prom circuit board is taller than a normal chip in a socket. Its ROMs plug into sockets on the board which in turn plugs into the empty ROM sockets on the motherboard. This height difference will cause problems if you have long peripheral cards plugged into slots 6 or 7 of the //e. The Disk][controller card will be OK, but a card more than a couple of inches longer than that will run into trouble. I was concerned that the micro-probe attached to the 65Ø2 would be too high for slot 4 or 5, but repositioning the probe leaves plenty of room for cards.

A Bargain?

Since this is still a fairly new product, it needs a little cleaning up, especially in the sector editor. However, having a sector editor and copy program ready to use at any time (DOS or no DOS) can help you overlook many of its limitations. It will take a couple hours of exploration and referring to the manual to get comfortable with it, and it will not make much sense if you haven't used machine language before. In fact, the "Deprotection Methods" manual emphasizes that you should read and study the "Apple // Reference Manual (//e)" and "Beneath Apple DOS", **and** learn assembly language and the Monitor.

If you add up the prices you would pay for each of the features separately, they could add up to over 15Ø dollars easily. On top of that, all the tools are in one neat little package that no program can detect. I intend to use it in my //e here at COMPUTIST for quite some time. It may not be pretty, but it can be a valuable and educational tool for the serious user of Apple //e's and //c's.



Masquerade and Other

by Steve and Rod Smith

Phoenix Software
6640 N. Sioux
Chicago, IL 60646

Requirements:
2 blank disk sides
The Inspector or similar sector editor
Super IOB

Masquerade is a fine graphic adventure with excellent graphics throughout. Unfortunately, it also comes with a fine protection scheme as well. Attempts at copying the disk will produce only marginally booting copies at best. This article deals with the protection scheme used and lays the foundation for deprotecting other programs using the same protection.

The Protection

Let's begin with a description of the protection scheme itself. To the best of our knowledge, it was created by Dav Holle, the man from Phoenix Software and Zoom Graphix fame. He is a acquainted with of Mark Pelczarski of Penguin Software, which may explain the reason why Penguin's protection is so similar. Basically it has to do with disk formatting. The address markers for the tracks alternate every other track, even tracks being D5 AA 96, odd tracks D4 AA 96. The data markers are left alone. The epilogue bytes are usually changed to DA AA or EB AA, although any legal value may be used for the first epilogue. This is one spot where the protections differ from program to program. The two major differences between Penguin's use of the scheme and others is the loader used and the disk volume number (which is tied in with the loader). Penguin uses a normal DOS with a common disk volume number, where as the others use Dav Holle's loader and all have a disk volume number of zero. This disk volume number is very important, but more on that later. The methods for deprotecting the Penguin version have been fairly well described, but the original version of this protection scheme needs a little more discussion.

An Overview of the Approach

The first objective in deprotection is to convert the program into a format that can be read with any normal copy program, such as COPYA from the Apple DOS 3.3 System Master disk. Then, the second step is to modify the program to operate under these conditions as well as to remove any further protection schemes from the program, such as nibble counts, et cetera. The one advantage in having to undo this protection is that it does not incorporate any further protection past the abnormal disk format. So this is where we begin.

Two different programs provide a way of converting to a normal format, the first of which is Super IOB. The controller at the end of this article has a few things worth noting. The last track (variable LT) specified in line 1010 does not neccessarily need to be 35. Some disks using this protection only use part of the disk, Masquerade being one of them. Its boot side only uses tracks 0 through 17. So, if you use this controller for a different program, set LT to one more than the last track actually used. The other note is in line 1110. The argument of the two pokes (POKE 47505,x : POKE 47413,x) is the epilogue byte mentioned before. For this particular controller, a value of 235 is used, but it could be different for another program. A way to find the correct value will be shown later.

Just install this controller in Super IOB and run it on the disk. Super IOB also provides an easy way to correct for a disk volume number of zero. When prompted, format your data disk with just such a number. For Masquerade, our job is a bit easier, as the actual playing side is in a normal format to begin with. So just convert side one. Other programs may use different configurations.

Making It Work

OK, that was the easy part. Now for the really fun stuff. We have changed the program's operating environment, so we need to modify the program to run in its new one. This requires the modification of four sectors of code on track zero. In order to do this efficiently, one must be able to read the specific sectors into the correct memory pages used. This is so we can see the code in its actual running environment. At the very minimum, you must at least be able to write out memory pages onto specific sectors, but if you can do this, you can read them in as well. The purpose of this is to avoid a boot trace, where the end result is the same. The program best suited for this purpose is the Inspector, and it is assumed here that this is what is being used.

The code under consideration resides all on track zero, the boot track. The best way to modify it is to just read the sectors used into their eventual memory locations. The following table shows what these should be:

```
(Track $00)
  Sector    should be read into
  ------    -------------------
    00              $800
    0E              $900
    0D              $A00
    0C              $B00
```

Once the code is in place, we want to enter the monitor and modify it. A few things should be noted also.

