

For The Serious User Of Apple][Computers

COMPUTIST

Issue No. 38

December 1986

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

Softkeys For:

Cyclod
Alternate Reality
Boulder Dash I & II
The Other Side
Gato V1.3
Wilderness
Golf's Best
Hard Hat Mack

Core:

Appavarex

Feature:

The Enhanced &
Unenhanced //e



(Page 8)

COMPUTIST
PO Box 110846-T
Tacoma, WA 98411

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

Coping With COMPUTIST

Welcome to COMPUTIST, a publication devoted to the serious user of Apple II and Apple II compatible computers. Our magazine contains information you are not likely to find in any of the other major journals dedicated to the Apple market.

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.

■ **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 by being in boldface and indented:

PR#6

The **[RETURN]** key must be pressed at the end of every such command unless otherwise specified.

Control characters are specially boxed:

6 [P]

Press **[6]**. Next, place one finger on **[CTRL]** and press **[P]**. Remember to enter this command line by pressing **[RETURN]**.

■ **Requirements:** COMPUTIST programs and softkeys require one of the Apple II series of computers and a disk drive with DOS 3.3. These and other special needs are listed at the beginning of the article under "Requirements".

■ **Software Recommendations:**

1) *Applesoft Program Editor* such as Global Program Line Editor (GPLE).

2) *Sector Editor* such as DiskEdit (from the Book of Softkeys vol I) or ZAP from Bag of Tricks.

3) *Disk Search Utility* such as The Inspector, The CIA or The CORE Disk Searcher (from the Book of Softkeys vol II).

4) *Assembler* such as the S-C Assembler from S-C software or Merlin/Big Mac.

5) *Bit Copy Program* such as Copy II Plus, Locksmith or The Essential Data Duplicator
6) *Text Editor* (that produces normal sequential text files) such as Applewriter II, Magic Window II or Screenwriter II.

COPYA, FID and MUFFIN from the DOS 3.3 System Master Disk are also useful.

■ **Super IOB:** This powerful deprotection utility (COMPUTIST 32) and its various controllers are used in many softkeys. This utility is now available on each Super IOB Collection disk.

■ **RESET Into The Monitor:** Softkeys occasionally require the user to stop the execution of a copy-protected program and directly enter the Apple's system monitor. Check the following list to see what hardware you will need to obtain this ability.

Apple II Plus - Apple IIe - 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 II 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 (COMPUTIST 6 or Book Of Softkeys III) or the "Dual ROM's" article (COMPUTIST 19).

Apple IIe - Apple IIc: Install a modified CD ROM on the computer's motherboard. Cutting Edge Ent. (Box 43234 Ren Cen Station-HC; Detroit, MI 48243) sells a hardware device that will give you this important ability but it will void an Apple IIc warranty.

■ **Recommended Literature:** The Apple II 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; *Assembly Language For The Applesoft Programmer*, Roy Meyers and C.W. Finley, Addison Wesley; and *What's Where In The Apple*, William Lubert, Micro Ink.

■ **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. If you type:

```
10HOME:REMCLEAR SCREEN
```

The LIST will look like:

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

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 commands. There are two types of spaces: those that have to be keyed and those that don't. Spaces that must be keyed in appear in COMPUTIST as delta characters (δ). All other spaces are there for easier reading. NOTE: If you want your checksums (See "Computing Checksums" section) to match up, you must only key in (δ) spaces after DATA statements.

■ **Keying In Hexdumps:** Machine language programs are printed in COMPUTIST as both source code and hexdumps. Hexdumps are the shortest and easiest format to type in. You must first enter the monitor:

```
CALL -151
```

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). A beep means you have typed something that the monitor didn't understand and must, therefore, retype that line.

When finished, return to BASIC with:

```
E003G
```

BSAVE the program with the correct filename, address and length parameters given in the article.

■ **Keying In Source Code** The source code is printed to help explain a program's operation. To key it in, you will need the S-C Assembler.

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 appears in COMPUTIST 17.

■ **Computing Checksums** Checksums are four-digit hexadecimal numbers which tell if you keyed a program exactly as it appears 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 appeared in COMPUTIST 1 and The Best of Hardcore Computing. An update to CHECKSOFT appeared in COMPUTIST 18. If the published checksums do not match those created by your computer, then you typed the program incorrectly. The line where the first checksum differs has an error.

■ **CHECKSOFT Instructions:**

```
LOAD filename  
BRUNCHECKSOFT
```

Get the checksums with: **& [RETURN]** and correct the program where the checksums differ.

■ **CHECKBIN Instructions:**

```
CALL -151  
BLOAD program 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 **[Y] [RETURN]**.

```
xxx.xxx [Y]
```

Correct the lines at which the checksums differ.

You have a LEGAL RIGHT to an unlocked backup copy

Our editorial policy is that we do NOT condone software piracy, but we do believe that 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 programs to meet his or her needs.

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

..."It is not an infringement for the owner of a copy of a computer program to make or authorize the making of another copy or adaptation of that computer program provided:

1) that such a new copy or adaptation is created as an essential step in the utilization of the computer program in conjunction with a machine and that it is used in no other manner, or

2) that such new copy or adaptation is for archival purposes only and that all archival copies are destroyed in the event that continued possession of the computer program should cease to be rightful.

Any exact copies prepared in accordance with the provisions of this section may be leased, sold, or otherwise transferred, along with the copy from which such copies were prepared, only as part of the lease, sale, or other transfer of all rights in the program. Adaptations so prepared may be transferred only with the authorization of the copyright owner."

United States Code title 17, §117 (17 USC 117)

HAPPY HOLIDAYS! HAPPY HOLIDAYS!

In light of the upcoming holiday season, COMPUTIST has decided to run a series of specials for our readers. The following pages contain a set of specials that were conceived with our special readership in mind. To take advantage of any of these offers, please refer to the special code in each box when ordering. Please hurry though, offers are only good while supplies last and expire on December 31, 1986.

X-86-01

FREE color-coded library case

with the purchase of 10 blank floppy disks.

DISK PRICES (includes shipping)
 U.S. \$10,
 Canada & Mexico \$11,
 Other Foreign \$15

Includes tyvek sleeves, hub rings,
 labels, and write-protect tabs

X-86-04

SAVE AS MUCH AS \$19.75

on back issues when purchased in quantities of 3.

Sale price U.S./Canada/Mexico \$10
 Sale price, other foreign \$20

Regular prices \$4.75 and \$13.25, respectively.

X-86-02

SAVE \$19.75 on library disks

& get a FREE color-coded case

with the purchase of 5 library disks.

That means, you get 5 library disks and a free color-coded case for only \$30.00.
 (Foreign orders add \$5.00 shipping & handling)

Regular prices are U.S./Canada/Mexico \$9.95 per disk
 and other foreign \$11.94 per disk.

X-86-05

SAVE \$5/\$8

Save \$5 on the price of Volume II of the Book of Softkeys, or save \$8 on the total price of both volume 1 & 2 when purchased together.

Sale Prices: Vol II \$12.95, BOTH \$22.90
 Regular Prices: Vol II \$17.95, Vol II \$30.90
 Shipping - US/Canada/Mexico \$2 per book
 Shipping - other foreign \$5 per book

X-86-03

SAVE \$3 on every back issue & disk combo.

With this special only 9.95 per set.
 (That's like getting a free back issue with every library disk.)

Normal retail price \$12.95 per set.
 Foreign shipping add \$3.00.

X-86-06

Get a FREE Core Special or FREE COMPUTIST T-shirt

with any total order of \$100.00 or more.

Yes I want to take advantage of your special Holiday offers. Enclosed is U.S. funds (drawn on U.S. bank) to cover my order.


Please send me _____ sets of 10 floppy disks. I understand that I will get a FREE color coded disk case with each set.
 Sale Price: U.S. \$10, Canada/Mexico \$11, Other Foreign \$15 per set.

I want to take advantage of the Book of Softkeys special prices.

Please send me volume II of the book of softkeys. Enclosed is 12.95 plus shipping & handling. (U.S., Canada & Mexico add \$2. Other Foreign add \$5).

Please send me Both volume I and volume II of the book of softkeys. Enclosed is \$22.90 plus shipping & handling. (U.S., Canada & Mexico add \$4. Other Foreign add \$10).

For remaining holiday sales, please consult 'Back Issues and Library Disks' order form on page 32.
 Send orders to: COMPUTIST PO Box 110846-T Tacoma, WA 98411

Name _____ ID# _____
 Address _____
 City _____ State _____ Zip _____
 Country _____ Phone _____
 _____ Exp. _____

Signature _____ CP38

U.S. funds drawn on U.S. bank. In Washington add 7.8% sales tax. Most orders shipped UPS so please use street address. Offer good while supplies last.

announcing new rates!

YES! COMPUTIST has **DROPPED** its annual subscription rate.

U.S. Domestic save \$8 per year
U.S. First Class save \$3 per year
Canada and Mexico save \$23 per year
All other foreign save \$45 per year

Additionally, COMPUTIST has incorporated a combination library disk and first class subscription rate to save you even more.



With this new 'COMBO' subscription, you will receive each monthly issue AND it's corresponding library disk for as much as 43% off the individual rate. Combination subscriptions are sent U.S. First Class mail.

If you have at least 3 issues left on your current subscription, you can upgrade to this special offer.

Yes I want to take advantage of the big money saving offer and subscribe to your fine publication. Enclosed are U.S. Funds (drawn on a U.S. bank) for a 12 issue subscription.

New Subscriber Please renew my subscription
 U.S. \$32 U.S. First Class/Canada/Mexico \$45 Other Foreign \$75

Combination magazine and corresponding disk subscriptions
 U.S./Canada/Mexico \$100 Other Foreign \$140
To upgrade your subscription to a combo subscription, U.S./Canada/Mexico send \$5.50 and other Foreign send \$6.50 per remaining issue. You must have at least 3 issues remaining to take advantage of this upgrade offer.

Name _____ ID# _____
Address _____
City _____ State _____ Zip _____
Country _____ Phone _____
  _____ Exp. _____
Signature _____ CP38
U.S. Funds drawn on U.S. bank. Subscription will not commence until funds are received. Send orders to: COMPUTIST PO Box 110846-T Tacoma, WA 98411

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

The Book Of Softkeys

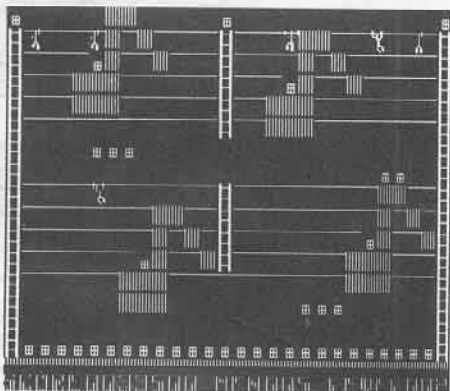
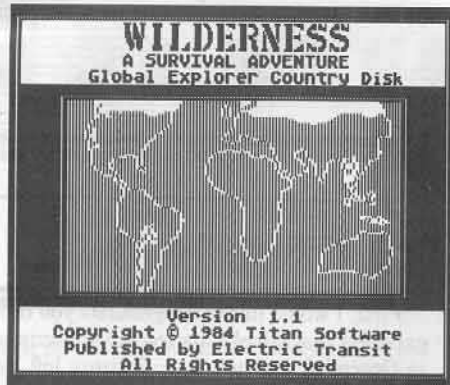
Volume I (\$12.95)

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

Volume II (\$17.95)

contains softkeys for: Apple Cider Spider | Apple Logo | Arcade Machine | The Artist | Bank Street Writer | Cannonball Blitz | Canyon Climber | Caverns of Freitag | Crush, Crumble & Chomp | Data Factory 5.0 | DB Master | The Dic*tion*ary | Essential Data Duplicator I & III | Gold Rush | Krell Logo | Legacy of Llylgamyn | Mask Of The Sun | Minit Man | Mouskattack | Music Construction Set | Oil's Well | Pandora's Box | Robotron | Sammy Lightfoot | Screenwriter II v2.2 | Sensible Speller 4.0, 4.0c, 4.1c | the Spy Strikes Back | Time Zone v1.1 | Visible Computer: 6502 | Visidex | Visiterm | Zaxxon | Hayden Software | Sierra Online Software | PLUS the complete listing of the ultimate cracking program...Super IOB 1.5 | and more!

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



This month's cover:
Graphics from Bantam Electronic's "I, Damiano"

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

Unsolicited manuscripts are assumed to be submitted for publication at our standard rates of payment. SoftKey publishing purchases all and exclusive rights. For more information on submitting manuscripts, see our writer's guide.

Entire contents copyright 1986 by SoftKey Publishing. All rights reserved. Copying done for other than personal or internal reference (without express written permission from the publisher) is prohibited.

The editorial staff assumes no liability or responsibility for the products advertised in the magazine. Any opinions expressed by the authors are not necessarily those of COMPUTIST magazine or SoftKey Publishing.

Apple usually refers to an Apple II computer and is a trademark of Apple Computers, Inc.

SUBSCRIPTIONS: Rates (for 12 issues): U.S. \$32, U.S. 1st Class, Canada & Mexico \$45, Foreign \$75. Direct inquiries to: COMPUTIST, Subscription Department, PO Box 110846-T, Tacoma, WA 98411. Please include address label with correspondence.

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

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

COMPUTIST

Issue 38

December 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

softkeys:

14 Cyclod

by Felix LeChat

19 Alternate Reality

by Stephen Lau

24 Boulder Dash I & II

by Randy R. Abel

25 Hard Hat Mack (Revisited)

by Brian Troha

26 The Other Side

by Dick Meikle and Jim McGreevy

feature:

10 The Enhanced/Unenhanced //e

Have you ever wished for a moment (perhaps when running some old software) that your Apple //e wasn't enhanced? With this modification to your computer, you can quickly switch between enhanced and unenhanced versions of the Apple //e. *by Wes Felty*

28 Looking into Flight Simulator's DOS

Following our softkey for Flight Simulator v1.05, COMPUTIST presents an in depth look at the DOS behind the program. *by Stephen L. Favor*

core:

16 Appavarex

This article details a how Applesoft program text is stored in memory and presents a program that documents in what lines of a program variables are used. *by Elwood J. C. Kureth*

20 Installing a RAM disk into DOS 3.3

If you have a //e with extended 80 column card, you can now have the super speed of a RAM disk with DOS 3.3. *by Robert Knowles*

departments:

4 Input

6 Most Wanted List

7 Readers' Softkey & Copy Exchange

MicroProse's F-15 Strike Eagle by John Howard, Broderbund's Championship Lode Runner by Sandy Eubanks, Spectrum Holobyte's Gato V1.3 by Robert Muir, Bantam's I, Damiano by Larry Rando, Intuit's Quicken by Greg Robinson, Electric Transit's Wilderness by Charles Taylor, One Step's Golf's Best by John Howard

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.

Opinions expressed are not necessarily those of COMPUTIST or SoftKey Publishing

Silent Service News

I only have one disk, Business or Recreational, that has resisted all efforts to back up. Including the use of COMPUTIST articles, EDD III, COPY II Plus, and a couple of older copy disks. This one is Silent Service by Microprose. A starting place is a deprotection method that appeared in COMPUTIST that partially works. However the deprotected version will run at a different level than you select. Even worse, a copy will always cause the submarine to hit an enemy mine and then sink. Having played many times on the original without ever hitting an enemy mine I can only conclude that the program counts cycles or keypresses on the copy. (Or perhaps my Apple just dislikes the copy.)

A quick way to tell is to play selection 2 under Convoy Actions, sink a couple of the ships and try to escape. If the copy is unsuccessful you will hit the mine. If you are playing the original disk the scenario will end by giving the readout of your accomplishments. (Assuming, of course that the destroyer does not send you to the bottom.)

Sheldon M. Atterbury
Sierra Madre, CA

John Einstein
Anchorage, AK

Arcade Boot Camp Softkey

Requirements:

Arcade Boot Camp
The Senior Prom
A blank disk

If you (for some strange reason or another) don't have the Senior Prom you will need to use a copy program that will allow modification of Address and Data Prolog bytes for the copying process.

Penguin software was very kind to provide Softdisk (a monthly magazine on disk) with the privilege to put a 1984 program (a fine quality commercial one at that!) called Arcade Boot Camp with the other programs and information they provide monthly.

I was almost overjoyed.

The joy I felt, however, was nothing compared to the joy I felt when I unleashed the power of The Senior Prom.

How I did it:

The first thing I did with Arcade Boot Camp was to find out how the data on the disk was perverted. I used the sector editor in the Senior Prom, called Snoopy and the Doctor (SAND). Pressing "N" does a nibble read and gives me a hex output of exactly what is on the disk. I found that the first Address Prolog byte was changed to \$D4 on every odd numbered track (ie: 1, 3, 5, 7, etc.). The second thing to do was to simply copy every other (odd) track with the first address Prolog byte changed to \$D4. Then copy all even tracks with normal Prolog bytes. This task is extremely simple with The Senior Prom's "Alter Prolog Bytes" function. It can also be done with COPYA, Super IOB, and so forth.

After I did this, realizing by the (now unusual) appearance of the Applesoft prompt. This means Applesoft is being used (usually, but now rarely) and as well DOS 3.3! As it turns out, the catalog sector is at track \$11 and the VTOC is normal. So all that this disk needs now is good old DOS 3.3 copied to tracks \$00-\$02. And it works!!

With the above information and as well, information from past COMPUTIST issues it is very easy to copy Arcade Boot Camp.

Hitchhiker Tip

First, I would like to congratulate you on an excellent magazine! I started my subscription in December and only have 2 issues left. My budget is at zero right now and I hope I find some way to save up some money and renew my subscription soon. Keep up the good work!

I also have one more thing to say, (unrelated, but useful). In Hitchhiker's Guide to the Galaxy, after you have gotten the Nutrimat-Computer Interface, don't hook it up until you hook up the Improbability drive to the console by plugging the large plug into the large receptacle. After that, go hook up the Interface to the Nutrimat, (you have to take out the old board first), activate it, run up to the bridge, and wait until Eddie announces that the ship is going to be destroyed then activate the drive. If anyone can help me after that point, please write to:

Chris Wood
1220 S. Sixth St
Rockford, IL 61109

On Ultimainland Editor...

I have a couple of suggestions for the Ultima IV Mainland Editor you printed in COMPUTIST No. 33.

- To make the mountains really stand out from the rest of the mainland, add a "POKE 947,42" to the end of line #110. This turns them inverse on the screen. When adding mountains using this option, the mountain will be a normal asterisk until you scroll it off & back on the screen again. It will then be inverse.

- It might be an idea to show the readers how to customize the program so that if they wanted to be able to add things to the surface that are currently not available, they could. (For example: boats, horses, stone pillars, etc.)

Secondly, for those folks who have MicroProse's **Nato Commander**, here is a short softkey for it (I haven't had the time recently to do a proper writeup).

- 1) Use the Locksmith fast copy program to copy the original onto a blank. Any copy program that can ignore bad tracks will work fine. By the way, this procedure will only work if Track \$06 is the "bad" track.

- 2) Get out your favorite sector editor and make the following changes to the copy:

input

TRACK	SECTOR	BYTE	FROM	TO
\$04	\$04	\$67-68	B0 06	18 EA
\$04	\$04	\$6B-6C	C5 2E	C9 05
\$04	\$04	\$70	F0	DB

Write protect the backup (for safety's sake) and hide the original.

Jim S. Hart
Jacksonville, NC


Mr. Hart: Thank you for your suggestions on Ultimainland Editor. The Bug you mentioned (edited from this letter) appeared in COMPUTIST No. 37.

Some APT's

I just received COMPUTIST for the first time and thought it was great, and I will stop subscribing to some other magazines to get yours. While I was reading Input, I saw an APT for Conan, by Kenny Khoo (COMPUTIST No. 32) and I tried it but it didn't work.

I do have a different one for Conan. When you get to the third screen, go down the ladder, but be careful you don't get eaten by the ants, and get the elevator or whatever it is to the top level. When you get there move over till you get near the tree and wait for the Avian Ally to come. It looks like a bird and if you jump and hit it, you will get an extra life. This only works once. Also on the fourth screen you get the key, a lot of swords, and an extra gem. This will be helpful for the next screen.

I also have one for Miner 2049er. Before you start playing the computer asks you how many players. Type in "#" and then the level number you want to be on. You can only do this with one player. In Hard Hat Mack do the same thing except don't type in "#", just the level number.

Here's one for Sea Dragon. Before you start the game just hit  and you will have 9999 air instead of the usual 6000.

Phillip Lai
Lincoln, RI

A Different Fantavision

Enclosed you will find a check for a subscription to the Best Apple magazine I have ever read. I normally buy your magazine at the newsstand but after reading about your circulation problem I decided to subscribe.

(Incidentally, one touches my COMPUTIST under penalty of severe bodily harm.)

Now down to the major reason for this letter. After purchasing a copy of Fantavision I immediately went to COMPUTIST No. 30 for Mike Saul's softkey. I followed all the directions for creating FVSTUFF, typed in the controller, started it up, and it bombed while trying to read the first sector. On examination of the disk with a nibble editor I found that the epilg marks on my copy were not the same as his. In addition, the epilg marks on the "authorized" copy (\$DE \$FF for address, \$DE \$AA for data) were different than the epilg marks on the original disk (\$FF \$FF for address and data). To copy the original disk I had to change line 1130 in the controller to read:

1130 DATA 255,255,255,255

To copy the "authorized" backup line 1130 had to be changed to:

1130 DATA 222,255,222,170

The rest of the softkey went like clockwork. Apparently, Broderbund installed minor differences in several of their Fantavision disks.

I would venture to say that if my copy is different from Mike Saul's, then there are probably more out there with different marks yet. I suggest that if you have trouble with the softkey, boot up a nibble editor and see what epilg marks you have got.

Thanks for the great magazine and keep the softkeys coming.

Jose A. Montano
Albuquerque, NM

More on SSI RDOS

Mr. McConnell's article about SSI's RDOS was both thorough and quite interesting. Since he invites for additional information regarding this topic, I'd like to contribute the following:

1) SSI has recently been using a fourth type of secondary protection. I've encountered this in two 1985 releases - Battle of Antietam and Battle Group. The protection involves what is essentially a nibble count of track 0.

In Battle of Antietam, it can be found in side 2, track 22, sector 2, beginning at byte 04. It goes under a filename of "J". In Battle Group, it can be found in side 1, track 22, sector 4, also at byte 04. It goes under a filename of "Arsenal". In other software, if the same routine is used, it can be found by searching for these bytes: CE 07 02 EF 07 02. Some of

your readers will recognize this as typical self-modifying code. They would be entirely correct; and that is why the routine is a little hard to find if you don't know what to look for.

It can be defeated by editing byte 04 of the sector from CE to 60 (an RTS). Meanwhile, the Qwerty file is still in the disk, but the protection routine is disabled. It's unnecessary to make the edits described in your article; perhaps the file had been left in the disk to throw off those who aren't aware that SSI had been using a new secondary protection.

2) I've also seen two recent releases, apparently produced by SSI jointly with a certain LDW Software, written in 16 sectors, and their protection is very light. These are Panzer Grenadier and NAM. The first byte of their address prologue alternates between D5 (on odd-numbered tracks) and D4 (on even tracks), very much like the Penguin format. The address epilg is also not normal. But all other marks are standard.

This can be cracked using COPYA. After running COPYA, reset first, go into monitor with a CALL - 151, and enter:

B954:4A 49 6A D0 EF
B993:00
B99B:00 F0

Then restart the COPYA by entering RUN 80, and you're off! The normalized copy should run without need of any further edits.

3) When the skew of the RDOS 3.3 disk is changed to "ascending 01", the skew of track 0 should not be changed from ("descending 02"). The boot will be faster.

4) In creating the RDOS 3.3, shouldn't the changes listed in your article include 7CCD:BF N 7CD1:BF?

Francisco Villaroman
Philippines

The Jenny Scandal

In COMPUTIST No. 29 Phil Pattengale suggested using the swap controller to deprotect Jenny of the Prairie. Unfortunately, more was needed on my disk.

After using the swap controller, my disk would crash right after the title screen. Using my Wildcard to enter the monitor at the point of the crash, I found the following interesting code.

4AF9:4C F9 4A JMP \$4AF9

Now I'm no machine language wizard, but even I can see an infinite loop that's this obvious.

input

I then just scanned the disk for those three bytes and found them at track \$1A, sector 01, byte C8. I then changed the F9 to FC to bypass the infinite loop and I had a perfectly working copy.

Gregory L. Moor, M.D.
FPO New York, NY

Spy's Demise APT's

Location \$60AB holds the number of men the game is initialized with. You can change this to any value less than \$7F. Either load, change, and save the appropriate file or search the disk for the bytes \$A9 \$05 \$85 \$27 \$85 \$17. Change the second byte in this sequence to the number of men you would like.

Location \$17 contains the number of men left during actual play. Do a NMI and change this location if you suddenly find yourself shorthanded. Again, please respect the \$7F limit.

By the way, the Senior Prom is a hacker's wet dream. Any aspiring Cracking Captain should be advised to beg, borrow, or pawn their grandmother for one of these.

David Todd
Cambridge, MD

Mystery

Boxed in the next column is this month's "Mystery Hexdump." You may have noticed Hexdumps appearing in previous issues of COMPUTIST with no explanation as to what they are or where they came from. The first of these was in COMPUTIST No. 35 on page 29.

Search your back issues for these strange and elusive hexdumps. You never can be quite sure as to what they will do.

This month's "Mystery Hexdump" becomes a normal BASIC program that can be RUN. A word of warning though, do NOT attempt to edit any lines of the program or you will turn into a birdcage liner!

Look for more of these bizarre hexdumps in future issues of COMPUTIST. The results could be wild.