Description of the Loader

$801-828: This loads in the rest of the boot code.
$887-92D: Sets up some zero page locations, clears the hi-res graphics screen, moves ROM into a RAM card (if present), and modifies the RESET vector.
$930-95F: This decodes the reset program and stores it down on the stack page. Then it decodes the rest of the boot code at $962-BFF, and stores it down on the text page. Jumps to continue the boot at $95F (JMP $792).

Well, all this code seems harmless. We are intersted in where it reads in from the disk again, so we can change it to read a normal format. This is the code that is encrypted starting at $962. So we must decode it to look at it. To do so, while the boot data is in memory, make the following changes:

Dav Holle Protections

952:6Ø

(to stop the decoding routine)

945:Ø9

(to store the code in the same spot)

Then we run the decode part of the boot by typing:

91BG

Now the code should be in its final format. Before we forget, we must return the part of the program we just ran back to its original form, so type:

952:A5
942:Ø9
945:Ø9

Once this is done, we can start to examine the rest of the boot code. Before, we saw a jump instruction to $792, so we begin our trace at $B92, since we did not allow the program to relocate the code. At this point in the discussion it should be mentioned that tracing this type of code is very difficult and tedious, since the program manipulates the stack's return addresses for much of its operation. You will note that the code does not flow smoothly from $B92 on up. This is because the returns from the JSRs do not come immediately back, but are moved ahead slightly. If you look at $BF4, you will see a JMP $FC58, which is the monitor's HOME routine. There is an RTS at the end of the routine, but just where will it return to? Well, by the time the code is this far, there are two extra values left over on the stack that have been forced on previously. The values point to the actual beginning of the program. If you look at $A3F and $A40, you will find an FF Ø7. This is one less than the actual program start, so the program really starts at $8ØØ. All the loaders are exactly like this one, except the code is perhaps longer or shorter by a few bytes. They are relatively the same though. So if you are examining a similar loader from a different program, bear in mind that the addresses used here are for the Masquerade loader only. The address you need should be very close, though. Just look for the same code in some relative position and see by how much they are shifted. Then just keep that difference in mind. OK, so in and amongst all this jumping about, there must be some disk routines. Well, yes, it turns out that they start at the very beginning. If you list $962 and up, you will notice the familiar drive addressings and such. So what needs changing?

The RWTS Modifications

$9A9: Change from EB to DE (with 9A9:DE). This is the epilogue byte that was modified.
$A1Ø: Change from EB to DE (same as above).
$B6E: Change from 29 to Ø9. This is part of a routine that takes the current track number and checks it with the current address header. Since we will no longer have a D4 as part of the address header, we must modify the routine to produce acceptable values. Basically, we changed an AND to an ORA, so we will not get zero values.

Now that we have the rest of the boot decoded and the RWTS fixed, there is but one more change to make. When the code was being decoded by the program, it EORed with the current value of the accumulator before storing. We do not want this, as it will destroy our own decoding job. So at $94Ø we change the instruction to LDA by making it a BD (94Ø:BD).

To finish, just write back the modified sectors to disk according to the sector/buffer table given previously. If you used the Super IOB to convert the disk, you should have already formatted the data disk with a volume number of zero. If you forgot to do so, you will want to use the INIT program from Bag of Tricks to re-format the disk, preserving data of course.

This completes the deprotection of Masquerade and any other program using Dav Holle's protection methods. Other programs that we are aware of using this scheme are:

Sherwood Forest
How About a Nice Game of Chess?
Chess 7.Ø
and **Queen of Phobos**

We have succesfully deprotected all but Chess 7.Ø (which we do not have) using this method with only minor changes. The changes involve only the diffences in epilogue bytes and code addresses mentioned earlier. To determine what the epilogue bytes are for your program, you may either boot trace the disk until the RWTS is decoded, or you may read in the sectors off track zero, following the procedure and table above. You may have to turn off the checksum at $B942 (B942:18) in order to read these sectors. Or yet, maybe one of the nibble analyzers with a bit copy program can give you an indication of what they are. Just remember to change the Super IOB controller accordingly. We have found that the deprotected versions boot faster and more rėliably than the originals do. So, besides having a back up of your precious original, you have another reason to take off the nasty protection scheme!

Good luck, and happy cracking!

controller

```
1ØØØ REM  DAV HOLLE CONTROLLER
1Ø1Ø TK = Ø :ST = Ø :LT = 35 :CD = WR
1Ø2Ø T1 = TK : GOSUB 49Ø : GOSUB 111Ø
1Ø3Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST <
     DOS THEN 1Ø3Ø
1Ø4Ø IF BF THEN 1Ø6Ø
1Ø5Ø ST = Ø :TK = TK + 1 : GOSUB 111Ø : IF TK < LT
     THEN 1Ø3Ø
1Ø6Ø GOSUB 23Ø : GOSUB 49Ø :TK = T1 :ST = Ø
1Ø7Ø GOSUB 43Ø : GOSUB 1ØØ :ST = ST + 1 : IF ST <
     DOS THEN 1Ø7Ø
1Ø8Ø ST = Ø :TK = TK + 1 : IF BF = Ø AND TK < LT THEN
     1Ø7Ø
1Ø9Ø IF TK < LT THEN 1Ø2Ø
11ØØ HOME : PRINT : PRINT "DONE^ WITH^ COPY" :
     END
111Ø POKE 475Ø5 ,235 : POKE 47413 ,235 : IF TK /
     2 < > INT (TK / 2 ) THEN POKE 47445 ,212
1115 IF TK / 2 = INT (TK / 2 ) THEN POKE 47445 ,213
112Ø RETURN
```

controller checksums

```
1ØØØ  - $356B     1Ø7Ø  - $A827
1Ø1Ø  - $3266     1Ø8Ø  - $8EE6
1Ø2Ø  - $A7C4     1Ø9Ø  - $ØD8Ø
1Ø3Ø  - $B1C5     11ØØ  - $7496
1Ø4Ø  - $35FA     111Ø  - $CØØØ
1Ø5Ø  - $7575     1115  - $FCDA
1Ø6Ø  - $AØ22     112Ø  - $6914
```

Sword of Kadash

by John Della Pia

Penguin Software
830 Fourth Ave.
P.O. Box 311
Geneva, IL 60134

Requirements:
A deprotected backup of Sword of Kadash (see COMPUTIST No. 27)
A sector editor

Sword of Kadash came in such a colorful package I just couldn't wait to boot it up and start playing. But after a few sessions I renamed it Bored of Kadash and stuffed it back on the shelf. The original protected copy of Sword of Kadash, at least the Apple version, is very hard to enjoy playing. Each time your last player is killed the entire playing disk must be recopied using the slow copy program that comes with the game. This becomes tedious very quickly. After receiving COMPUTIST No. 27 with Michael R. Ditz's softkey for Sword of Kadash and using it to deprotect my disk, I decided this game might be worth playing if I could give my character a fighting chance. So, armed with Copy II Plus' sector editor and the Crucial Code Finder by Enrique Gamez (a memory search utility from COMPUTIST No. 6), I spent the next few evenings digging through code. With the resulting APT you will be able to customize Sword of Kadash as you wish. You can save the game position before trying anything dangerous, add as many lives as you like, or build a much stronger character to aid you in your quest. For those who are in a hurry to start playing I have added instructions at the end. Just follow the steps there and refer to the character attribute table to build your super Kadash character. For those who want to follow along I'll explain the reasoning behind the APT.

The back of the Kadash disk contains all of the room data as well as the character attributes. The data table for the character attributes is located on track $22, Sector $F, bytes $00-$1A. Table two is a listing of these bytes and what each one equates to in the game. When you use a new player disk for the first time these attributes are read into the game from the player disk. If you save the game by pressing ⌘Q the character attributes you have developed during play are saved back to the player disk to the same track and sector where the original default values were stored. The condition of the last screen being played is also saved at this time to the track and sector on the disk originally occupied by that screen. When restoring an old game, the program reads in the saved attributes and then goes to the last-used screen to resume play. Restoring with one player disk and then replacing it with a fresh disk will give you a fresh set of screens complete with treasures to find, traps to avoid and monsters to slay while leaving you with the character attributes developed on the first disk. That's not much of a bonus unless your character is strong enough to handle the job.

The easist change you can make to Sword of Kadash is to the number of lives you get before the dreaded skull and crossbones appears on the screen telling you it's time to copy yet another disk. By changing byte $1A in the data table from $02 to any other number you can add that number of lives to your character. However, changing any of the other bytes leads to trouble. When you start to play and place your newly modified player disk in after being prompted to do so all you will get is an "Illegal Character Disk" message. This is because Kadash loads in the data table and then checks to see if these bytes have been changed before it reads in the screen data and starts the game. If you have changed any bytes the program won't run.

The subroutine that does this check is found in the program "HIMEM" and loads into the computer memory at $5E00. Table one is a listing of this code along with some remarks on what it is doing. As you can see from the listing, the table on track $22, sector $F of the data disk (Side B of the original Kadash disk) is first loaded into memory at $4400. Then the byte at $4416 is tested. If this byte is zero the program jumps to $4FC0 and the "Illegal Character Disk" message appears. If the byte is not zero the code branches around the jump and loads both the A and X registers with $00. Following this, the memory locations from $4400 to $4415 are Exclusive-ORed against the A register. The result of this is compared with the number at memory location $4419. If the sums match the program goes on. If they don't match the program jumps to $4FC0 and we get the "Illegal Character Disk" message. To remove the checksum and allow any attributes you like to be given to your Kadash character all that is necessary is the removal of the two JuMPs to memory location $4FC0. You can do this by searching and editing the disk or by BLOADing the program HIMEM, modifying it and BSAVEing back to the disk. Once this is done you are free to modify your character as you wish.