```
0800: 00 24 08 0A 00 86 52 4E $35C9
0808: 44 28 34 29 3A 81 44 41 $F77B
0810: 54 41 D0 30 C1 34 3A 87 $92AF
0818: 52 4E 44 28 44 41 54 41 $AD4E
0820: 29 3A 82 00 44 08 14 00 $F0C0
0828: 47 45 54 D0 C9 31 36 33 $159D
0830: 38 34 3A 48 54 41 42 D0 $4E01
0838: 31 34 30 3A 56 54 41 42 $3309
0840: D0 39 36 00 86 08 1E 00 $FDE3
0848: 97 3A BA 22 50 4F 4C 41 $D305
```

```
0850: 52 20 50 52 49 4E 54 53 $95AB
0858: 3A 22 3A A2 34 3A 84 22 $5529
0860: 4E 55 4D 42 45 52 20 4F $5062
0868: 46 20 50 52 49 4E 54 20 $3FC5
0870: 28 33 2E 32 20 49 53 20 $65E8
0878: 47 4F 4F 44 29 3F 22 3B $D946
0880: 50 52 49 4E 54 00 C2 08 $065F
0888: 28 00 BA 3A 84 22 53 50 $58F2
0890: 45 45 44 20 28 31 2D 35 $911A
0898: 30 30 2C 20 31 30 30 20 $E703
```

```
08A0: 49 53 20 47 4F 4F 44 29 $BC9A
08A8: 3F 22 3B 53 50 45 45 44 $6A3F
08B0: 3A 53 50 45 45 44 D0 53 $26C0
08B8: 50 45 45 44 CB 31 30 30 $BF84
08C0: 30 00 E7 08 32 00 91 3A $83C7
08C8: 92 33 3A B9 C9 31 36 33 $2692
08D0: 30 32 2C 30 3A 93 48 54 $0C0A
08D8: 41 42 2C 56 54 41 42 3A $947C
08E0: 53 54 45 50 D0 30 00 28 $FD52
08E8: 09 3C 00 53 54 45 50 D0 $A188
```

```
08F0: 53 54 45 50 C8 53 50 45 $4C97
08F8: 45 44 3A 50 4F 50 D0 39 $A11F
0900: 35 CA DF 28 50 52 49 4E $2607
0908: 54 CA 53 54 45 50 29 3A $BD2E
0910: 58 43 4F 53 D0 28 DE 28 $D1FD
0918: 53 54 45 50 29 CA 50 4F $6B0C
0920: 50 29 CB 31 2E 37 33 00 $094B
0928: 55 09 46 00 59 53 49 4E $B02E
0930: D0 DF 28 53 54 45 50 29 $2C32
0938: CA 50 4F 50 3A 93 C1 48 $226B
```

```
0940: 54 41 42 C8 32 CA 58 43 $7A8F
0948: 4F 53 2C 56 54 41 42 C9 $97BA
0950: 59 53 49 4E 00 80 09 50 $34E7
0958: 00 AD D4 28 59 53 49 4E $A6D7
0960: 29 D1 31 30 CD D4 28 58 $13D2
0968: 43 4F 53 29 D1 31 30 C4 $9C90
0970: B9 32 38 2C 52 4E 44 28 $573D
0978: DB 28 31 29 CA 35 29 00 $2FF0
0980: 9E 09 5A 00 AD E2 28 47 $6167
0988: 45 54 29 CF 31 32 37 C4 $77DF
```

```
0990: BE 49 4E 50 55 54 24 3A $A4E1
0998: 89 3A AB 33 30 00 A6 09 $EEF5
09A0: 64 00 AB 36 30 00 BD 09 $7844
09A8: 6E 00 83 34 32 2C 38 35 $3C45
09B0: 2C 31 32 37 2C 31 37 30 $FFD7
09B8: 2C 32 31 33 00 00 00 00 $84EE
```

AF:BF 09

SAVE POLAR PRINTS

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.

Apple Business Graphics Apple Computer

Jane Arktronics

Visiblend Microlab

Catalyst Quark, Inc.

Gutenberg 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

ABM Muse

Mychess II Datamost

Agent U.S.A. Scholastic

Handicapping System Sports Judge

Odin Odesta

Mabel's Mansion Datamost

Brain Bank The Observatory

Crimson Crown Penguin

Crypt of Media Sir Tech

The Works First Star Software

Cross Clues Science Research

Peeping Tom Microlab

Jigsaw Microfun

Miner 2049er II Microfun

Create with Garfield DLM

Print Master Unision World

Bandits Sirius Software

Operation Frog Scholastic Software

readers' softkey & copy exchange

John Howard's softkey for...

Yet Another F-15 Strike Eagle

MicroProse
120 Lake Front Dr.
Hunt Valley, MD 21031

Requirements:

A copy program that can select tracks
A sector editor

Larry Jasonowicz did a good job of tracking down the protection on F-15 Strike Eagle in COMPUTIST No. 24 page 19. Although my copy was different from his, I was able to deprotect it using the information from his article.

The bytes to change on my copy are as follows: On track \$1F sector \$0, starting at byte \$D4, the code I found was A9 60 8D 0D 6A and the byte to change was byte \$D5 from a \$60 to \$86. On track \$21 sector \$A, starting at byte \$69, the code was A9 C0 8D 01 02 and the byte to change was byte \$70, from a \$C0 to \$DB.

- 1) Format a new disk.
- 2) Copy tracks \$05 and \$06 from a normal DOS disk.
- 3) Copy tracks \$00-\$04 and tracks \$07-\$22 from the original disk to the normal DOS disk.
- 4) Sector edit track \$1F sector \$00, byte \$D4, from \$60 to \$86.
- 5) Sector edit track \$21 sector \$0A, byte \$70, from \$C0 to \$DB. The copy is now COPYAable.



Sandy Eubanks' softkey for...

Championship Lode Runner

Requirements:

Championship Lode Runner disk
A blank disk
A sector editor
The ability to move memory pages 0 to 7 on
Reset (Modified F8 ROM or Senior PROM)
Super IOB 1.5
Hardcore COMPUTIST No. 22

Championship Lode Runner continues the tradition of its predecessor, only the game levels are much more difficult. An additional feature is the ability to save your game and continue it at a later time. This feature was sorely needed by the original release; 150 levels are too much for one sitting. The following softkey will give you that option. Read on.

There have been two softkeys for Lode Runner published in COMPUTIST. Tom Phelps' (issue No. 22) is the one on which this softkey is based. Steve Marvin's softkey (issue No. 28) is for those without a way into the monitor. Being a novice to assembly language, I could not modify Mr. Marvin's code to work with Championship Lode Runner. Having a Senior PROM, however, I could do enough disk snooping to use Mr. Phelps method with just a few modifications. Therefore, you should follow the softkey as listed in Mr. Phelps' article, substituting the step numbers below for its corresponding number in that article.

4) Type in a BASIC program that will run our Championship Lode Runner main file.

```
10 TEXT: HOME: PRINT  
CHR$(4)“BRUN CHAMP.LR.B”
```

5) Insert a blank disk and format it.

```
INIT CHAMP.LODERUNNER
```

8) Type in the following hexdump which will move everything back into place and start our game.

```
2900: A0 5F A9 9B A2 25 20 42 $9A89  
2908: 29 A0 2A A9 60 A2 35 20 $FE2E  
2910: 42 29 A9 40 85 E6 20 F2 $90D6  
2918: F3 A0 40 A9 09 A2 06 20 $5F9D  
2920: 42 29 A0 22 A9 02 A2 07 $CB9F  
2928: 20 42 29 A0 00 B9 00 20 $B8CD  
2930: 99 00 00 C8 D0 F7 B9 00 $47D4  
2938: 21 99 00 01 C8 D0 F7 4C $0EDA  
2940: 00 60 84 01 85 03 A9 00 $F0BD  
2948: 85 00 85 02 A0 00 B1 00 $DA89
```

```
2950: 91 02 C8 D0 F9 E6 01 E6 $7E71  
2958: 03 CA D0 F0 60 $B753
```

9) Check your typing with CHECKBIN and save the listing to another DOS 3.3 disk (other than CHAMP.LODERUNNER).

```
BSAVE CHAMP.LR.MOVES  
,A$2900,L$5D
```

13) When the title screen appears, press RESET and then a “7” (if you're using the Senior PROM) or a “:” (if you have the modified F8 ROM) to move \$0-8FF to \$2000-28FF.

14) While still in the monitor, compact the code.

```
2A00<6000.94FFM  
5F00<9B00.BFFFm
```

17) Install the move routines. (Insert the disk on which you saved the hexdump).

```
BLOAD CHAMP.LR.MOVES
```

18) Insert the disk with the copied game boards and save the main file we just created.

```
BSAVE CHAMP.LR.B,ASEFD,L$7503
```

20) Back it up immediately with COPYA or some other fast copy program.

Obviously, the main changes in the hexdump from Mr. Phelps' softkey are to allow saving of some additional memory areas. We also had to delete part of his routine that would have zeroed those areas we wanted to save. The additional memory locations contained the sound routines present in the Championship version.

Note that it's possible to put both the Lode Runner main file and the Championship Lode Runner main file on either game board disk. Both files search for the game boards on the same tracks, \$03-0C. If you want to do this, you will have to mark the VTOC as Mr. Phelps suggests. Additionally, you'll have change your hello program to allow BRUNning of either file. The advantage of using the Lode Runner file to play the Championship game boards is being able to practice difficult levels with the so-called “cheat” keys. Conversely, using the Championship file to play Lode Runner game boards allows you to save your game and continue it later.

In any case, it is now your option to do as you please... as it should have been all along, Enjoy!



Robert Muir's softkey for...

Gato V1.3

Spectrum Holobyte Inc.
1050 Walnut, Suite 325
Boulder, CO 80302

Requirements:

128K Apple //e or //c (required by game)
At least one drive
COPYA
A sector editor
A blank disk
Gato

Gato is a superb multi-screened submarine simulation game. Since the softkey in COMPUTIST No. 23 did not work for my

readers' softkey & copy exchange

version (1.3) of Gato, I set out to discover what it was about. I discovered that Gato was programmed in Pascal and that the only protection my copy had on it was to change the address header and address epilogue. On odd tracks the headers are D4 AA 96 instead of the normal D5 AA 96 (there was no "nibble count" on my version). This can be circumvented easily by using the following code to read address headers:

```
B94F- BD 8C C0 LDA $C08C,X
B952- 10 FB BPL $B94F
B954- 4A LSR
B955- C9 6A CMP #$6A
B957- D0 EF BNE $B948
```

This may look familiar because it is the same trick used to read the alternating address headers in Penguin Software releases. The bytes D4 and D5 will both turn into the value 6A when shifted to the right, so they will both be considered the same byte when read.

The altered address epilogues can be ignored by not checking for them.

The following softkey will make a COPYAable copy of Gato. It can also be CATALOGed using the //c utilities disk and probably read using Pascal (I'm not sure because I'm not willing to spend what little money I have on yet another language).

- 1) Turn on your computer and boot DOS 3.3.
- 2) Alter the Read Address Header routine.

CALL -151
B954:4A C9 6A D0 EF

- 3) Tell DOS to ignore the address epilogue.

B98B:18 60

- 4) RUN COPYA and copy Gato to a blank disk.
- 5) Run your favorite sector editor and make the following sector edits:

Track	Sector	Byte	From	To
\$0	\$E	\$F8	\$38	\$18
\$0	\$F	\$80	\$38	\$18

The first modifies the second stage boot to ignore address epilogues. The second modifies the final Pascal DOS to ignore address epilogues.

- 6) Write the sectors back to the new copy.

That will make a COPYAable version of GATO!

Larry Rando's softkey for...

I, Damiano

Bantam Electronic Publishing
666 Fifth Avenue
New York, NY 10103

Requirements:
Fast disk copier
A sector editor

I, Damiano is an adventure based on a popular fantasy trilogy and is part of Bantam Electronic's "Living Literature" series. The screen format has a different look opposed to the typical full screen graphics. The hi-res pictures are on the upper half of the screen with text residing on the lower half of the screen. It includes some primitive animation. Overall it is a fun adventure game to play, especially if you are familiar with the book in which the adventure is based.

The Protection

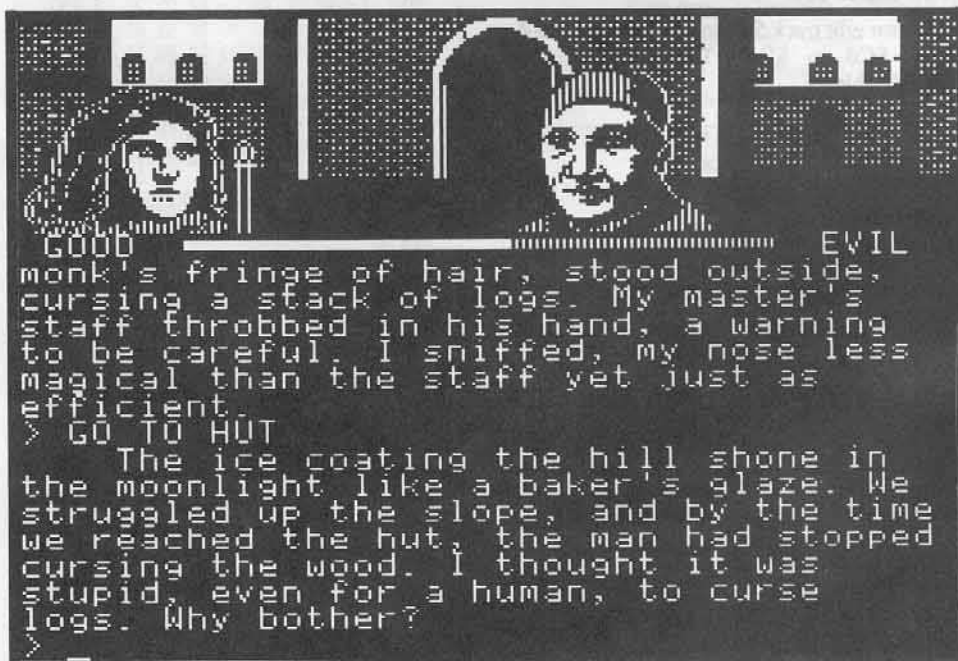
When I used a fast copier to copy the disk, I only got one read error, on track \$11. That told me that the disk was protected in some way. When I booted the disk the head did some kind of long movement and stayed on one track for a while. It was doing this because the nibble count went to Track \$11 to check the disk. If it was a copy it would come back with a

message saying "DISK READ ERROR". I proceeded to search for the text DISK READ ERROR to see where it was on on the disk. I found the message on track 0, sector 0. I then set up a boot code trace to find out what caused the program to print this message; in other words there had to be some sort of nibble count routine that checked Track \$11 for something and then returned to display the message "DISK READ ERROR." I found the nibble count on track 0, sector 0. The protection code looks like this:

```
0821- JSR $083F -Calls nibble count.
0824- LDA #$00 -Normal loader routine.
0826- ASL
0827- LDY #$02
0829- JSR $09C0
082C- JSR $0A33
082F- INC $0AB1
0832- INC $0A62
0835- LDA $0A62
0838- CMP #$0A
083A- BNE $092C
083C- JMP $09A7 -end of loader.
083F- LDA #$40 -Actual nibble counter.
```

The Procedure

- 1) Use any fast disk copier and copy both sides. Ignore the read error on Track \$11 (second side unprotected).
- 2) Get out your sector editor and edit side 1, track 0, sector 0, starting at byte \$21 from \$0 3F 08 to EA EA EA. This eliminates the JSR to the nibble count.
- 3) Write it back out.
That's it!



readers' softkey & copy exchange

Charles Taylor's softkey for...

Wilderness

Electric Transit
501 Marin Street
Suite 116
Thousand Oaks, CA 91360

Requirements:

Apple II
Four blank disk sides
Super IOB
Copy II Plus

WILDERNESS is a unique adventure simulation game. Imagine that you have just crashed your airplane somewhere in the high Sierras. You manage to escape from the wreck with a few survival items, including a topographical map, a compass, tent, some tools, and some food. You see a ranger station on your map, but your crash-site could be anywhere, and the map covers hundreds of square miles. In order to survive, you will have to battle the elements, deal with the various wildlife, such as bears, snakes, cougars, and insects, and ford numerous streams and lakes. In addition you must eat, sleep, camp, avoid injury, exhaustion, overheating and exposure. This is just one of the scenarios available to you if you play WILDERNESS.

The game uses a three-dimensional graphics generating system, a series of accurate models of weather patterns, and a 300 word vocabulary with which to negotiate the adventure. WILDERNESS is accurate down to the most mundane details, such as hanging your food from a branch at night so the bears don't get it, correcting your compass readings for the difference between true north and magnetic north, and putting your fire out before breaking camp.

While the game is sophisticated, the copy protection is not, consisting of four protected tracks (0-3). Any bit copier can deal with this program, but the fame and fortune of being a COMPUTIST contributor beckons, so here goes.

Since we will replace their DOS with DOS from the 3.3 system master, we only have to deal with one protected track (3). Track three has the address epilogue changed from DE AA to AA DE. Once track three is converted we find that there is a normal catalog listing of three short files on track 3, including a boot program

called BOOT and a special DOS image called SMALLDOS. The normally formatted tracks 4-22 run under SMALL DOS.

The game consists of two sides, the JOURNEY side and the SIERRA NEVADA side. Separately available is a third side, called the GLOBAL EXPLORER. This disk gives you five more countries in which to adventure. The JOURNEY disk is unprotected, and the SIERRA NEVADA and GLOBAL EXPLORER sides use identical protection schemes.

Step By Step

- 1) Boot your DOS 3.3 system master disk.
- 2) Type INIT BOOT and init two blank disks (one if you don't have GLOBAL EXPLORER.) Label these disks SIERRA NEVADA and GLOBAL EXPLORER.
- 3) Using Super IOB (the Controller Writer helps) copy track \$3 of both the GLOBAL EXPLORER and SIERRA NEVADA sides to their respective INITed disks, changing the address epilogue from DE AA to AA DE.
- 4) Again, using Super IOB, copy tracks \$4 through \$22 from the SIERRA NEVADA and GLOBAL EXPLORER to the INITed disks. Use the default (normal) marker settings this time. The Controller below will perform both steps 3 and 4 in one run for you.
- 5) Use COPYA to copy the JOURNEY side to the back side of each of your new copies. (The GLOBAL EXPLORER and SIERRA NEVADA sides are independent of each other, but both use the JOURNEY side.
- 6) On my copy of GLOBAL EXPLORER, I got an error on sector \$F of track \$0F. I used the sector editor of COPY II+ patched to custom DOS, ignored data checksums and data epilogues to read the protected sector, then wrote it back to my copy. If you are not copying GLOBAL EXPLORER at this time, you may find it easier to use the sector editor from COPY II+ for tracks \$4-\$22.
- 7) Take a hike! (with your newly softkeyed WILDERNESS of course).

controller

```

1000 REM WILDERNESS
1010 TK=3 : LT=4 : ST=15 : LS=15 : CD=WR : FAST
      =1 : POKE 776 , 0 : GOSUB 170
1020 T1 = TK : GOSUB 490
1025 GOSUB 610 : IF LT = 4 THEN GOSUB 230 : TK = 4
      : LT = 35 : GOTO 1025

```