A side note: I had a great deal of trouble using the softkey for Kadash on my copy. After repeated failures I retraced Mr. Ditz's softkey and broke the controller back down into two

Table 1

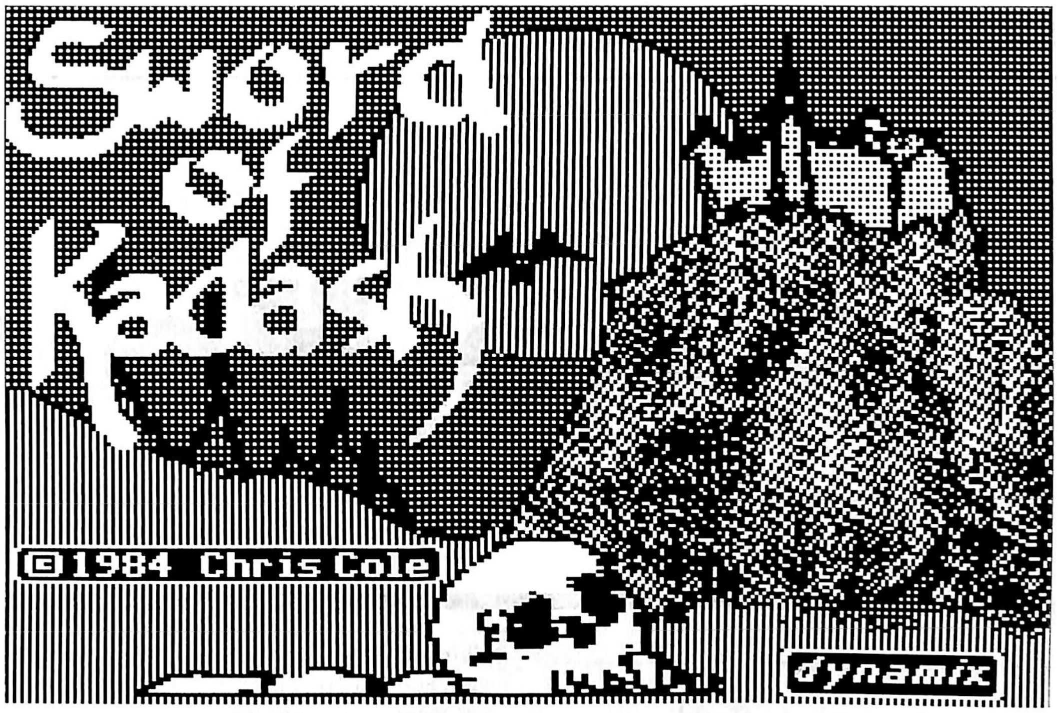
Code in program HIMEM that reads character attributes and does a checksum on them.

```
5E00-  LDX #$22    Track Number to read.
5E02-  STX $030E
5E05-  LDX #$0F    Sector Number to read.
5E07-  STX $030F
5E0A-  LDX #$44    Load Address = $4401?
5E0C-  STX $0313
5E0F-  LDX #$01
5E11-  STX $0316
5E14-  JSR $0300   Read the data disk
                   and come back
5E17-  LDA $4416   Get the byte now at
                   memory location $4416
5E1A-  BNE $5E1F   If byte is zero then
                   perform next command.
                   If not zero branch
5E1C-  JMP $4FC0   to "illegal disk"
                   subroutine.
5E1F-  LDA #$00
5E21-  LDX #$00
5E23-  EOR $4400,X Perform checksum on
                   locations $4400-$4415
5E26-  INX
5E27-  CPX #$16    Loop back $16 times
5E29-  BCC $5E23
5E2B-  CMP $4419   Compare the result
5E2E-  BEQ $5E33   with the byte at
                   memory location $4419.
                   If they match, skip
                   the JMP to "illegal"
                   routine and continue
                   program.
5E30-  JMP $4FC0   Else goto "illegal
                   disk" subroutine.
```

parts. I also removed the data statements and did the DOS editing using a sector editor. That did the job.

Here's How

To patch the deprotected Sword of Kadash to allow changes to the player disk:



Load in the binary file "HIMEM" from side A and replace the JMP $4FC0s (4C C0 4F) with NOPs (EA EA EA).

BLOAD HIMEM
CALL-151
5E1C:EA EA EA
5E30:EA EA EA
BSAVE HIMEM,A$4000,L$5400

To modify the character attributes:

Sector edit track $22, sector $F, bytes $00-$1A as shown in table two.