```

1030 TK=T1 : GOSUB 490 : GOSUB 610 : IF PEEK (TRK
      ) = LT THEN 1050
1040 TK=PEEK (TRK) : ST=PEEK (SCT) : GOTO 1020
1050 HOME : PRINT "COPYDONE" : END
5000 DATA 170 , 222 , 222 , 170

```

controller checksums

1000	- \$356B	1030	- \$E182
1010	- \$F843	1040	- \$5B61
1020	- \$0BAB	1050	- \$0C5D
1025	- \$5506	5000	- \$D2C1

John Howard's softkey for...

Golf's Best

One Step Software

Requirements:

A program that will read non-standard sectors and write them back in a normal format (Super IOB, Copy II Plus, CIA).

Golf's Best is a very good golfing simulator that will let you see the direction your ball is going before you hit it. The overall tee-to-pin graphics display is not as good as some golfing simulators but the approach and screen displays are excellent. The only thing that I would like to see in this simulation is for the players to have honors on tee-off. The game keeps the same tee-off order from start to finish.

The Protection

Tracks \$00-\$02 and even numbered tracks \$04-\$22 are normally formatted and can be easily copied with the Copy II Plus manual sector copy function. Odd tracks \$03-\$21 have altered address headers, trailers, and checksums. By disabling the address header and the checksum using the CIA you can copy all the non-standard sectors to a normal disk. This copy boots and works fine as is.

feature \fē-cher\ n [ME *feture*, fr. MF, fr. L *factura* act of making, fr. *factus*, pp. of *facere* to make—more at DO] 1: the structure form, or appearance esp. of a person 2: a software bug mentioned in its documentation

Creating A Switchable

by Wes Felty

Requirements:

Unenhanced Apple //e
//c enhancement kit
Access to another computer
EPROM programmer that can handle 27128's
One 2764 and two 27128 EPROMs
(additional supplies listed at end of article)

Note: *The procedure described in this article requires modification of your computer, which may void its warranty. COMPUTIST will not be held responsible for any damages incurred while following this procedure. Observe the usual precautions when working with electronic equipment.*

Why create a Dual System?

The folks at Apple Corp. are claiming that more and more software is going to be written for the Apple //c and enhanced //e computers only. Considering the advantages of the faster 65C02 chip, its additional machine language instructions, some improvements in the monitor ROM chips, and the mouse text character set, it is quite likely that many new programs will be developed that have to run on the enhanced computers. But if you get your present //e enhanced, it will scramble some of your old software and not run some of it at all. If you are just starting out and buying all new software, then you may not need a dual system. But if you have old software like PFS, AppleWriter, ASCII Express Pro (DOS version), etc., then you need a dual system to be able to run the old unenhanced software and the new enhanced software.

PFS:FILE and PFS:WRITE programs released before November, 1985 are severe victims of the scrambling that the //c and enhanced //e Apple computers do to some older software. All of their command prompt lines, field category prompts, and underlined or boldfaced text are replaced with mouse text icons. AppleWriter //e's DOS 3.3 version has its data line partially scrambled. The DOS 3.3 version of Sensible Speller also has its command

lines scrambled. In fact, any program that uses inverse characters anywhere in upper and lower case may fall victim to this scrambling. If the program has a solid, non-blinking or a custom cursor, it probably will get all of its inverse upper case characters converted to Mouse Text icons. By the way, I have patches available for fixing AppleWriter, PFS:FILE, and PFS:WRITE to run on enhanced //cs.

Some other programs will not run properly on the enhanced //e due to monitor ROM problems. My version of ASCII Express Pro will boot up, but then later lock up after making contact, usually long distance, but before I have done any communicating. (United Software will give free updates to some users depending on the serial number of their disk). My Ultra ROM board with GPLE built into it just freaks out. These problems of hardware and software incompatibility are usually due to the programmers having used "unofficial" jumps into monitor routines. The unsupported jumps show up in most non-trivial programs and in many cases were needed to get around slow operation or bugs in the original //e ROMs. But you don't have to buy new versions of hardware and software to use with the enhanced //e. Just create a switchable enhanced/unenhanced //c and be able to use all of your old and the newly developed hardware and software.

Some programs will not boot up at all on the //cs or enhanced //es due to the programmer's "protection" to keep their programs from running on "clones." This "protection" on programs like the "Disk Scanner" are the height of ridiculousness since they worked by detecting a bug that was known to be in the Apple's 6502 CPU that Apple had never bothered to fix but that some of the clones had. These old programs will have to be patched to run on any enhanced //e since my switchable system doesn't do anything to the new 65C02 that replaces the 6502 CPU chip.

How to tell if your Apple //e is Enhanced

It is quite simple to tell an enhanced //e from an unenhanced one. First of all, it probably says "Enhanced" right above the green power-on light. Also, an enhanced //e says "Apple //e" when cold booted and an unenhanced one says "Apple II". With an enhanced //e, you can enter non-DOS commands like "call -151" in lower case without generating a "SYNTAX

ERROR". And finally, from the monitor prompt ("*"), entering "!" puts you into the mini-assembler with its "!" prompt.

How the Enhanced/Unenhanced Switching system works

The main thing that makes the switching system relatively simple is the fact that the Apple //e uses chips that are completely compatible with 2764 EPROMs for the monitor ROMs and a 2732 EPROM for the Video ROM. Therefore the "official" Apple chips can be directly replaced with EPROMs with no changes needed to the computer's circuit board. Therefore, the "official" chips can be replaced back into the computer easily at any time.

The other main fact that simplifies constructing a dual system is the fact that a 27128 EPROM can act just like two 2764 ROMs rolled into one chip and a 2764 EPROM can act like two 2732s in one chip. A hardware switch can connect a select pin on the EPROM to either +5 Volts to activate one bank or to ground potential to select the other bank. The 65C02 chip can't be bank switched, but then it doesn't need to be since a 65C02 can be used in any of the Apple // line of computers.

When an Apple //e is made enhanced, four of its chips are replaced; the 6502 CPU, the Video ROM, and the CD and EF monitor ROMs. The switchable system has both the old code for the unenhanced computer and new code for the enhanced computer burned into double sized memory chips replacing the Video ROM and the two monitor chips. The monitor chips have the same pinout arrangement as the original Apple chips, so they are directly replaceable.

With the Video ROM a 28 pin chip has to replace a 24 pin chip, so a more complex adapter socket is required. Wires run from the three new chips to a switch mounted on the back of the computer to switch between enhanced/unenhanced. When the switch is in the enhanced position, the computer looks internally like a normal enhanced Apple //e. When the switch is in the unenhanced position, the computer looks exactly like an unenhanced //e except for having a 65C02 chip in place of the 6502 CPU. The I.D. bytes and everything else that distinguishes each system is fully intact for both switch settings.

Enhanced/Unenhanced //e

If your only problem with the enhanced //e comes from the mouse text characters scrambling upper case inverse letters as in the PFS software, then you can greatly simplify the modifications given below by just following the procedures for the Video ROM. If you did just that part and then later found the need for the full conversion, it would not be any harder to finish the job with the two monitor ROMs at a latter time.

What Has to Be Done

To create a dual enhanced/unenhanced system, you need to

- 1) Capture the code from the old CD, EF, and Video ROMs (**before** you have the computer enhanced!)
- 2) Capture the code from the new CD, EF, and Video ROMs
- 3) Burn the two sets of code into the double capacity ROMs
- 4) Construct adapter sockets for bank switching the two monitor ROMs
- 5) Construct an adapter socket for the Video ROM to:
 - a) Bank switch between enhanced and unenhanced
 - b) Connect a 28 pin chip into a 24 pin socket
- 6) Replace the 6502 CPU chip with a 65C02 chip
- 7) Connect the adapter sockets to the switch
- 8) Replace the original chips with the dual chips
- 9) Test the new chips

Each of the above processes will be explained in detail below.

Capturing the ROMs' Code

There are at least three ways to capture the code from the unenhanced and enhanced monitor ROMs. The code from the EF ROM can be just moved down in memory with the monitor move operation:

1000<E000.FFFM

and BSAVED from there. Unfortunately the CD ROM is closely tied with the I/O area. Depending on the settings of some soft switches, it may be visible or invisible when you try to read the \$Cxxx area. Also, some soft

switches go crazy if you try the monitor move operation on the ROM and accidentally trip them.

Another way to capture the codes is to use Don Lancaster's "Snatchmon" routines. These routines take care of the soft switches and everything and essentially download the code into RAM memory. Copies of these programs have been published in the November, 1985 issue of Modern Electronics, an issue of Computer Shopper, and in the Synergetics (Don Lancaster's company, address below) Absolute RESET mod package (\$19.50). If you called the Synergetics helpline at (602) 428-4073, Don Lancaster would probably mail you a free copy, but I highly suggest that you buy from him his Absolute RESET package. It is way underpriced at \$19.50 for the value given. It is full of valuable information on EPROMs, it tells you how to build personality modules to let a 2732 EPROM burner program 2764s and 27128s, it gives you the Snatchmon programs, and it tells you how to make a small change to the monitor ROM to give you a way to RESET out of ANY program.

While this second method will allow you to capture the monitor ROM's code, there isn't any way to recover the code stored in the Video ROM from the computer that it is running on. The only way to recover this code, and therefore the best way to recover the code from all three chips, is to use an EPROM burner that can download code from a chip. Using another computer (you can't use your computer when you remove the ROMs!), download the code from the Video ROM, the CD ROM, and the EF ROM and save it to disk. Include in the disk file name something to tell you if these codes were enhanced or unenhanced. You will find these chips well marked in the two rows of chips closest to the keyboard.

If your //e is already enhanced (your Video ROM will be part number 342-0265, not 342-0133), then you will **have** to find someone else with an unenhanced //e to copy the code for the three chips from. Unfortunately, Apple dealers won't let you keep your old chips when they enhance the computer. If your computer is not yet enhanced, then be sure to copy the unenhanced chips' code before having it enhanced. Getting it officially enhanced is the easiest, though most wasteful, way to get access to the enhanced chips AND the 65C02 CPU

chip that you will need. Probably the most ideal situation would be for a person with an enhanced //e and one with an unenhanced //e working together to both help each other in this project.

Using whatever method works best for you, capture the code from the three chips for both the enhanced and unenhanced computers. Burn both the enhanced and the unenhanced code into a 2764 EPROM for the Video ROM and into 27128s (2728s) for the two monitor ROMs. Be sure to use the same order in all three EPROMs, for example enhanced in the low end and unenhanced in the high end. Therefore, if your EPROM burner expects code to start at \$1000, then:

BLOAD VIDEO.ENHANCED, A\$1000
BLOAD VIDEO.UNENHANCED, A\$2000

and for the monitor ROMs:

BLOAD CD.ENHANCED, A\$1000
BLOAD CD.UNENHANCED, A\$3000

Do the same for the EF code.

If you are starting with an unenhanced //e, and have managed to get the enhanced ROMs from other than an "official" enhancement, then you will need to find a 65C02 to replace the 6502 CPU in your computer. I doubt that you will be able to get one from your "official" Apple dealer any more than he will let you keep your chips or tell you in advance about the software that won't work properly on your //e after you get it enhanced.

The only source that I have found for the 65C02 is The Western Design Center, Inc. (address below) for about \$9.15. They use part number W65C02P-2 and will ship C.O.D.

Constructing the Adapter Sockets

The three special adapter sockets are each constructed by mounting one low profile socket on top of another with the modifications being done between the two sockets. This adds to the strength of the arrangement, prevents damage that can be caused by soldering directly onto the EPROM, and allows the EPROM to be easily removed from the adapter socket for any later reprogramming.

Before we start constructing the adapter sockets, let's be sure we know how to locate the correct pins. Looking at a chip or socket, you should find one end marked with a small

half circle, a dot, a beveled edge, or a number "1". (See Figure 1). If you hold the chip or socket right side up with the marked end toward you, then the pin closest to you on the right side is pin number one. Pin number two is the next farther pin away on the right side. This continues up to pin 12 or pin 14 for 24 and 28 pin chips respectively.

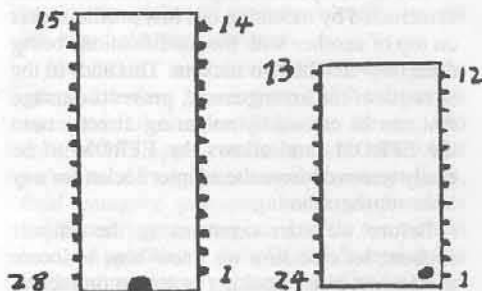
Then the numbers continue from the pin straight across from the last one continuing to increase coming toward you until the highest number pin is at the front left side of the chip. Be sure to check the numbering with the chip or socket RIGHT SIDE UP.

Now let's see how the chips are bank switched. Some of the pins of the 2764 and 27128 EPROMs are bank select pins (address lines). For our purposes this is pin 2 on the 2764 EPROM (Video ROM replacement) and pin 26 on a 27128 EPROM (monitor ROM replacements). If the select pin is brought to +5 Volts, it selects one bank and if it is brought to ground potential, it selects the other bank. The +5 Volts potential comes from pin 28 and the ground potential comes from pin 14 for the 28 pin chips. These are pins 24 and 12 respectively in the 24 pin chips. To set up a hardware bank switcher, we need to disconnect the select pin from contact with the mother board and connect it through a Double Throw switch to connect it to either the +5 Volts pin or the ground pin of the chip (see Figures 2 & 3). We will disconnect the select pin by removing the select pin on the bottom socket so that it doesn't enter the mother board socket when we plug the adapter socket into the mother board.

Constructing the Video ROM Socket Adapter

The Video ROM chip adapter takes more than is shown in Figure 3 because of the problem of plugging a 28 pin chip into a 24 pin socket. Fortunately, the four pins that stick out over the end of the 24 pin mother board socket (1, 2, 27 and 28) are easy to deal with. Pin 1 (VVP line) and pin 27 (PGM line) need to be at +5 Volts for the chip to be read. Pin 28 is supposed to be plugged into the socket's +5 Volt outlet, but that is pin 24 on the 24 pin socket. The chip's pin 26 which is not connected internally is the one that plugs into the mother board's pin 24, +5 Volt supply. Therefore, if

Figure 1



we solder a wire connecting pins 26, 27, 28, and 1 together, the Video EPROM is all set up for a read action, which is what we need. And pin 2, the select pin, would have had to be removed from the lower socket anyway and soldered to the center position of the double throw switch. Therefore, the 28 pin chip in the 24 pin socket comes out being to our advantage. See Figure 4 for the complete Video ROM wiring diagram.

To actually construct the Video ROM adapter socket, start with three pieces of hookup wire about 14 inches long each, a 28 pin socket and a 24 pin socket. Strip enough insulation off of one end of one piece of wire to connect pins 1, 28, 27, and 26. Strip just enough insulation from the ends of each wire to solder to the socket pins or the switch. Tin both ends of each wire. Solder the wires as close to the sockets as you can being careful that the wire doesn't touch the adjacent pin (except for 26, 27, and 28). For the present, just solder them enough to get to stick to the surface of the pins. Be careful to not apply too much heat yet. Some sockets will melt with too much heat and the pins will pull out. Once the wires are "tacked" onto the pins as in Figure 4, install the 24 pin socket onto the bottom of the 28 pin socket so that the pins away from the pin 1 end line up. The lines on Figure 4 from the 28 pin socket to the 24 pin socket are not representing wires, but just the pins of the top socket inserted into the lower socket. In other words, pin 3 of the 28 pin socket goes into hole 1 of the 24 pin socket, pin 14 of the 28 pin socket goes into hole 12 of the 24 pin socket, etc.

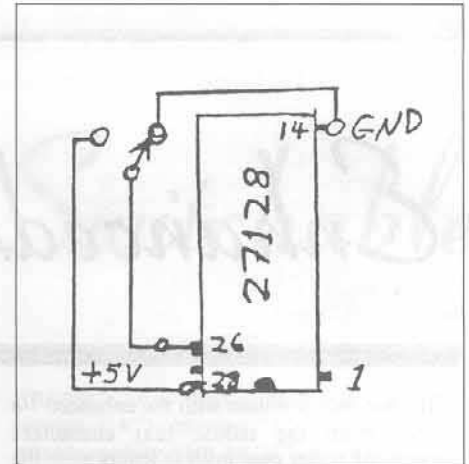
Being careful to not bend the pins on the lower socket, work the pins of the 28 pin socket as deep as you can into the 24 pin one. An alligator clip on each end helps to hold the sockets together while doing this. When you have the two sockets as close together as you can get them, (they will be held apart by the soldered wires), then gently squeeze one end of the socket adapter with a pair of pliers and heat the solder connection on that end of the adapter to permanently solder the wire to the pin.

Heating it well at this time allows the lower socket to push the solder connection firmly up against the top socket and may even melt the two sockets together. Repeat at the other end. Connect the other end of the wire from pin 2 to the center post of a DPDT switch and the other two wires to the outside posts of the same switch. That completes the construction of the Video ROM adapter.

Constructing the Monitor ROM Adapters

These two socket adapters are made from nesting four 28 pin sockets together into two sets of sockets somewhat similar to the Video ROM adapter. Pick two sockets. Solder a 5 inch piece of wire from pin 26 of one of the sockets to pin 26 of the other one. Then solder an 11 inch piece of wire from this jumper wire to the center post on the other (unused) half of the DPDT switch. Next solder wires on pins 14 and 28 on either one of the two sockets. The other

Figure 2



socket will just have the one wire connected to it. Plug each of these two modified sockets into another 28 pin socket.

The lines on Figure 4 from the 28 pin socket to the 24 pin socket are not representing wires, but just the pins of the top socket inserted into the lower socket. On both sockets, heat the solder connections while squeezing the sockets together like with the Video ROM. Connect the other two wires to the DPDT switch being careful to put the wires in the same order as you did for the Video ROM socket. In other words, be sure that the end of the switch that has the pin 14 Video ROM wire also has the pin 14 monitor ROM wire connected to it.

With a pair of needle nose pliers, carefully pull out or break off pin 26 of each of the monitor ROM's lower sockets. And finally, put some little blobs of Epoxy Putty on the pieces of wire to bond the wires to the sides of the adapter sockets for strain relief. Also put some Epoxy Putty on the sides of the adapters to bond the two layers of sockets together. Epoxy Putty can be found in most auto and hardware stores. It consists of a foot long strip of a blue and a yellow material. You tear off a couple of inches of both colors and knead the two layers together. When it is uniformly green, it is ready to use as a sticky, thick, putty like material. If you keep your fingers wet while handling the material, then it doesn't stick to them as much.

If you have some very long cards in slots six or seven, then these double socket adapters may be too high for the cards to fit above them. Just try to use only the top socket soldering as close to the socket's body as possible. While doing the soldering, you want to loosely fit another socket on the bottom of the pins to hold the top pins in place since the heat from the soldering may temporarily loosen the top socket's pins.

Do NOT try to take a short cut and connect all three EPROMs together using one switch. It seems like it should work, but it can blow your mother board. A \$3 DPDT switch is much cheaper than a \$150 mother board.

Installation of the Dual Chips

Be sure to have the power turned off during all chip replacements! First, install the Video ROM EPROM in the first constructed solo

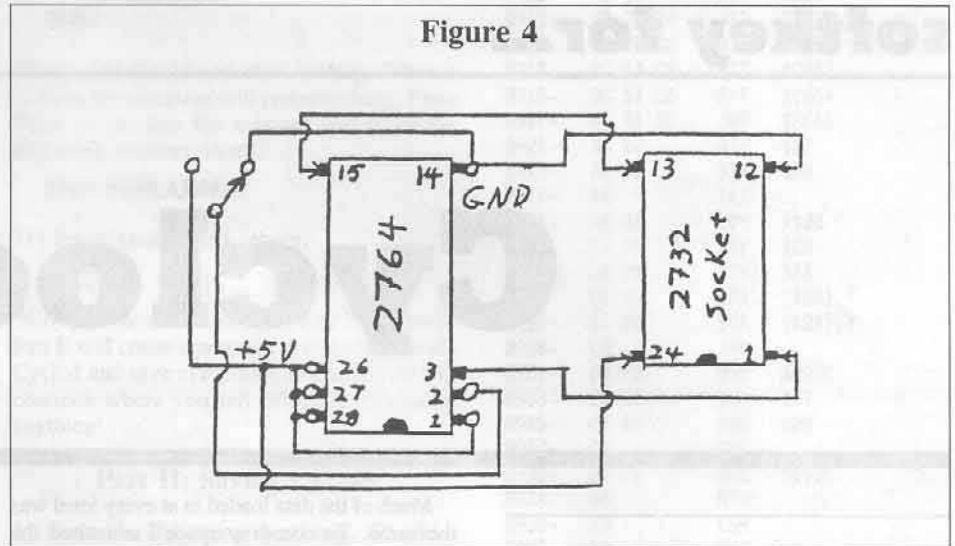
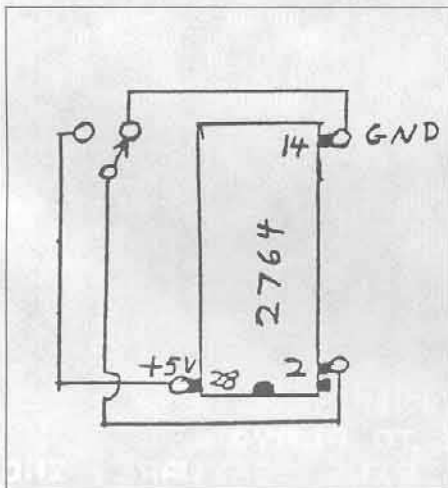
socket. Be sure that the number one marked end of the chip is on the same end as the number one marked end of the socket. Replace the mother board's Video ROM (position F-4 on the mother board) with the EPROM. Be sure that the EPROM's marked end is toward the keyboard. Be careful to not bend under any of the adapter socket's pins while installing them.

Install the CD and EF EPROMs in the two sockets that are connected together. It doesn't matter which one goes in which socket yet. Be sure that they are both positioned with the number one marked end of the chips in the number one marked end of the sockets. Remove the CD ROM (position D-8) and the EF ROM (position D-10) from the mother board and replace them with the respective EPROMs. Be sure that the marked ends of the EPROMs are toward the keyboard. If your Apple //e was unenhanced, then replace the 6502 CPU (position B-4) with a 65C02. That's it! Now for testing.

Testing the Dual System

The first test is a quick one. Open the disk drive door and turn on the computer. Be ready to turn it off immediately if it doesn't beep and start the disk drive. If there is any problem then check that you didn't bend any pins while installing the three adapters or CPU chip. Notice rather the screen displays "Apple][]" for the unenhanced switch setting or "Apple //e" for the enhanced setting. Flip the switch and press Ctrl-Open Apple Reset and you should get the other message. Now use the built in Diagnostic routine. Press **Control-Closed-Apple-Reset**. For the unenhanced setting, the screen should clear and occasionally turn inverse. It should finish with the message "KERNAL OK". The enhanced self test fills the screen with small squares which occasionally flutter and finally, after a long wait, the message "System OK" is displayed. Self test both switch settings. Then try booting a non-important disk in each setting with a cold boot. Also boot a disk with "PR#6" and from the monitor **6** **[P]** in both settings. You may flip the switch while the computer is on but if a program is running, you will find that some

Figure 3



programs, like PFS and AppleWriter, are not hurt by the flip, but many programs will be clobbered. PFS and AppleWriter use their own internal routines instead of the monitor ROM's, but many programs use the monitor ROM's input routines and those are different in the two types of ROMs, so the Apple gets lost. You often have to cold boot after flipping the switch.

Finally, let's make sure that the enhanced Video ROM is on at the same time as the enhanced monitor ROMs. Flip the switch into the enhanced position (Apple //e), cold boot the computer, and turn on the alternate character set by typing

```
PR#3
PRINT CHR$(27)
INVERSE
```

You may also wish to type

```
PRINT CHR$(17)
```

to return to 40 columns with the alternate character set still active. Type some upper case and lower case characters. The lower case characters should show up in inverse and the upper case characters should come out as mouse text icons. Enter "FG" together for the running man. Flipping the switch back and forth should alternate between mouse text and inverse text for the upper case characters. But after flipping the switch a couple of times, you may find that it "flakes out" and needs to be cold booted again. If your computer boots up with "Apple //e" and doesn't give icons with the "PR#3" sequence above, then you need to switch the pin 14 and pin 28 wires on one pole (side) of the DPDT switch. Then run through all of the tests again.

If you have a copy of AppleWriter, or a pre-November, 1985 copy of PFS:FILE or PFS:WRITE, then switch into enhanced and boot any of them up. After noting the "garbage", flip the switch and note the return to english. With PFS:FILE flip the switch back and forth when in the selection menu. This is really impressive if your field prompts are in all upper case letters.

I suggest mounting the switch in one of the holes on the back of the computer, preferably on the right side so that you won't accidentally hit it when turning the power on/off.

Materials Needed:

- EPROM Programmer (burner) capable of handling up to a 27128 chip and capable of downloading code from a chip
- Access to a second computer to use EPROM burner in
- The code from an unenhanced Apple //e's Video ROM, CD ROM, and EF ROM
- The code from an enhanced Apple //e's Video ROM, CD ROM, and EF ROM
- One 2764 EPROM
- Two 27128 (2728) EPROMs
- One 65C02 Microprocessor
- One low profile 24 pin socket
- Five low profile 28 pin sockets
- One DPDT switch (or two SPDTs)
- About two yards of hookup wire
- Some Epoxy Putty (for strain relief)

Synergetics

746 First St. P.O. Box 809
Thatcher, Arizona 85552
(602) 428-4073

The Western Design Center, Inc.

2166 East Brown Road
Mesa, Arizona 85203
(602) 962-4545
TELEX 6835057

Wes Felty

2628 133rd Place S.E.
Bothell, Washington 98012
(206) 337-3720

This whole procedure is very complex, but then it is a pretty sophisticated change being made to the computer. If you don't feel like tackling the whole construction process, then you may wish to contact the author about the availability of pre-constructed setups.

Cyclod

By Felix LeChat

Sirius Software
(retired)

Requirements:

- Two blank disks
- A sector editor
- Any bit copy program

Cyclod is another game from Sirius who, as we all know, love(d) to protect their products. Cyclod is not the greatest game around because it does get repetitious, but cracking it is the most challenging part. Even the most up-to-date bit copiers can't seem to copy Cyclod and there's enough disk access to keep NMI users away. However, after some delicate snooping and examination, it can be saved as an ordinary binary file.

You won't need any special little tools to crack Cyclod, except for a nibble copier. This procedure is fairly long and involved because of the level of protection. I split the softkey into two parts: getting Cyclod into memory and saving Cyclod.

The Protection

Sirius protected every track except for track 0. It uses 4+4 encoding, checksums of both the boot program and the main program, and resets the page 3 vectors. To start off I bit copied what I could onto another disk, that is, just track 0. After boot code tracing I was able to capture the boot program and defeat the checksum routines. This code lies in the text page but modifying track 0, sector 0 (to move it elsewhere) allowed me to examine this code. This boot program simply loads data and the program and JuMPs to the start of Cyclod at \$8EA6. When Cyclod is played, It constantly JSRs to \$400 to read in data for every level. All I did was load in all the data and the game and stopped the execution. After booting up a slave disk I was able to save the game from a normal DOS 3.3 disk. Now the difficult part was to make the program fit in memory and run correctly.

Much of the data loaded in at every level was the same. To conserve space I crunched the program and got rid of similar data. Then I wrote a short program to uncrunch Cyclod and simulate the boot program to "load in" data. After testing I also had to make a few patches to the main program to defeat checksum routines.

0) First pre-read these instructions and write-protect Cyclod if it isn't already.

Part I: Getting Cyclod

- 1) Initialize your two disks. You may use a fast DOS if you wish. Call one of them Disk I and the other Disk II. Delete the HELLO program on Disk II.
- 2) Bit copy only track 0 of Cyclod onto Disk I.
- 3) Make the following sector modifications to track 0 sector 0 of Disk I.

Warning! Be very careful. Track 0 is very weird. After making the patches, you must write the sector at least three times to assure proper writing. Then read the sector at least twice and

make sure the patches are still there. If you write it only once, chances are the patch won't work.

1st byte	change from	to
\$60	1F 04	B3 08
\$B3	zeros	A0 00 A9 10 99 AB 07 C8 18 69 10 C0 08 D0 F5 A9 A9 8D 4D 04 A9 00 4C E8 08
\$E8	zeros	8D 4E 04 99 AB 07 4C 1F 04

4) Boot Disk I. After you see the hi-res page flicker, remove Disk I and insert your original Cyclod disk. When you hear several beeps and the disk reboots, immediately take out the Cyclod disk and insert Disk II. Don't worry, this process will not harm your original disk.



5) Go into the monitor and make the following memory moves.

```
CALL -151
1000<2000.2050M
1080<3000.3050M
1100<4000.4050M
1180<5000.5050M
1200<6000.6050M
1280<7000.7050M
1300<8000.8050M
```

6) Save this data (this is a shortened form of the data loaded in at every level).

```
BSAVE CYCLOD.DATA,A$1000,L$351
```

7) Insert the Cyclod disk. In the monitor you'll get ready to boot trace Cyclod. The first instruction disconnects DOS, the second clears memory, and the third copies the boot ROM to RAM.

```
0P0K
800:00 N 801<800.9000M
6600<C600.C6FFM
66F8:00
6600G
```

8) Make the following memory patches to bootl and execute it.

```
82F:14 N 859:18 N 86D:69 FF
300:A2 60 86 2B 4C 01 08
300G
```

9) When you see the hi-res screen type the following (you'll be typing blind to start with) to modify Cyclod's boot program. Then execute it.

```
C051
C054
144D:A9 00
146C:A9 69
147A:A9 FF
172C:69 FF
300:A2 60 4C 1F 04
```

```
400<1400.17FFM
300G
```

10) Cyclod should now start loading. When it is done, the computer will probably hang. Press Reset to get into the monitor and make the following memory move.

```
2300<9400.AE00M
```

11) Insert Disk II and reboot.

```
6P
```

Now you have the bulk of Cyclod in memory. Part II will create a program to properly execute Cyclod and save everything in one file. It will continue where you left off so don't change anything!

Part II: Saving Cyclod

1) Enter the following hexdump. Check it against the Listing that follows.

```
900:20 58 FC A0 40 A2 B0 A9
908:10 20 21 09 A0 23 A2 94
910:A9 1B 20 21 09 2C 50 C0
918:2C 57 C0 2C 54 C0 4C A6
920:8E 84 07 86 09 AA A0 00
928:84 06 84 08 B1 06 91 08
930:C8 D0 F9 E6 07 E6 09 CA
938:D0 F2 60 4A 38 E9 01 85
940:19 A0 B0 A2 40 A9 0C 20
948:21 09 A4 19 B9 66 09 85
950:07 B9 6D 09 85 06 A0 00
958:B1 06 99 00 40 C8 C0 48
960:D0 F6 4C A6 8E BC BC BD
968:BD BE BE BF BF 00 80 00
970:80 00 80 00 80
```

```
0900- 20 58 FC JSR $FC58
0903- A0 40 LDY #$40
0905- A2 B0 LDX #$B0
0907- A9 10 LDA #$10
0909- 20 21 09 JSR $0921
090C- A0 23 LDY #$23
090E- A2 94 LDX #$94
```

```
0910- A9 1B LDA #$1B
0912- 20 21 09 JSR $0921
0915- 2C 50 C0 BIT $C050
0918- 2C 57 C0 BIT $C057
091B- 2C 54 C0 BIT $C054
091E- 4C A6 8E JMP $8EA6
0921- 84 07 STY $07
0923- 86 09 STX $09
0925- AA TAX
0926- A0 00 LDY #$00
0928- 84 06 STY $06
092A- 84 08 STY $08
092C- B1 06 LDA ($06),Y
092E- 91 08 STA ($08),Y
0930- C8 INY
0931- D0 F9 BNE $092C
0933- E6 07 INC $07
0935- E6 09 INC $09
0937- CA DEX
0938- D0 F2 BNE $092C
093A- 60 RTS
093B- 4A LSR
093C- 38 SEC
093D- E9 01 SBC #$01
093F- 85 19 STA $19
0941- A0 B0 LDY #$B0
0943- A2 40 LDX #$40
0945- A9 0C LDA #$0C
0947- 20 21 09 JSR $0921
094A- A4 19 LDY $19
094C- B9 66 09 LDA $0966,Y
094F- 85 07 STA $07
0951- B9 6D 09 LDA $096D,Y
0954- 85 06 STA $06
0956- A0 00 LDY #$00
0958- B1 06 LDA ($06),Y
095A- 99 00 40 STA $4000,Y
095D- C8 INY
095E- C0 48 CPY #$48
0960- D0 F6 BNE $0958
0962- 4C A6 8E JMP $8EA6
0965- BC BC BD LDY $BDBC,X
0968- BD BE BE LDA $BEBE,X
096B- BF ???
096C- BF ???
096D- 00 BRK
096E- 80 ???
096F- 00 BRK
0970- 80 ???
0971- 00 BRK
0972- 80 ???
0973- 00 BRK
0974- 80 ???
```

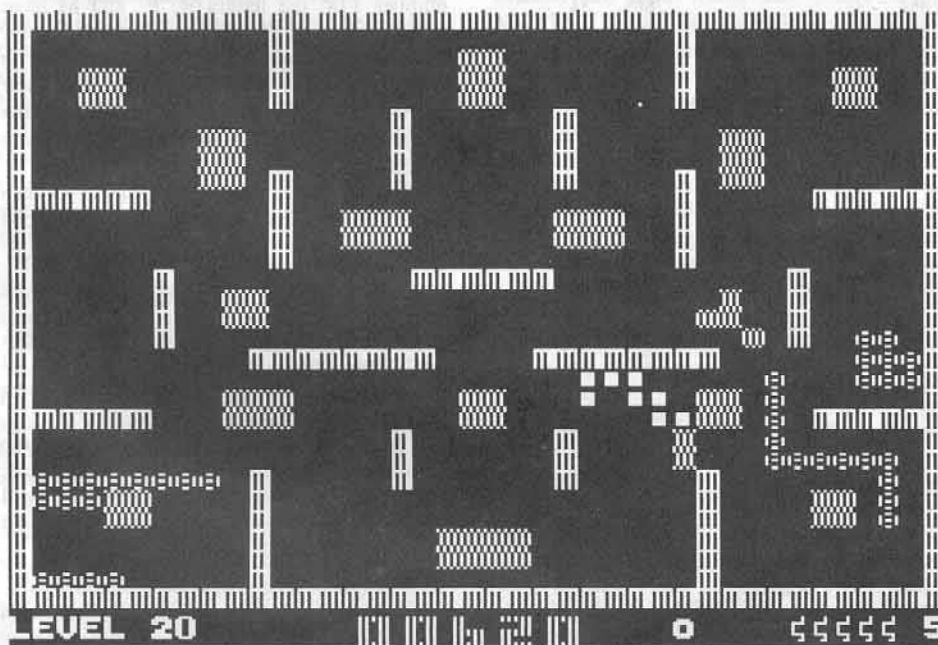
2) Make the following memory patches to the main program of Cyclod:

```
8262:20 3B 09
825E:00 EA EA EA
828F:00
```

3) Save the entire thing and test it out. "A964:FF" allows DOS to BSAVE more than 57F pages.

```
BLOAD CYCLOD.DATA,A$4C00
A964:FF
BSAVE CYCLOD,A$900,L$8D00
DELETE CYCLOD.DATA
```

There! You've got your own unprotected Cyclod. You can re-use Disk I. Cyclod should end up to be a 144 sector binary file.



Appavarex

by Elwood J.C. Kureth

Requirements:

Applesoft
A program to be debugged

One of the minor inconveniences a programmer faces when coding in Applesoft is the fact that Applesoft recognizes only the first two characters of a variable name. COMPARE and COUNT equate to the same variable name as far as Applesoft is concerned, even though the programmer may intend for them to be totally different. This fact makes programming in Applesoft somewhat frustrating because the programmer must keep a much closer watch on what variable combinations have been used.

In other versions of BASIC, KEEPTRACK1 and KEEPTRACK2 would be recognized as separate variable names with (perhaps) different values. While the chances of someone mistakenly duplicating KEEPTRACK1 as a variable name are slim, it would be very easy to unknowingly duplicate the first two letters of KEEPTRACK1, with a variable name such as KEYBRD.

One solution to the problem of keeping track of variable names in Applesoft is to use a lettering system such as AName, BName, CName, etc. Use the first two letters to distinguish the variable name, and use the remaining letters to identify the variable's purpose. This solution certainly works, but even so it's still possible to forget the last letter combination used.

It was for the reasons previously mentioned that I decided to write a short routine that would help me keep track of my variable names. It would list the names and, more importantly, all the line numbers where the name occurred.

First Thoughts

My original idea was to code the program in Applesoft and recode it in machine language. But after creating the Applesoft version, I thought others might like to see how the program works, and at the same time gain an understanding of how a BASIC program is stored in program text. So here, then, is the

BASIC program I call Appavarex, and an explanation of its workings. The machine code version may be presented in a future issue.

About BASIC...

Before we look at Appavarex, let's take a quick look at how Applesoft stores a BASIC program in memory. It's important to understand how it's done if you are to understand the workings of Appavarex.

Let's say, you key the following:

```
10 REM DEMO 20 PRINT "DEMO" 30 FOR X=1 TO 5:PRINT
X:NEXT X 40 END
```

Unless the programmer specifies where the program is to be stored in memory, the computer will begin storage at location \$801, or 2049 in decimal. Since most people are used to working with numbers in decimal format, all numbers will be displayed as decimal. Here's how the program would be stored in memory:

Addr	Val	TrnsI	Addr	Val	TrnsI
2049	12		2060	24	
2050	8		2061	8	
2051	10	10	2062	20	20
2052	0		2063	0	
2053	178	REM	2064	186	PRINT
2054	32		2065	34	"
2055	68	D	2066	68	D
2056	69	E	2067	69	E
2057	77	M	2068	77	M
2058	79	O	2069	79	O
2059	0		2070	34	"
			2071	0	

Addr	Val	TrnsI	Addr	Val	TrnsI
2072	41		2089	47	
2073	8		2090	8	
2074	30	30	2091	40	40
2075	0		2092	0	
2076	129	FOR	2093	128	END
2077	88	X	2094	0	
2078	208	=			
2079	49	1	2095	0	
2080	193	TO	2096	0	
2081	53	5			
2082	58	:			
2083	186	PRINT			
2084	88	X			
2085	58	:			
2086	130	NEXT			
2087	88	X			
2088	0				

In the above table, "Addr" is the address in memory, "Val" is the value stored at that location and "TrnsI" is the BASIC code.

Each line of text that you type (that is, terminated with a RETURN) is called a record in program text. The first two bytes of each record point to the next record in memory. The bytes are in low byte/high byte order. In the example above, the 12 and 8 equate to 2060 (12 + (256*8)). Location 2060 then, is the starting location of the next record. The above table verifies this.

The next two bytes of each record represent the BASIC line number. These two bytes are also in low/high order. In the example above, the 10 and 0 equate to 10 (10 + (256*0)), the first line number used in the demo program.

Each reserved word in Applesoft, such as REM, PRINT, FOR, NEXT and END is represented by a single byte called a token. In the example above, REM is represented by 178. All Applesoft tokens have a value greater than 127.

The remainder of the program is represented by an ASCII value. For example, \$ = 61, A = 65, " = 34 and 1 = 49. As stated in the preceding paragraph, reserved words are represented by a one byte number. Even though the reserved word REM is represented by the number 178, the letters R, E, and M have ASCII values of 82, 69 and 77, respectively. It's only when letters are grouped together to form a reserved word that they lose their individual ASCII identity. Look at the table above to see other ASCII values for program characters.

The end of each record is indicated by a zero byte. Two consecutive zero bytes signal the end of program text.

Enter the Realm

That's essentially all there is to program text. And now, armed with an understanding of BASIC program storage, let us venture forth into the realm of Appavarex.

You will find the complete listing of the Appavarex program at the end of this article, but as we go through the program here I will break it down into sections for ease of explanation. So let's begin.

```
120 DIM NAMES(200), LINES(200)
130 NUM = 1
140 ADDRESS = 2049
```


Line 120 dimensions two subscripted variables that are used in the program. These variables will be explained as we run across them in the program. Line 130 simply initializes a key variable. NUM, which is used as a counting mechanism (to increase the subscript number of the subscripted variables). Line 140 sets the address to be read from to 2049. This is the usual starting place for a BASIC program.

```
150 QUOTE = 0 : D1TA = 0 : R1EM = 0
160 LINE = PEEK (ADDRESS + 2) + 256 * PEEK
    (ADDRESS + 3)
170 VTAB 12 : HTAB 21 : PRINT LINE
180 NXTLINE = PEEK (ADDRESS) + 256 * PEEK
    (ADDRESS + 1)
190 ADDRESS = ADDRESS + 4
```

Lines 150 through 190 are executed before each program line that is scanned for variables. First, the quote flag, data flag and rem flag (QUOTE, D1TA, R1EM) are cleared which means that we are not between quotes, after a DATA statement or after a REM statement. Next, the line number of the current BASIC line is calculated and stored in the variable LINE. The line number calculated in line 160 is then printed on the screen so that you can see that Appavarex is working. NXTLINE is then equated to the starting memory location of the next program line. Finally, 4 is added to the address pointer so that it is pointing to the first byte in the BASIC program line.

```
200 BYTE = PEEK (ADDRESS)
210 IF BYTE = 58 AND NOT QUOTE AND NOT R1EM THEN
    D1TA = 0 : GOTO 260
220 IF BYTE = 34 THEN QUOTE = 1 - QUOTE : GOTO 260
230 IF BYTE = 178 THEN R1EM = 1 : GOTO 260
240 IF BYTE = 131 THEN D1TA = 1 : GOTO 260
250 IF BYTE > 64 AND BYTE < 91 AND NOT QUOTE AND
    NOT D1TA AND NOT R1EM THEN 300
260 ADDRESS = ADDRESS + 1
```

We now enter the main line scan loop. Line 200 simply equates BYTE to the value found at the current address we are scanning.

Line 210 tests for a colon. If a colon is found and it is not between quotes and not after a REM command, then the data flag is cleared. This is done in case the program we are scanning has data and program code on the same line.

Line 220 tests for a quote. If one is found, then the quote flag is toggled (from 0 to 1 or from 1 to 0). During the operation of the program, if QUOTE is equal to 1, then we are between quotes.

Line 230 tests for a REM command. Even though the REM word is tokenized, any characters after it will appear as individual ASCII values. If a REM is found, we must therefore turn on a flag so that the bytes following the REM won't be mistaken as variable names. The rem flag is only cleared by line 150 so it will stay on until then next line is scanned.

Line 240 tests for a DATA command word and for the same reason as the rem flag is turned on, turns on the data flag if one is found.

Line 250 tests to see if the byte we are on is the start of a variable name by first checking to see if it is in the A through Z range (65 -

90) and secondly, making sure we aren't between quotes, after a DATA statement or after a REM statement. If all these conditions are true, then the execution continues at line 300 where the variable and the line it was found on will be collected into the arrays. We'll talk about the routine at 300 later.

Line 260 increments the address pointer so that we can look at the next byte in the program code.

```
270 IF ADDRESS < NXTLINE - 1 THEN 200
280 IF PEEK (ADDRESS) = 0 AND PEEK (ADDRESS+1)
    = 0 THEN 440
290 ADDRESS = ADDRESS + 1 : GOTO 150
```

Line 270 checks to see if we are at the start of the next line yet. If we are not, then execution continues at line 200 where the next byte of program code is evaluated.

When execution hits line 280, we are at the start of a new line. Line 280 therefore determines whether or not there are two consecutive zero bytes. If so, Appavarex has reached the end of the BASIC program and jumps to line 440 where it begins preparing the variables and line numbers for display. More on that later.

At line 290, since we are not yet through with all of the program text, the address is incremented by one (to skip over the end of record zero byte) and execution continues at the main scan loop.

```
300 NAMES$ = ""
310 NAMES$ = NAMES$ + CHR$(BYTE)
320 IF BYTE = 36 AND PEEK (ADDRESS+1) <> 40 THEN
    ADDRESS = ADDRESS + 1 : GOTO 390
330 ADDRESS = ADDRESS + 1 : BYTE = PEEK (ADDRESS)
340 IF BYTE = 40 THEN NAMES$ = NAMES$ + "(*)" :
    ADDRESS = ADDRESS + 1 : GOTO 390
350 IF BYTE < 36 OR BYTE > 90 THEN 390
360 IF BYTE > 37 AND BYTE < 48 THEN 390
370 IF BYTE > 57 AND BYTE < 65 THEN 390
380 GOTO 310
```

This series of lines seeks out the name of the variable we have found and places it in NAMES. Line 300 simply nulls the string that will contain the variable name.

Line 310 adds to the end of NAMES\$ the ASCII character at the current address which is part of the variable name.

Line 320 checks to see if the character we just added to the string was a dollar sign and that it isn't followed by an open parenthesis. If this is the case, then the variable we just got the name for is a string type and execution continues at 390. This line is here so that a command similar to "PRINT D\$A\$" will register both D\$ and A\$ rather than one variable named D\$A\$.

Line 330 increments the current address and sets BYTE equal to the value found at the current address.

Line 340 checks for an open parenthesis. If one is found, then the variable name is an array type and the symbols "(*)" are added to the end of the name. Execution then continues at 390.

Lines 350 through 370 check the appropriate ranges to insure that the next byte is still part

of the variable name. If any of the IF statements in lines 350 through 370 are true, then the next byte is not part of the variable's name and execution continues at line 390.

Line 380 goes to 310 to add the character to the name of the variable since it checked out correctly.