To change your level, spells, hitpoints, maximum hitpoints or experience use only decimal digits 0-99 (Don't use digits $A-$F). Maximum hitpoints, current hitpoints and experience are stored in reverse. That is, 00 20 is actually 2000 when you start playing.

Don't be too greedy. If, for example, you give yourself 99 spells and then grab one more when playing the game the counter will flip to 00 and you won't have any spells at all!

To save your position before attempting something dangerous:

1) Press ▭Q to save the game position.

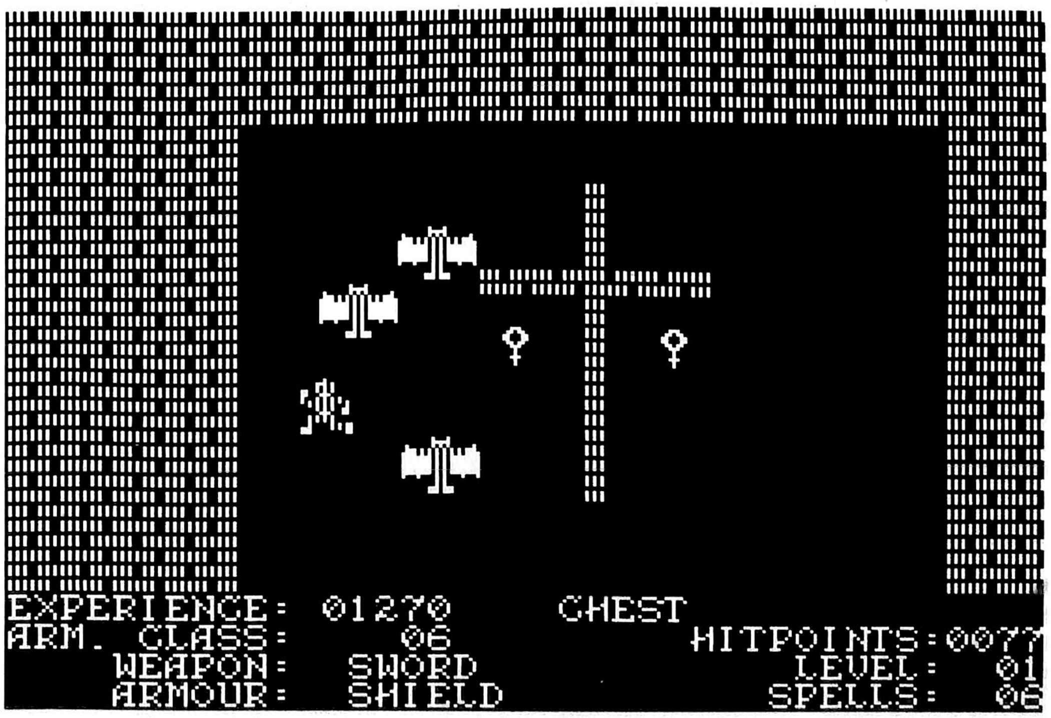
2) Press the spacebar and "B" to continue.

3) If you wish to return to the point in the game where you pressed ▭Q, remove the player disk and insert a blank formatted disk.

4) Press ▭Q.

5) Remove the blank disk and insert your player disk.

6) Press the spacebar and "B".



```
EXPERIENCE=  01270      CHEST
ARM. CLASS=     06              HITPOINTS:0077
    WEAPON=  SWORD                  LEVEL:  01
    ARMOUR=  SHIELD                SPELLS:  06
```

Table 2

Character Attributes for Sword of Kadash.

Track $22, Sector $F, Bytes $00-$1A

Byte	Used for
$00-$03	Screen no. and character position
$04	Weapon $00-$0A
$05	Armour $0C-$12
$06	Saves weapon value when cursed
$07	Saves armour value when Cursed
$08	Level 01-99 decimal
$09	Cursed = 01 Uncursed = 00
$0A-$0B	Maximum Hitpoints (low byte first)
$0C-$0D	Actual Hitpoints (low byte first)
$0E	Armour class
$0F	(Armour class backup?)
$10-$11	Experience (low byte first)
$12	
$13	
$14	
$15	Spells (Use 00-99 decimal)
$16	0 = illegal disk. 1 = good disk.
$17	
$18	
$19	Checksum of attributes
$1A	Number of lives (Use decimal)

The Hobbit

by J. J. Gifford

Addison-Wesley Publishing Co., Inc.
Jacobs Way
Reading, MA 01867

Requirements:
Copy program (COPYA is fine)
Sector Editor (such as DiskEdit)
or, Super IOB 1.5

Addison-Wesley has taken J.R.R. Tolkien's classic The Hobbit and used it as the basis for a mediocre graphics adventure game. Like nearly every other game produced today, it is copy protected. I tried to copy the disk using a few bit copiers and met with little success.