```
390 FOR CHECK = 1 TO NUM - 1
400 IF NAMES$ = NAMES$(CHECK) THEN LINES$(CHECK)
    = LINE$(CHECK) + STR$(LINE) + " " : CHECK =
    NUM : NEXT : GOTO 270
410 NEXT
420 NAMES$(NUM) = NAMES$ : LINES$(NUM) =
    STR$(LINE) + " "
430 NUM = NUM + 1 : GOTO 270
```

Once a character other than a letter or a number has been found, Appavarex has reached the end of the variable name. The program jumps to line 390, where the variable name contained in NAMES\$ is compared with all the variable names found so far.

If the variable is not found, its name is dumped into a subscripted variable called NAMES\$(NUM) and the string equivalent of the line where it was found is put into the array LINES\$ (this occurs in line 420). The counter of the number of variables found is incremented and control goes back into the main scan loop.

If the variable name has already been found, then line 400 adds to the LINES\$ that is associated with the variable name, the string equivalent of the line number where the other occurrence of the variable was found (the current line).

```
440 VTAB 12 : HTAB 1 : PRINT "SORTING DATA" SPC
    (40)
450 FLAG = 0
460 FOR CHECK = 1 TO NUM - 2
470 IF NAMES$(CHECK) < NAMES$(CHECK + 1) THEN 500
480 FLAG = 1 : TEMPS$ = NAMES$(CHECK) :
    NAMES$(CHECK) = NAMES$(CHECK + 1) :
    NAMES$(CHECK + 1) = TEMPS$
490 TEMPS$ = LINES$(CHECK) : LINES$(CHECK) =
    LINES$(CHECK + 1) : LINES$(CHECK + 1) = TEMPS$
500 NEXT : IF FLAG THEN 450
```

Once the end of program text has been reached, the main control loop directs Appavarex to LINE 450. At this point all variable names and their line numbers have been identified. It is now necessary to sort the names in alphabetical order. Lines 450 through 500 are a typical bubble sort routine.

The remainder of the program is pretty straight forward. Lines 510 through 530 ask you what the name of the program you are examining and whether you want the printer on during print out or not.

Lines 540 through 590 simply print the variable names and their corresponding line numbers. One point to note is that if a variable name appears more than once in a BASIC program line, that line number will appear after the variable name the same number of times the name appears in the line. Example:

```
COUNT - 15 8000 8000 8010
```

In this example, the variable name COUNT appears in lines 15, 8000 (twice) and line 8010.

Typing it In

First, type in the Appavarex program and save it:

SAVE APPAVAREX

Next, type in the Install Maker program. When this program is run, it will create a text file that can be EXECed. This text file moves the pointer to the start of BASIC text after any program that happens to be in memory and then RUNs Appavarex.

Using Appavarex

Always type

FP

before using Appavarex! This sets the start of BASIC pointer to \$801 (2049), which is where line 140 is expecting to find the start of the program. If you just

RUN APPAVAREX

it will document the variables used in itself. Try it. You should get:

NAME OF FILE: APPAVAREX

ADDRESS - 140 160 160 180 180 190 190 200
260 260 270 280 280 290 290 320 320 320
330 330 330 340 340

BYTE - 200 210 220 230 240 250 250 310 320
330 340 350 350 360 360 370 370

CHECK - 390 400 400 400 400 460 470 470
480 480 480 480 490 490 490 490 550 560
560 580

D1TA - 150 210 240 250

FLAG - 450 480 500

LINE - 160 170 400 420

LINE\$(*) - 120 400 400 420 490 490 490
490
560

NAMES - 300 310 310 340 340 400 420 520
540

NAMES\$(*) - 120 400 420 470 470 480 480
480
480 560

NUM - 130 390 400 420 420 430 430 460 550

NXTLINE - 180 270

QUOTE - 150 210 220 220 250

R1EM - 150 210 230 250

TEMP\$ - 480 480 490 490 530 530

To document the variables of another program, first type FP and then

LOAD program

where "program" is the name of the program you wish to document. Finally, type:

EXEC INSTALL

This will move the start of BASIC pointer to the end of the program in memory and run Appavarex. When you're done, it is a good idea to type:

FP

again to insure that the start of BASIC pointer is correctly set.

In the Future...

I am currently working on a machine language version of Appavarex. It may be published in an upcoming issue of COMPUTIST. Until then, I hope you have enjoyed this trip down "memory" lane.

Appavarex

```

10 REM APPLESOFT
20 REM PROGRAM
30 REM VARIABLE
40 REM EXTRACTOR
50 REM
60 REM BY
70 REM ELWOOD J.C. KURETH
80 REM
90 REM
100 TEXT : HOME : PRINT "APPAVAREX" : PRINT :
    HTAB 12 : PRINT "BY^ ELWOOD^ J.C.^ KURETH"
110 VTAB 12 : PRINT "NOW^ WORKING^ ON^ LINE"
120 DIM NAME$(200) : LINE$(200)
130 NUM = 1
140 ADDRESS = 2049
150 QUOTE = 0 : D1TA = 0 : R1EM = 0
160 LINE = PEEK (ADDRESS + 2) + 256 * PEEK
    (ADDRESS + 3)
170 VTAB 12 : HTAB 21 : PRINT LINE
180 NXTLINE = PEEK (ADDRESS) + 256 * PEEK
    (ADDRESS + 1)
190 ADDRESS = ADDRESS + 4
200 BYTE = PEEK (ADDRESS)
210 IF BYTE = 58 AND NOT QUOTE AND NOT R1EM THEN
    D1TA = 0 : GOTO 260
220 IF BYTE = 34 THEN QUOTE = 1 - QUOTE : GOTO 260
230 IF BYTE = 178 THEN R1EM = 1 : GOTO 260
240 IF BYTE = 131 THEN D1TA = 1 : GOTO 260
250 IF BYTE > 64 AND BYTE < 91 AND NOT QUOTE AND
    NOT D1TA AND NOT R1EM THEN 300
260 ADDRESS = ADDRESS + 1
270 IF ADDRESS < NXTLINE - 1 THEN 200
280 IF PEEK (ADDRESS) = 0 AND PEEK (ADDRESS + 1)
    = 0 THEN 440
290 ADDRESS = ADDRESS + 1 : GOTO 150
300 NAME$ = ""
310 NAME$ = NAME$ + CHR$ (BYTE)
320 IF BYTE = 36 AND PEEK (ADDRESS + 1) <> 40 THEN
    ADDRESS = ADDRESS + 1 : GOTO 390
330 ADDRESS = ADDRESS + 1 : BYTE = PEEK (ADDRESS)
340 IF BYTE = 40 THEN NAME$ = NAME$ + "(" * ")"
    : ADDRESS = ADDRESS + 1 : GOTO 300
350 IF BYTE < 36 OR BYTE > 90 THEN 390
360 IF BYTE > 37 AND BYTE < 48 THEN 390
370 IF BYTE > 57 AND BYTE < 65 THEN 390
380 GOTO 310
390 FOR CHECK = 1 TO NUM - 1
400 IF NAME$ = NAME$(CHECK) THEN LINE$(CHECK)
    = LINE$(CHECK) + STR$(LINE) + "^" : CHECK
    = NUM : NEXT : GOTO 270
410 NEXT
420 NAME$(NUM) = NAME$ : LINE$(NUM) = STR$(LINE)
    + "^"
430 NUM = NUM + 1 : GOTO 270

```

```

440 VTAB 12 : HTAB 1 : PRINT "SORTING^ DATA" SPC(
    40)
450 FLAG = 0
460 FOR CHECK = 1 TO NUM - 2
470 IF NAME$(CHECK) < NAME$(CHECK + 1) THEN 500
480 FLAG = 1 : TEMP$ = NAME$(CHECK) : NAME$(CHECK
    ) = NAME$(CHECK + 1) : NAME$(CHECK + 1) =
    TEMP$
490 TEMP$ = LINE$(CHECK) : LINE$(CHECK) =
    LINE$(CHECK + 1) : LINE$(CHECK + 1) = TEMP$
500 NEXT : IF FLAG THEN 450
510 HOME : NORMAL : PRINT "APPAVAREX^ -^ by^
    Elwood^ J.C.^ Kureth"
520 VTAB 4 : INPUT "NAME^ OF^ FILE=>" : NAME$
530 PRINT : PRINT : INPUT "TURN^ ON^ PRINTER?"
    : TEMP$ : IF LEFT$(TEMP$, 1) = "Y" THEN PR#
    1
540 HOME : PRINT "NAME^ OF^ FILE:" : NAME$ : PRINT
    : PRINT
550 FOR CHECK = 1 TO NUM - 1
560 PRINT NAME$(CHECK) + "^ ^" : LINE$(CHECK)
570 PRINT "-----"
580 NEXT CHECK
590 PR# 0 : END

```

Install Maker

```

10 PRINT CHR$(4) "OPEN^ INSTALL" : PRINT CHR$
    (4) "WRITE^ INSTALL"
20 FOR A = 1 TO 4 : READ A$ : PRINT A$ : NEXT
30 PRINT CHR$(4) "CLOSE"
40 DATA "POKE(PEEK(176)+1) * 256, 0"
50 DATA "POKE104, PEEK(176)+1"
60 DATA NEW , RUN^ APPAVAREX

```

Appavarex checksums

10	- \$BADD	310	- \$E1F8
20	- \$9B13	320	- \$9983
30	- \$4D3B	330	- \$2179
40	- \$AD92	340	- \$9D28
50	- \$C899	350	- \$20DD
60	- \$FF65	360	- \$E888
70	- \$A3BF	370	- \$BCA1
80	- \$A900	380	- \$8B82
90	- \$924D	390	- \$5C3A
100	- \$9EB2	400	- \$49AA
110	- \$BC38	410	- \$A200
120	- \$AB0D	420	- \$C40B
130	- \$559A	430	- \$2E92
140	- \$D90B	440	- \$A564
150	- \$26B6	450	- \$479E
160	- \$60B6	460	- \$B627
170	- \$E9E6	470	- \$9AC9
180	- \$B9CD	480	- \$CE62
190	- \$B166	490	- \$B715
200	- \$E067	500	- \$6776
210	- \$F42F	510	- \$A510
220	- \$117B	520	- \$EC34
230	- \$B0A2	530	- \$B8A4
240	- \$493D	540	- \$7858
250	- \$5ED4	550	- \$FEE3
260	- \$1B05	560	- \$3A63
270	- \$368A	570	- \$D237
280	- \$FD47	580	- \$9B27
290	- \$61CF	590	- \$D80C
300	- \$8A03		

Alternate Reality

by Stephen Lau

Datasoft Inc.
9421 Winnetka Ave
Chatsworth, CA 91311

Requirements:
Apple II with 64K
Super IOB v1.5

Alternate Reality is the first in a new series of fantasy games by Datasoft. Because of the large amount of disk access during the game, I decided to make a backup copy of it. EDD will produce a copy that will constantly tell you to insert the original disk during play. This then prompted me to deprotect the program.

Going About It

Using the CIA, I found that track 0, sector 0 of both sides are normal and all the other sectors on the disk have a bad field data checksum. Next, I booted the disk, Resetted into the monitor when the text logo appeared and scanned through memory.

I found a working copy of the RWTS starting at \$800 instead of \$B800. When I compared this RWTS with the normal one, I found that only one significant byte (in the translate table) had been changed. This then explained the bad data field checksum.

I then created a Super IOB controller that read with the strange translate table and wrote with the correct one. Unfortunately, the resulting disk wouldn't boot.

The Nitty Gritty

It was then time to get tough with the disk, so I challenged it to a boot code trace. I quickly found the code that modifies the normal translate table at \$36C-\$3D5 (created by a routine in the disk controller card before the boot process) to read the strange sectors on my Alternate Reality disk. After NOPing the bytes that alter the translation table, the disk booted all the way to the main menu and the dreaded "Insert the original disk" message appeared.

My Replay card then told me that the program was running somewhere near \$68AE.

Scanning this code, I found the following:

```

$686F: BIT $C054
        BIT $C050 ; DISPLAY HI-RES
        BIT $C057 ; PAGE 1
        BIT $C052 ;
        BIT $C081 ; WRITE ENABLE
        BIT $C081 ; THE RAM CARD
        JSR $6F6E ; CLEAR HI-RES SCREEN
        JSR $744F ; CHECK THE DISK
        BCC $68B9 ; IF OK THEN
                    BRANCH TO $68B9
    
```

To defeat this, I changed the JSR \$744F to a JMP \$68B9. This worked until I entered an inn. Using similar techniques, I discovered that another disk check was at \$D2F9 which does a JSR to \$DE5E. Removing this in a similar fashion produced a seemingly perfect backup.

I then discovered that the "Backup" function provided on Alternate Reality would not work. So, after a little more work I came up with a few sector edits that fixed this minor difficulty. I then incorporated all these sector edits into my Super IOB controller and the resulting disk worked perfectly!

Step by Step

1) Capture the RWTS by boot code tracing the original disk.

```

CALL -151
9600<C600.C6F7M
96F8:A9 4B 8D 61 08 A9 58 8D
9700:62 08 A9 FE 8D 63 08 4C
9708:01 08
9600G
1900<800.FFFM
    
```

Insert your original disk and type:

```

6☐P
CALL -151
2900<B800.BFFFM
2B29<1B29.1BFFM
☐B
BSAVE ALT.RWTS,A$2900,L$800
    
```

2) Type in the controller at the end of this article and use it to deprotect Alternate Reality.

3) Remember to format the boot side with a volume of 2 and the other side with a volume of 3.

That's it, have fun with your backup!

controller

```

1000 REM ALTERNATE REALITY
1002 TEXT : HOME : PRINT "BOOT^ SIDE^ OR^ SIDE^
        TWO^ ?^ " ;
1003 GET A$ : IF A$ <> "B" AND A$ <> "2" THEN 1002
1004 BK = 0 : IF A$ = "2" THEN BK = 1
1006 POKE 775 , 96 : ONERR GOTO 550
1010 TK = 0 : LT = 35 : ST = 15 : LS = 15 : CD = WR : FAST
        = 1
1020 GOSUB 360 : GOSUB 490 : GOSUB 610
1021 IF BK THEN 1030
1022 IF TK = 0 THEN GOSUB 5000
1024 IF TK = 14 THEN GOSUB 5010
1026 IF TK = 21 THEN GOSUB 5020
1030 GOSUB 360 : GOSUB 490 : GOSUB 610 : IF PEEK
        (TRK) = LT THEN 1050
1040 TK = PEEK (TRK) : ST = PEEK (SCT) : GOTO 1020
1050 POKE 775 , 176 : TK = 0 : LT = 1 : ST = 0 : LS = 15
        : CD = WR : FAST = 0
1060 GOSUB 490 : GOSUB 610
1065 IF BK = 0 THEN GOSUB 5030
1070 GOSUB 490 : GOSUB 610
1080 HOME : PRINT "COPYDONE" : END
5000 A$ = "3429:96^ 97^ N^ 3449:D6^ D7^ N^
        3496:0^ 1^ N^ 34D6:20^ 21^ " : GOTO 5040
5010 A$ = "29 7B:EA^ EA^ EA^ EA^ EA^ EA^ EA^ EA^ EA^
        EA^ EA^ EA^ EA^ N^ 298E:EA^ EA^ EA^ EA^ EA^ EA^
        EA^ EA^ EA^ EA^ EA^ EA^ EA^ GOTO 5040
5020 A$ = "4D84:EA^ EA^ EA^ F0^ N^ 69F9:EA^ EA^
        EA^ F0^ " : GOTO 5040
5030 A$ = "2707:EA^ EA^ EA^ EA^ EA^ EA^ EA^ EA^ N^
        2727:E9^ E9^ E9^ E9^ E9^ E9^ E9^ "
5040 A$ = A$ + " N^ D9C6G" : FOR A = 1 TO LEN (A$)
        : POKE 511 + A , ASC (MID$(A$ , A , 1)) + 128
5050 NEXT : POKE 72 , 0 : CALL - 144 : RETURN
10010 PRINT CHR$(4) "BLOOD^ ALT.RWTS"
    
```

controller checksums

1000 - \$356B	1050 - \$43E3
1002 - \$B297	1060 - \$0ED6
1003 - \$0C74	1065 - \$D163
1004 - \$6AB1	1070 - \$366D
1006 - \$CD45	1080 - \$F821
1010 - \$A3AC	5000 - \$C1F8
1020 - \$4EE1	5010 - \$09F9
1021 - \$1952	5020 - \$B89C
1022 - \$2FBF	5030 - \$3AF8
1024 - \$AA90	5040 - \$F792
1026 - \$F759	5050 - \$326B
1030 - \$C5CD	10010 - \$E8EF
1040 - \$F8EB	

RAMdisk

by Robert Knowles

Requirements:

Apple //e with extended 80 column card
DOS 3.3 or similar

If you have an extended 80 column card (64K or more) and ProDOS, you may have had a chance to use the /RAM volume ProDOS automatically installs in the auxiliary RAM. Slick, isn't it? Well, if you don't have, don't like, or can't use ProDOS, you can still use the extra RAM as a simulated disk drive under DOS 3.3. This article will show you how to use most of that RAM as a RAMdisk with a short relocatable routine that intercepts the RWTS and fools DOS into thinking there is a disk drive in slot 3 (or wherever).

The Aux RAM

Normally, the auxiliary ("aux" or "card") RAM cannot be used by Applesoft or DOS 3.3. You probably found this out *after* you bought it. Machine language programming is required to get the most out of the aux RAM. This part of the article is not necessary for the proper care and feeding of the RAMdisk, but may help in understanding what's going on.

If you followed through the article "More Rom Running" by Wes Felty (COMPUTIST No. 34), you should be familiar with how the language card built into the //e and //c works. The language card parallels the Apple's ROM space with 12K of RAM, which you can write data to while reading from the ROM, or select to replace the ROM. The auxiliary RAM works in a similar manner. The aux RAM can be thought of as having a second language card and a second lower 48K of RAM. When you set a few soft switches, you can, in effect, replace the RAM on the main board with the RAM on the 80 column card. The following switches affect the read/write status of the banks:

address	name	function
\$C002	RDMINRAM	read from main 48K
\$C003	RDCARDRAM	read from card 48K
\$C004	WRMAINRAM	write to main
\$C005	WRCARDRAM	write to card
\$C008	SETSTDZP	use main lcard/zpage
\$C009	SETALTZP	use aux lcard/zpage

The Apple starts up with main RAM both read and write enabled. When you activate \$C005

by storing a byte there, your program will be able to read from main RAM and write to aux RAM. This is handy for copying data from one bank to the other. However, if you set \$C003, your computer will crash because it will be attempting to run a program that suddenly disappears with the RAM it was originally reading. A program can only run if the computer can read the locations it is stored at. To counteract this, you could duplicate your whole program to the other bank before you switched, but that would be a bit wasteful.

When the 48K section of the aux RAM is selected, (\$C003 and \$C005) the lower 48K of main RAM will be replaced by "the other lower 48K." If the language card area is switched (\$C009), it will be replaced with "the other language card."

Apple Corp. to the Rescue

Luckily, Apple has provided ROM routines to save us some trouble when moving memory and transferring program control between banks. At \$C311 is a routine called MOVE, or AUXMOVE, that works just like the monitor's MOVE routine with the exception that the code is transferred between banks, the direction of the transfer dependent on the carry flag. To operate this, the starting address of the block of memory to be moved is stored in locations \$3C and \$3D. The last address to be moved is placed at \$3E-\$3F, and the target address in the opposite bank is placed at \$42-\$43. If the move is from main to auxiliary, the carry flag must be set. For the other direction, the carry must be clear. Call this routine with a JSR.

If you are really adventurous, the last two soft switches listed in the table above select whether you are using the language card from the main RAM or the aux RAM. At the same time, the zero page and stack (\$000-\$1FF) are switched also. This means that the auxiliary language card area will have its own private zero page and stack! This also means extreme caution is in order when you switch memory around. The other aux RAM-related routine is called XFER (\$C314). When this is called, you specify the address in aux or main memory to go to, and which language card/zero page you want. Some interesting effects can be performed using this. A full discussion of this feature is beyond this article for now. You can read more about it in the //e Reference Manual.

Hopefully, that will help clarify a few things to provide a background for the RAMdisk explanation that follows.

The RAMdisk

The RAMdisk works by simulating a disk drive in slot 3 drive 1, and storing the "sectors" in the auxiliary RAM rather than on a disk. It attaches itself to DOS by making DOS' JSR to the RWTS entry point to the RAMdisk driver instead. When called by DOS, the RAMdisk saves the location of the IOB table and checks to see if the sector requested is from slot 3. If the IOB is not for slot 3, then it restores the processor registers and JuMPs to the original RWTS, pretending nothing has happened.

If the request IS for slot 3, then it checks to see if the drive number is 1 and sets an I/O error if it is not. The "last slot", "last drive", and "last volume" in the IOB are set up and the real fun begins. The track and sector numbers are converted into an address for the aux bank and stored in the "source start" location (\$3C,\$3D) for the ROM routine MOVE (\$C311). The DOS buffer address is copied to the "target address" location (\$42,\$43). Then the RWTS command is checked. If the code is \$02 (write) then the "source" and "target" are swapped so that the data in the buffer will be sent to the aux RAM instead of being fetched from the RAM. If the code is \$01 (read) then

```
*-----*
*                            *
*              RAMDISK DRIVER   *
*              FOR 128K //E OR //C, DOS 3.3  *
*              BY R.A. KNOWLES   *
*-----*
*              COPYRIGHT 1986 SOFTKEY PUBLISHING *
*-----*

C311- MOVE      .EQ $C311      ENTRY PT. FOR CROSSBANK MOVE
BD00- NXT.RWTS .EQ $BD00      LOCATION OF REAL RWTS
B7B8- HOLE      .EQ $B7B8      LOCATION OF 'JSR $BD00'
003C- A1L       .EQ $3C        MONITOR SCRATCHPAD LOCATIONS
003D- A1H       .EQ $3D
003E- A2L       .EQ $3E
003F- A2H       .EQ $3F
0042- A4L       .EQ $42        ALSO USED AS FILENAME POINTER
0043- A4H       .EQ $43

                    .OR $6000
                    .TF OBJ.RAMDISK
```



```

6000: 38      INSTALL SEC
6001: AD 00 9D LDA $9D00    DROP BUFFERS BY LENGTH
6004: E9 A9      SBC #LAST-RAMDISK -OF RAMDISK CODE
6006: 8D 00 9D STA $9D00
6009: AD 01 9D LDA $9D01
600C: E9 00      SBC /LAST-RAMDISK (HI BYTE)
600E: 8D 01 9D STA $9D01
6011: 18      CLC
6012: AD 00 9D LDA $9D00    STORE BUFFER START+$26 AT A1
6015: 69 26      ADC #26
6017: 85 3C      STA A1L
6019: AD 01 9D LDA $9D01
601C: 69 00      ADC #0
601E: 85 3D      STA A1H
6020: A0 A9      LDY #LAST-RAMDISK
6022: B9 39 60 .1 LDA RAMDISK,Y COPY RAMDISK TO NEW LOC
6025: 91 3C      STA (A1L),Y
6027: 88      DEY
6028: C0 FF      CPY #$FF
602A: D0 F6      BNE .1
602C: A5 3C      PATCH LDA A1L    STORE ADDR OF RAMDISK
602E: 8D B8 B7 STA HOLE    IN PLACE OF REAL RWTS' ADDR
6031: A5 3D      LDA A1H
6033: 8D B9 B7 STA HOLE+1
6036: 4C D3 03 JMP $3D3    COLDSTART DOS, RAMDISK IS UP!

6039: 84 48      RAMDISK STY $48    STEAL DOS' SPOT FOR STORAGE
603B: 85 49      STA $49
603D: A0 01      SLOT LDY #1    FETCH SLOT # FROM IOB
603F: B1 48      LDA ($48),Y
6041: C9 30      CMP #$30    SLOT 3?
6043: F0 0F      BEQ SETSLOT YEP, CONTINUE BELOW

6045: A5 49      TO.NEXT LDA $49    ELSE RESTORE A&Y AND RETURN
6047: A4 48      LDY $48
6049: 4C 00 BD JMP NXT.RWTS

604C: A9 40      IOERR LDA #$40    DRIVE ERR
604E: A0 0D      LDY #$0D
6050: 91 48      STA ($48),Y
6052: 38      SEC        SHOW ERR BEFORE RETURNING
6053: 60      RTS

6054: A0 0F      SETSLOT LDY #$0F
6056: 91 48      STA ($48),Y SET 'LAST SLOT'
6058: A0 0D      LDY #$0D
605A: A9 00      LDA #0    TURN ERROR INDICATOR OFF
605C: 91 48      STA ($48),Y
605E: A0 02      DRIVE LDY #2
6060: B1 48      LDA ($48),Y GET DRIVE NO.
6062: C9 01      CMP #$01    DRIVE 1?
6064: D0 E6      BNE IOERR    NOPE
6066: A0 10      LDY #$10
6068: 91 48      STA ($48),Y COPY DRIVE NO. TO 'LAST DRIVE'
606A: A0 0E      VOLUME LDY #$0E    SET 'LAST VOLUME'
606C: A9 FE      LDA #$FE
606E: 91 48      STA ($48),Y
6070: A0 04      GET.TRK LDY #$04    GET DESIRED TRACK NO.
6072: B1 48      LDA ($48),Y
6074: C9 11      CMP #$11    MIN.TRACK?
6076: 90 D4      BCC IOERR
6078: C9 1C      CMP #$1C    MAX.TRACK?
607A: B0 D0      BCS IOERR
607C: A0 05      GET.SCT LDY #$05    IOB SECTOR
607E: B1 48      LDA ($48),Y
6080: 29 F0      AND #$F0    LESS THAN $10?
6082: D0 C8      BNE IOERR
6084: A5 42      CONVRT LDA A4L    A4 IS USED BY DOS,
6086: 48      PHA        SO WE MUST SAVE IT
6087: A5 43      LDA A4H
6089: 48      PHA

```

the swap is not made. After that, the carry is set for the proper MOVE direction and the MOVE routine is called. After the transfer, the program returns to DOS with an RTS.

On the Right Track

The main puzzle in writing the RAMdisk was how best to fake a floppy disk in RAM. I decided to convert the track and sector numbers almost directly into memory addresses. Since there are 16 (\$0F) sectors per track, the track number could be used as the base address to start with. However, it was also logical to keep track \$11 in the continuous block of RAM so that the program would not have to have a special routine to simulate track \$11. Sooo... the high nibble of the track number will be ignored and the VTOC of the "disk" will show the unuseable sectors as unavailable to DOS. The RAMdisk driver takes the track number, ASLs it (shifts left) four times, which loses the four high bits. The sector number is ORed with the result, and that is used as the high byte of the address. It works out so that track \$10 sector \$0 becomes \$0000 in memory, all the way up to track \$1F sector \$F becoming \$FF00.

For this program, I did not use all of the RAM in the auxiliary bank. To keep the routine short, only the memory from \$1000 to \$BFFF is used. The memory above that is in the language card area. The programming headaches involved in copying memory between 4K bank 1 of language card RAM and 4K bank 2 of auxiliary language card RAM were not worth the trouble at the time this was written. On the lower end of RAM, pages 0-1 are reserved for the zero page and stack, and \$400-7FF should be set aside for 80 column text. These locations could be reserved by marking them as "used" in the VTOC, but an easier method is to simply ignore track \$10. This pares the RAMdisk space down to tracks \$11-\$1B in its present form but that's plenty for most purposes.

The Hook

The RAMdisk program consists of two parts; an installer (\$6000 - \$6038) and the RAMdisk driver (\$6039 - \$60E1). The RAMdisk driver is relatively small and could be tucked almost anywhere in RAM (it's written to be location independent) provided you correctly hooked it to DOS.

However, if you're like most people, you already have a favorite program at \$300, and if you're using a speed-enhanced DOS, the best locations inside DOS are taken. This is why I created the installer portion which places the RAMdisk driver between DOS and its buffers. This is accomplished by checking the current location of the DOS buffers, moving them down by the length of the RAMdisk driver (\$A9 bytes), and copying the driver to just above the new location, allowing \$26 extra bytes for the DOS buffer's filename.

To connect the RAMdisk driver to DOS, the installer stores the address of the RAMdisk driver at location \$B7B8, which is the operand of the JSR \$BD00 instruction at \$B7B7, turning

it into a JSR \$xxxx (where xxxx is the location of the driver) instead.

The installer then re-enters DOS through its coldstart vector at \$3D3.

INITIally speaking

The RAMdisk will not accept the RWTS command code for INITing, returning instead with an I/O error. To initialize the RAMdisk, a formatting routine is provided here (Listing #2). It will start by creating two empty catalog sectors in track \$11, enough room for fourteen filenames. It then proceeds to create a VTOC for the disk, making it look just like a regular floppy, with the exception that only tracks \$12 through \$1B plus the unused sectors in track \$11 are marked as free. DOS' file manager will only bother to try to allocate the free sectors, and will only be reading sectors from those in the track/sector lists with the files, so it won't try to use nonexisting sectors.

Keying it In

To get your RAMdisk up and running, first boot up with DOS 3.3 or a fast substitute. Type in hexdump #1 and save it.

BSAVE RAMDISK,A\$6000,L\$E2

Type in hexdump #2 and save it.

BSAVE RAMFORMAT,A\$6000,L\$93

To use the RAMdisk, type (in either order):

**BRUN RAMDISK
BRUN RAMFORMAT**

Notes & Cautions

I hope the RAMdisk will come in handy. Most programs should not have any problem using it. DiskEdit and Super IOB have both been tested on it, and none of your BASIC programs should have any trouble.

Remember, when you modify DOS, the changes made to it are passed on to disks INITIALIZED after the modification. In this case, DOS will carry the patch made to it at \$B7B8, but the RAMdisk driver itself is not inside DOS. Moving it to unused space in DOS like at \$BB00 or the INIT code are good choices.

The formatter program uses the ROM's aux memory move routine routine at \$C311 directly, so you don't need to have the RAMdisk already running to format the disk. Just BRUN it, either before or after you install the RAMdisk.

If you want to use double hi-res graphics, you will need to have the formatter mark all of tracks \$12 and \$13 as used so DOS will ignore the memory from \$2000-\$4000. If you want to use more of the auxiliary RAM as a RAMdisk you will need to add a language card handler to the RAMdisk. This is not as easy as it looks, as you will need to keep track of the language card's status before, while, and after copying RAM. This driver also does not allow for the larger 80 column cards like Applied Engineering's Ramworks card.

A fast easy way to save and restore your RAMdisk is to use Super IOB to copy tracks \$11-\$1B to and from a blank disk. The fast

```

608A: A0 04      LDY #$04      TURN TRACK & SECTOR INTO
608C: B1 48      LDA ($48),Y   HI BYTE OF SOURCE ADDR:
608E: 0A        ASL           (ASSUMING READ)
608F: 0A        ASL           MUL TRACK BY 16
6090: 0A        ASL           ($1x = $x0)
6091: 0A        ASL
6092: C8        INY           POINT TO IOB SECTOR#
6093: 11 48      ORA ($48),Y   MIX WITH 'TRACK'
6095: 85 3D      STA A1H       STORE TRK&SCT IN A1 (SOURCE)
6097: A9 00      LDA #0
6099: 85 3C      STA A1L       SOURCE= $xx00
609B: A0 08      LDY #$08      MOVE BUFR ADDR TO A4 (TARGET)
609D: B1 48      LDA ($48),Y
609F: 85 42      STA A4L
60A1: C8        INY
60A2: B1 48      LDA ($48),Y
60A4: 85 43      STA A4H
60A6: A0 0C      LDY #$0C      IOB COMMAND
60A8: B1 48      LDA ($48),Y
60AA: F0 34      BEQ NOERR     SEEK CMD, IGNORE
60AC: C9 03      CMP #3        INIT OR BAD CMD?
60AE: B0 9C      BCS IOERR     THEN EXIT
60B0: C9 01      CMP #1        READ?
60B2: F0 10      BEQ MAKE.A2   DON'T SWAP IF SO
60B4: A5 3C      LDA A1L       'WRITE' GOES OTHER WAY
60B6: A6 42      LDX A4L       SO ADDRESSES MUST BE
60B8: 85 42      STA A4L       SWITCHED
60BA: 86 3C      STX A1L
60BC: A5 3D      LDA A1H
60BE: A6 43      LDX A4H
60C0: 85 43      STA A4H
60C2: 86 3D      STX A1H
60C4: 18        MAKE.A2      CLC
60C5: A9 FF      LDA #$FF      ADD $FF TO SOURCE (A1)
60C7: 65 3C      ADC A1L       TO CREATE SOURCE.END
60C9: 85 3E      STA A2L
60CB: A9 00      LDA #0
60CD: 65 3D      ADC A1H
60CF: 85 3F      STA A2H
60D1: A0 0C      RD.WRT      LDY #$0C      GET COMMAND CODE AGAIN
60D3: B1 48      LDA ($48),Y
60D5: C9 02      CMP #2
60D7: 20 11 C3   JSR MOVE     CARRY WILL BE SET IF 'WRITE'
60DA: 68        PLA          COPY SECTOR TO BUFFER OR V.VERSA
60DB: 85 43      STA A4H      RESTORE A4 TO PREVIOUS STATE
60DD: 68        PLA
60DE: 85 42      STA A4L
60E0: 18        NOERR      CLC
60E1: 60        RTS
        LAST
        .EN

```

The Ram Formatter

```

*          RAMDISK FORMATTER FOR 'RAMDISK' PROGRAM          *
*          BY R.A. KNOWLES                                  *
*          COPYRIGHT 1986 SOFTKEY PUBLISHING                *
*
C311- MOVE .EQ $C311 COPY MAIN TO AUX RAM
003C- A1L .EQ $3C  MONITOR SCRATCH LOCATIONS
003D- A1H .EQ $3D
003E- A2L .EQ $3E
003F- A2H .EQ $3F
0042- A4L .EQ $42
0043- A4H .EQ $43
*
*          'SECTOR' IS DEFINED AS RIGHT AFTER PROGRAM,
*          BUT CAN BE PLACED ELSEWHERE INSTEAD
*
        .OR $6000
        .TF OBJ.RAMFORMAT

```



```

6000: A0 00   FORMAT   LDY #000   CREATE BLANK SECTOR
6002: A9 00   LDA #000   FOR 2ND (LAST) CAT SECTOR
6004: 99 93 60 .1   STA SECTOR,Y WIPE OUT 256 BYTES
6007: C8      INY
6008: D0 FA   BNE .1
600A: A9 00   LDA #000
600C: 85 42   STA A4L    POINT TO RAMDISK'S
600E: A9 1E   LDA #01E   TRACK $11, SECTOR $E
6010: 85 43   STA A4H
6012: 20 7E 60 JSR MOVIT  MOVE THE 'SECTOR' TO AUXRAM
6015: A9 11   LDA #011   1ST CATALOG SECTOR:
6017: 8D 94 60 STA SECTOR+1 POINT TO SECOND SECTOR
601A: A9 0E   LDA #00E   IN CAT CHAIN
601C: 8D 95 60 STA SECTOR+2
601F: A9 00   LDA #000   POINT TO TRK $11, SECT $F
6021: 85 42   STA A4L
6023: A9 1F   LDA #01F
6025: 85 43   STA A4H
6027: 20 7E 60 JSR MOVIT  COPY THE SECTOR

602A: A9 0F   VTOC     LDA #00F   POINT TO 1ST CAT SECTOR
602C: 8D 95 60 STA SECTOR+2
602F: A9 03   LDA #003   DOS VERSION
6031: 8D 96 60 STA SECTOR+3
6034: A9 FE   LDA #0FE   DISK VOLUME
6036: 8D 99 60 STA SECTOR+6
6039: A9 7A   LDA #07A   122 SECTORS PER T/S LIST
603B: 8D BA 60 STA SECTOR+$27
603E: A9 11   LDA #011   LAST TRACK ALLOCATED
6040: 8D C3 60 STA SECTOR+$30 (NEXT TRACK IS $12)
6043: A9 01   LDA #001   DIRECTION OF TRACK ALLOCATION
6045: 8D C4 60 STA SECTOR+$31
6048: A9 23   LDA #023   PRETEND DISK HAS $23 TRACKS
604A: 8D C7 60 STA SECTOR+$34
604D: A9 10   LDA #010   16 SECTORS PER TRACK
604F: 8D C8 60 STA SECTOR+$35
6052: EE CA 60 INC SECTOR+$37 256 BYTES PER SECTOR
6055: A9 3F   LDA #03F   ALLOW USE OF TRACK $11
6057: 8D 0F 61 STA SECTOR+$7C
605A: A9 FE   LDA #0FE
605C: 8D 10 61 STA SECTOR+$7D

605F: A2 12   BITMAP   LDX #012   SHOW TRACKS $12-$1B AS FREE
        .1   TXA      CONVERT TRACK NUM TO OFFSET
6061: 8A      ASL      -MUL BY 4 BYTES PER TRACK
6062: 0A      ASL
6063: 0A      TAY      PUT IT IN Y
6064: A8      LDA #0FF
6065: A9 FF   STA SECTOR+$38,Y
6067: 99 CB 60 STA SECTOR+$39,Y
606A: 99 CC 60 STA SECTOR+$39,Y
606D: E8      INX
606E: E0 1C   CPX #01C   DONE?
6070: D0 EF   BNE .1     CONTINUE IF NOT

6072: A9 00   LDA #000
6074: 85 42   STA A4L
6076: A9 10   LDA #010   TRACK $11, SECTOR $0
6078: 85 43   STA A4H
607A: 20 7E 60 JSR MOVIT  WRITE SECTOR
607D: 60      RTS      DONE!!!!