In order to study the boot code, I booted my copy (that would not boot completely). I noticed that it hung only after the hi-res title page. I used my Wildcard 2 to enter the monitor and poke around a bit.

The first page of code I examined was page $8. The code looked like a standard boot, except for one thing: it moved code from pages $B6 and $B7 to pages $2 and $3 respectively. Then it jumped to $200. The code for these steps starts at $839 and looks like this:

```
0839- A2 00       LDX  #$00
083B- BD 00 B6    LDA  $B600,X
083E- 9D 00 02    STA  $0200,X
0841- BD 00 B7    LDA  $B700,X
0844- 9D 00 03    STA  $0300,X
0847- E8          INX
0848- D0 F1       BNE  $083B
084A- 4C 00 02    JMP  $0200
```

As you can see, it is not hidden or encrypted at all. Although the memory move looks innocuous, there is something suspicious about moving code from the RWTS routine area to page $2 (the input buffer), which gets partially overwritten by a reset into the monitor. To follow the program flow I decided to examine pages $B6 and $B7 rather than $2 and $3. Since they were mirror images of each other (with the exception that page $2 was a little scrambled by my entry into the monitor), it shouldn't matter which set I examined.

Page $B6 had the unusual begining:

```
B600- 4C 04 02    JMP  $0204
B603- 2C A9 01    BIT  $01A9
B606- 8D EE 03    STA  $03EE
```

Since this code also resides at $200 in memory, the JMP at $B600 (and $200) only jumps over 1 byte, $B603 (or $203). If we list the code as the computer would see it upon execution (skipping byte $B603, that is) it would look like this:

```
B600- 4C 04 02    JMP  $0204
B604- A9 01       LDA  #$01
B606- 8D EE 03    STA  $03EE
```

It loads the accumulator with a #$01 and stores it at $03EE. This funny jump is one of the few bits of trickery Addison-Wesley has included in their protection scheme, however. Looking further down the sector, we see a segment of code that turns on the hi-res screen. This code starts at $B617 and runs through $B620. At $B623 there is a JSR (Jump to SubRoutine) to $348. The routine at $348 is also at $B748. Thus, we can see what the routine does by typing **B748G**. The disk spins, and the drive head reads several tracks in rapid succession. If you listen carefully to a boot of The Hobbit, you find that this is what the drive does immediately before the computer hangs, on a bad copy. Notice that the routine returns control to the monitor. This tells us that it returns to the instruction immediately following the JSR; it does not jump off to other code.

The instructions following the JSR start at $B626 and look like:

```
B623- 20 48 03    JSR  $0348
B626- A5 01       LDA  $01
B628- F0 03       BEQ  $B62D
B62A- 4C 2A 02    JMP  $022A
B62D- A9 03       LDA  #$03
```

Note the BEQ (Branch if EQual) instruction at $B628. This sends program control to $B62D if the value stored at $01 is zero (if a BEQ or BNE is executed without a CoMPare statement preceding, the computer assumes you meant to compare the accumulator with zero). This jump only skips over one instruction, the JMP $022A at $B62A. The JMP at $B62A must be important. If we type **B62DG**, the program continues booting, and the game runs normally. Now look at the JMP $022A carefully. Remember the memory move commands we saw on page $8? They transferred everything from $B600-$B6FF to $200-$2FF. That means that the JMP $022A is stored in memory at $22A. It is a continuous loop. This is what causes the computer to hang if the nibble count fails. The program simply branches to a loop from which it cannot escape. If we can get rid of the JMP $022A, and hence destroy the loop, we have deprotected the game.

It turns out to be quite simple to disconnect. First I searched the disk for the code surrounding the JMP. I found the JMP on track $0, sector $1, byte $2A. I replaced the three bytes that compose the JMP instruction with NOP ($EA, No Operation) instructions. Thus the program would continue to the code at $B62D ($22D) regardless of the result of the nibble count.

Step by Step

1) Copy The Hobbit using any standard disk copier. Even COPYA will work.

2) Using a sector edit such as DiskEdit, make the following sector edits:

Track	Sector	Byte	From	To
$00	$01	$2A	$4C	$EA
$00	$01	$2B	$2A	$EA
$00	$01	$2C	$02	$EA

That's it! Enjoy your unprotected version of The Hobbit.

```
4000: 20 58 FC 20 E2 F3 8D 52 C0 A9 FF
400B: 85 E4 A0 00 84 FC 84 FD 84 FB B9
4016: D0 40 84 01 49 A5 C9 1A D0 0F A9
4021: 00 85 FC 85 FD A5 FB 18 69 0E 85
402C: FB D0 62 0A AA BD 9C 40 85 FE E8
4037: BD 9C 40 85 FF A2 0F 06 FF 26 FE
4042: 90 3F 86 00 8A 0A AA 48 BD 67 41
404D: 48 4A 4A 4A 4A A4 FD 18 65 FC 90
4058: 01 C8 AA 68 29 0F 18 65 FB 20 11
4063: F4 68 AA E8 BD 67 41 48 29 0F 18
406E: 65 FB A8 68 A6 FD 4A 4A 4A 4A 18
4079: 65 FC 90 01 E8 20 3A F5 A6 00 CA
4084: 10 B8 A5 FC 18 69 0E 85 FC 90 02
408F: E6 FD A4 01 C8 C0 97 F0 03 4C 15
409A: 40 60 00 5F 00 7F 00 33 14 91 00
40A5: 3B 00 1B 20 73 00 5D 40 00 00 74
40B0: 0B 11 00 31 02 D5 04 D5 00 77 00
40BB: 1F 00 00 08 1F 00 6B 40 02 00 75
40C6: 00 08 0B 00 14 80 10 0D 00 00 A2
40D1: A1 AE AA A0 B1 AE B5 A2 AD A8 B6
40DC: BF BF B1 B7 AD A8 A3 B5 A0 AE AB
40E7: AA AA BD B5 A6 AD B7 AF B7 BF A5
40F2: B7 B5 A4 AD B4 A6 A7 A5 A3 A1 B5
40FD: AE AD A8 A1 B4 B7 BF B6 A1 A8 A6
4108: B7 B5 B6 AB B5 A6 A1 B7 B6 B4 AB
4113: BD BF A6 A5 B6 A5 BF BF B6 A2 A1
411E: B5 A6 AD B7 AF A4 B1 B7 B6 A1 B4
4129: B7 BF B6 B0 B7 A2 AD B4 B6 B5 AD
4134: B7 B5 A9 B1 A7 A2 B5 A9 AB B4 A1
413F: BF B1 B7 A1 A0 B1 AE B5 AD A8 B5
414A: B6 A2 AD B7 BF A5 B7 AA A1 A7 B6
4155: BF BF B3 B4 A1 B1 B7 A1 A5 A4 AE
4160: A1 B5 B6 AB AB B2 BF 00 05 00 A0
416B: A0 A5 05 A5 05 0A 0A AA A5 AA 00
4176: 53 05 53 53 A0 53 A5 05 AA 0A A5
4181: 55 A5 50 5A
```

BACKUP

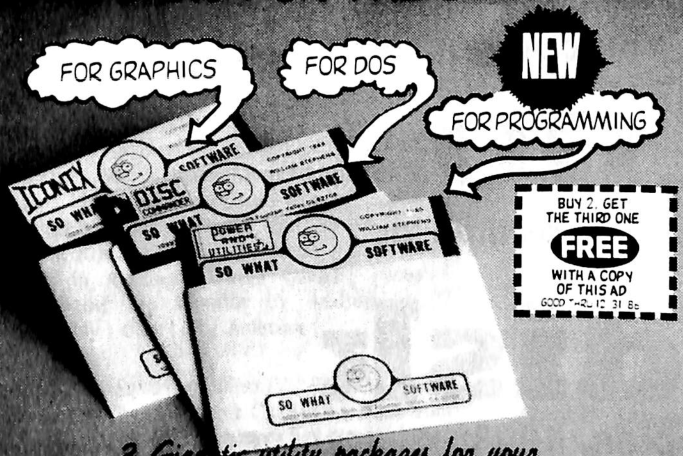
ESSENTIAL DATA DUPLICATOR

Back up your copy-protected disks with **Essential Data Duplicator 4 PLUS.** ■ **EDD 4 PLUS** is new technology, not just "another" copy program. The **EDD 4 PLUS** program uses a specially designed hardware card which works with your disk drives to back up disks by accurately copying the bits of data from each track. Don't be fooled . . . no other copy-program/system for Apples can do this! ■ In addition to backing up disks, **EDD 4 PLUS** includes several **useful** utilities such as examining disk drives, certifying disks, displaying drive speed rpm's, plus more! ■ **EDD 4 PLUS** runs on Apple II, II Plus (including most compatibles), and IIe, and is priced at $129.95 (duodisk/unidisk 5.25 owners must add $15 for a special cable adapter). Add $5.00 ($8.00 foreign) ship / handling when ordering direct. ■ A standard **EDD 4** version which doesn't include any hardware is available, and can be used on Apple IIc and III (using emulations mode) and is priced at $79.95. Add $3.00 ($6.00 foreign) ship/handling when ordering direct. ■ If you own an earlier version, send us your **EDD** disk and deduct $50 from your order. ■ Ask for **EDD 4 PLUS** at your local computer store, or order direct. Mastercard and Visa accepted. All orders must be prepaid. ■ In addition, registered owners may purchase **EDD's** printed 6502 SOURCE CODE listing for educational purposes.

WARNING: EDD is sold for making archival copies only.

UTILICO MICROWARE
3377 SOLANO AVE., SUITE 352
NAPA, CA 94558 / (707) 257-2420

Presents
BANG FOR THE BUCK



BUY 2, GET THE THIRD ONE FREE WITH A COPY OF THIS AD GOOD THRU 12 31 86

3 Gigantic utility packages for your Apple II series computers, self documenting and unprotected!

for dos ☐ Disc Commander __________ $29.95
for graphics ☐ Iconix __________ $29.95
for programming ☐ Power & Utilities __________ $29.95 NEW!
for free ☐ FREE catalog __________ $00.00

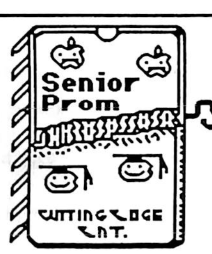
FREE ~~[illegible]~~ SHIP & HANDLE   FOREIGN-ADD FREE ~~[illegible]~~   CA FREE ~~[illegible]~~

SO WHAT SOFTWARE, 10221 SLATER AVE., SUITE 103,
FOUNTAIN VALLEY, CA 92708

Apple is a registered trademark of Apple Computer Inc.

Graduate...
to the Senior PROM!

Examine, modify, and backup your Apple //e and //c software!



The **Senior PROM** is a hardware device with deprotection utilities instantly available from **any** program. Including:

- Enter the Monitor to examine or change memory.
- Display where in memory the program was running.
- Disassemble, view or save any memory, even $00-7FF.
- Display the Stack for return subroutine addresses.
- Instantly switch between two different 64k programs.

After interrupting a program and examining or altering memory, the program may be instantly **restarted**. Or it may be **saved** to disk in normal B-files & later restarted. The **Senior PROM** also has a sophisticated Sector Editor and Memory / Disk Detective, and its own DOS with disk copy, format, edit, and protected disk utilities, all without booting a disk first! Assembly Language utilities include Step and Trace, an Assembler, and more. Undetectable by software or hardware, does not use a peripheral slot. Extensive documentation and guide to copy protection.

Economically priced at **$79.95** for prepaid orders with check or money order. Credit card orders available for **$88.95**. Specify //c, or //e standard or enhanced ROMs.

For VISA orders call 317-743-4041, 10-5 E.S.T.
or 313-349-2954 Modem 24 hrs. Not intended for illegal use.

Cutting Edge Enterprises
43234 Ren Cen Station, Detroit, MI 48243

Coming soon...

(to a magazine near you)

COMPUTIST No. 36:

Softkeys for: Flight Simulator II v1.Ø5, Autoduel, Critical Reading, Troll's Tale, Robot War, General Manager, Plasmania, Telarium Software, Kidwriter v1.Ø, 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.

COMPUTIST No. 37:

Softkeys for: Under Fire, Pegasus][, Take 1 (revisited), Magic Slate, Alter Ego, Rendezvous, Quicken, Story Tree, Assembly Language Tutor, Avalon Hill games, Dark Crystal.

Features: Playing Karateka on a //c, Track Finder, Sylk to Dif.

Core: Breaking In: tips for beginners, The DOS Alterer.

Review: Copy][Plus, 6.Ø.

5 FREE DISKS

With every set of 20 disks you order. You can get sets for as low as

$17.00

That's a total of 25 disks for as little as

68¢ a disk

These are SS/DD Namebrand, 5¼" floppies, 100% guaranteed, & include: reinforced hubs, write-protect tabs, & Tyvek sleeves.

Name ______________________ ID# ________

Address ______________________

City ____________ State ________ Zip ________

Country ____________ Phone ____________

VISA ________-________-________-________ Exp. ________

Signature ______________________ CP35

Most orders shipped UPS. Please use street address.

Send me ______ sets at $17.00 per set.
Add $3 shipping and handling for the first set, and $1 for each additional set.
Foreign orders add 20% shipping and handling.
U.S. funds drawn on U.S. banks only.
Washington orders add 7.8% sales tax.
Send your orders to:

SoftKey Publishing
PO Box 110846-T
Tacoma, WA 98411

Description of Available Back Issues

34 *Softkeys* | Crisis Mountain | Terripin Logo | Apple Logo II | Fishies 1.0 | SpellWorks | Gumball | *Readers' Softkeys* | Rescue at Rigel | Crazey Mazey | Conan | Perry Mason: The Case of the Mandarin Murder | Koronis Rift | *Feature* | More ROM Running | *Core* | Infocom Revealed | ....

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 | Mufplot | 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 | ....

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
35	☐	☐	☐
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	NA	☐	NA
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 ______________

________-________-________-________ Exp. ________

Signature ______________________________ CP35

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