607E: A9 93   MOVIT    LDA #SECTOR SET MOVE PARAMETERS
6080: 85 3C   STA A1L   A1=SOURCE START
6082: A9 60   LDA /SECTOR
6084: 85 3D   STA A1H
6086: A9 92   LDA #SECTOR+$FF A2=SOURCE END
6088: 85 3E   STA A2L
608A: A9 61   LDA /SECTOR+$FF
608C: 85 3F   STA A2H
608E: 38      SEC      MAIN TO AUX
608F: 20 11 C3 JSR MOVE  CALL THE MOVE ROUTINE
6092: 60      RTS      AND RETURN TO PROGRAM
SECTOR
        .EN

```

controller is preferable. Make a permanent version of Super IOB for that purpose, and remove the unused routines in it. If you copy a normal floppy to RAM, the RAM will contain a VTOC that allows writing to illegal sectors. To avoid this, first copy the RAMdisk to a floppy disk, THEN you can write to that floppy without trouble.

I have attempted to make this program as flexible as possible so that you will have little trouble customizing it for your own purposes. Watch your step, but have fun!

Hexdump #1: RAMDISK (and installer)

```

6000: 38 AD 00 9D E9 A9 8D 00 $D144
6008: 9D AD 01 9D E9 00 8D 01 $A4E2
6010: 9D 18 AD 00 9D 69 26 85 $3F51
6018: 3C AD 01 9D 69 00 85 3D $12A0
6020: A0 A9 B9 39 60 91 3C 88 $FB8D
6028: C0 FF D0 F6 A5 3C 8D B8 $E225
6030: B7 A5 3D 8D B9 B7 4C D3 $691E
6038: 03 84 48 85 49 A0 01 B1 $120B
6040: 48 C9 30 F0 0F A5 49 A4 $8B6B
6048: 48 4C 00 BD A9 40 A0 0D $2730

```

```

6050: 91 48 38 60 A0 0F 91 48 $6CE1
6058: A0 0D A9 00 91 48 A0 02 $C918
6060: B1 48 C9 01 D0 E6 A0 10 $822F
6068: 91 48 A0 0E A9 FE 91 48 $C272
6070: A0 04 B1 48 C9 11 90 D4 $45D4
6078: C9 1C B0 D0 A0 05 B1 48 $8ECE
6080: 29 F0 D0 C8 A5 42 48 A5 $B149
6088: 43 48 A0 04 B1 48 0A 0A $0F58
6090: 0A 0A C8 11 48 85 3D A9 $A724
6098: 00 85 3C A0 08 B1 48 85 $38D8

```

```

60A0: 42 C8 B1 48 85 43 A0 0C $88B9
60A8: B1 48 F0 34 C9 03 B0 9C $49A2
60B0: C9 01 F0 10 A5 3C A6 42 $1BDD
60B8: 85 42 86 3C A5 3D A6 43 $DDFB
60C0: 85 43 86 3D 18 A9 FF 65 $99CA
60C8: 3C 85 3E A9 00 65 3D 85 $00BF
60D0: 3F A0 0C B1 48 C9 02 20 $98B6
60D8: 11 C3 68 85 43 68 85 42 $73D3
60E0: 18 60      $C59D

```

Hexdump #2: RAMFORMAT

```

6000: A0 00 A9 00 99 93 60 C8 $A753
6008: D0 FA A9 00 85 42 A9 1E $8BE5
6010: 85 43 20 7E 60 A9 11 8D $84B5
6018: 94 60 A9 0E 8D 95 60 A9 $CCA2
6020: 00 85 42 A9 1F 85 43 20 $488A
6028: 7E 60 A9 0F 8D 95 60 A9 $3025
6030: 03 8D 96 60 A9 FE 8D 99 $1A81
6038: 60 A9 7A 8D BA 60 A9 11 $41B2
6040: 8D C3 60 A9 01 8D C4 60 $5C51
6048: A9 23 8D C7 60 A9 10 8D $9E6D

```

```

6050: C8 60 EE CA 60 A9 3F 8D $4E9D
6058: 0F 61 A9 FE 8D 10 61 A2 $C6AD
6060: 12 8A 0A 0A A8 A9 FF 99 $B916
6068: CB 60 99 CC 60 E8 E0 1C $C2CE
6070: D0 EF A9 00 85 42 A9 10 $1167
6078: 85 43 20 7E 60 60 A9 93 $19D6
6080: 85 3C A9 60 85 3D A9 92 $FEC5
6088: 85 3E A9 61 85 3F 38 20 $6C32
6090: 11 C3 60      $D861

```

Boulder Dash I & II

by Randy R. Abel

*Electronic Arts
1820 Gateway Drive
San Mateo, CA 94404*

Requirements:

- Apple //e or equivalent
- A blank disk
- A sector editor
- Fast copier (Locksmith or equiv.)

Boulder Dash is one of the most fun games I own. Originally marketed by Micro Fun and now by Electronic Arts, it now includes a new Boulder Dash II. For those of us that do not have the Boulder Dash by Micro Fun, I will add the softkey to allow a deprotected copy of both games. Boulder Dash I is on the front side of the diskette and Boulder Dash II is on the back.

Electronic Arts' copy protection techniques have always made the art of deprotecting a

challenge. Their method used on Boulder Dash resembles the technique used on most of the other games such as Sky Fox, Archon II, and Adventure Construction Set. This method, as it was called in COMPUTIST No. 21, is Track Imaging. That is, track \$5.5 is an exact image of track \$5, and because track imaging is very hard to reproduce on a standard Apple compatible disk drive we need a way around the protection.

To start with, I needed to find out how close to normal the address and data markers were so I pulled out Locksmith's fast copier and tried a quick check. To my surprise, Locksmith's fast copier picked up all tracks except track \$6. we don't need to worry about this track so ignore the error on track \$6. The same process can be done to the back side with the same indications. Now that we have a disk to play with we can put away the original.

Just for fun I tried to boot the copy but, as you can guess, it did not get very far and started doing weird things. The next thing to check is location \$A000. In all of Electronic Arts' programs so far \$A000 has always been the first check. We can get by this point by making a quick sector edit. With any sector (I used

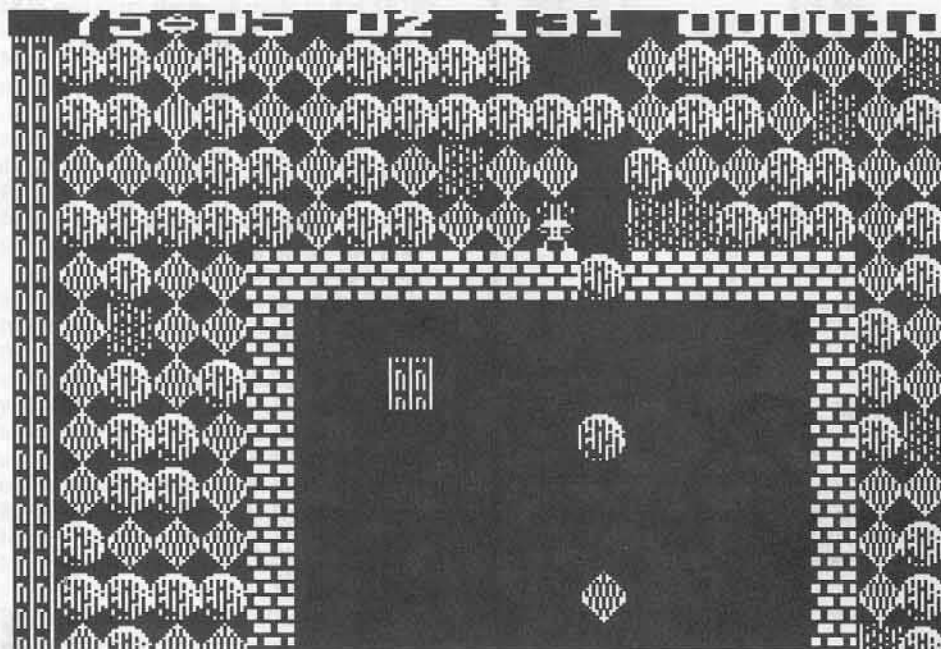
Inspector), read track \$01, sector \$0C. You will find a \$4C at byte \$00. To bypass this first check change the value to a \$60. Now any jump subroutine to \$A000 will return after doing nothing. If you try to boot after this change it still begins to do weird things after a bit. This is because there is a check done just before jumping to the routine to make sure that no changes have been made. To get around this check we need to make another change at track \$01 sector \$7. Here we look at byte \$FF and find a \$0F. Changing this to \$02 will correct the checksum and allow the boot to proceed all the way to the point of selecting either the Apple joystick or a Joypoint. The process of finding the right value to correct the checksum is very lengthy and need never be done again unless the initial boot is changed.

After selecting the device used for play the drive starts up again and does another check. At this point I reset into the monitor. A disassembly at \$13EB will reveal:

```

13EB- A9 14   LDA #$14
13ED- 48     PHA
13EE- A9 04   LDA #$04
13F0- 8D 07 14 STA $1407
13F3- 09 FB   ORA #$FB
13F5- 8D 06 14 STA $1406
13F8- A9 00   LDA #$00
13FA- EE 06 14 INC $1406
13FD- D0 03   BNE $1402
13FF- EE 07 14 INC $1407
1402- EE 05 14 INC $1405
1405- 4C 05 14 JMP $1405 (!)
1408- CE 05 14 DEC $1405
140B- AC 07 14 LDY $1407
140E- C0 07   CPY #$07
1410- D0 E8   BNE $13FA
1412- AC 06 14 LDY #$1406
1415- C0 DF   CPY #$DF
1417- D0 E1   BNE #$13FA
1419- 49 4E   EOR #$4E
141B- 09 2B   ORA #$2B
141D- 48     PHA
141E- A9 04   LDA #$04
1420- 48     PHA
1421- A5 4F   LDA #$4F
1423- 45 4F   EOR #$4F
1425- 18     CLC
1426- E9 00   SBC #$00
1428- 48     PHA
1429- B0 01   BCS $142C
142B- 60     RTS

```



For those who want to know what is happening here, read on. If you want to figure it out for yourselves, skip this paragraph. The routine pushes \$14 on the stack to make the high byte of the return address somewhere in the \$1400 range. From \$13EE to \$1418 they are EORing the range from \$0500 to \$07DF, then at \$1419 we see EOR # \$4E. If you Exclusive-OR a value with itself, the end result will be zero. \$4E is the value we find for the \$0500 to \$07DF range. The next thing that happened is an ORA # \$2B. Since the accumulator had a zero we wound up with the \$2B. this will be the low order address (minus one) the routine will return to. The next part pushes \$04 and the \$FF on the stack. Can you figure out how this is done? The branch on carry set fails and is executed. Where will the routine return to? Right, \$0500. This is where the next check is done for the track image. This check ends with a RTS, so where is it going to return to? Right again, \$142C!

There is no useful data returned to the program in this routine, so why can't we just jump over all that mess and get on with the game? We can by placing at \$13EB a JuMP to \$142C (\$4C 2C 14). To do this we need another sector edit. To speed things up I search the disk with the Inspector (or the CORE Disk Searcher) for the sequence \$A9 14 48. To my surprise it was not encrypted and Inspector found the pattern. The only thing left to do is make the change and try again. To save you time, there are two places on the disk where this routine is found, so search the whole disk before you try booting the disk again.

Now you you have Boulder Dash I liberated from the copy protection. Boulder Dash II is just as easy. Make the first sector edit just like Boulder Dash I, (change track 0 sector 1 byte \$FF to a \$02, and track 1 sector 5C byte \$00 to a \$60). Next search the disk for \$A9 15 48. You will find two places that need to be changed. This time we will put at \$1561 a jump to \$15A2. Check at \$1561 and you will find almost the same routine.

I made the following changes on my disks:

	Track	Sector	Byte	From	To
BD I	\$1	\$7	\$FF	\$0F	\$02
	\$1	\$F	\$00	\$4C	\$60
	\$D	\$4	\$EB	\$A9	\$4C
	\$D	\$4	\$EC	\$14	\$2C
	\$D	\$4	\$ED	\$48	\$14
	\$E	\$C	\$EB	\$A9	\$4C
	\$E	\$C	\$EC	\$14	\$2C
	\$E	\$C	\$ED	\$48	\$14
BD II	\$1	\$7	\$FF	\$0F	\$02
	\$1	\$F	\$00	\$4C	\$60
	\$D	\$2	\$61	\$A9	\$4C
	\$D	\$2	\$62	\$15	\$A2
	\$D	\$2	\$63	\$48	\$15
	\$E	\$A	\$61	\$A9	\$4C
	\$E	\$A	\$62	\$15	\$A2
	\$E	\$A	\$63	\$48	\$15

An update to the softkey for..

Hard Hat Mack

by Brian Troha

Requirements:

Hard Hat Mack softkey
(Issue No. 5 or Book of Softkeys I)

After softkeying Hard Hat Mack (see "Boot Code Tracing Hard Hat Mack," COMPUTIST No. 5), I started looking into the program code. There are a total of six calls to the protection routine. The first is taken when the game is started, while the other five are taken during the run of the demo. After each JSR \$43D4 you will find a STA \$xxxx. This is done to scramble the game if you put a RTS at \$43D4. Otherwise the program pulls the return address from the stack and adds three to it (thus "jumping" over the STA command), pushes it back on, then pushes two more values onto the stack so it returns to \$500. One possible way to fix this is to pull the last value off the stack, add three to it and return. The following code will do this:

```

4D34:68    PLA        pull it off
4D35:18    CLC        ready for add
4D36:69 03  ADC #03    add three to it
4D38:48    PHA        put it back
4D39:60    RTS        done, return

```

The other thing we could do is eliminate the jumps to the code altogether. After the first call and STA instruction there is valid program code, while after all the others there is an RTS. To eliminate the jumps we would do the following:

```

864: EA EA EA EA EA EA
1114:60
1212:60
12D8:60
5F32:60
70A8:60

```

Now there are no jumps to the protection so we can overwrite all the memory from \$3000 to \$3FFF (sixteen free pages). After a little work you would find the memory from \$2A00 to \$2FFF, \$2000 to \$21FF, and \$2310 to \$23FF are all free. If you put your move routine at \$2310 you can save twenty-four pages of memory. Follow the procedure in the original article up to step 13, then:

14) Do a few memory moves: restore page \$08 to its original location, then pack high memory

into the free area mentioned above.

```

800<3400.34FFM
2A00<7F00.94FFM
2000<7D00.7EFFM

```

15) Type in a memory move routine to move it all back.

```

2310:A0 00 B9 00 2A 99 00 7F
2318:C8 D0 F7 EE 14 23 EE 17
2320:23 EA 17 23 E0 95 D0 E8
2328:A0 00 B9 00 20 99 00 7D
2330:B9 00 21 99 00 7E C8 D0
2338:F1 60

```

16) Fix the start of the game and set up for some APTs.

```

800:20 23 10 20 04 22 4C 2D
808:08 00 00 00

```

17) Change the routine that sets up the reset vector to point to the monitor.

```

2213:69
2218:FF

```

18) Save the file to disk.

```

BSAVE HARD HAT
MACK,A$800,L$7600

```

Now to do any of the following APTs just hit Reset and you will be in the monitor. You can use one or all of the APTs to practice the game or to help master one single level. After doing an APT restart the program with 82DG then play the game. For the entire game:

1. Number of men to start (do not exceed \$80)

```
A72:xx
```

2. Immune to OSHA and Vandals

```
5C40:60
```

3. No Bonus countdown

```
581D:60
```

For first level:

1. Can't fall down holes

```
95C:EA EA EA
```

2. No rivets from machine

```
1660:60
```

For second and third levels:

Immune to crunchers (not including "vice jaws" on second level)

```
5A2A:60
```

The Other

by Dick Meikle

Tom Snyder Productions, Inc.
123 Mt. Auburn St.
Cambridge, MA 01238

Requirements:

- 48K Apple II Plus and up
- Super IOB 1.5 or sector editor
- One blank disk
- Disk search utility (optional)

You are the national leader of one of the two great powers in the world. You must manage your economy for maximum prosperity, compete with the other world power for scarce resources, maintain the security of your nation, and build a bridge to world peace. You soon discover that these goals, difficult to achieve individually, are nearly unattainable when taken together.

This is the scenario for **The Other Side**, a global conflict resolutions simulation created by Tom Snyder Productions. Designed primarily for use in a classroom environment, this program presents its participants with the challenge of managing events in a simulated "world" where negotiation and overcoming conflict are the only sure way to "win" the game. The "world leaders" may approach the game from either a competitive or collaborative strategy. The simulation may run on a single computer or, ideally, on two computers in separate locations so that the players cannot be certain what the "other side" is planning. All communications between nations take place over a "Hot-line" link, via modem or inexpensive cable (available from Tom Snyder Productions). Currently one of the hottest pieces of educational software, **The Other Side** was once featured on NBC's Today program which reported a simulation carried out by high school classes in America and Europe linked by modem.

The Protection

When a disk is used around children, even 15 to 18 years old, it is not a very good idea to use an original disk. Although **The Other Side** comes with a backup, Murphy's Law being what it is, I prefer using an easily backed up and unprotected copy.

The first copy program I try when making a backup is COPYA. Sometimes, even in this copy protected world, this works. Not this time. Using COPYA on this program gives a "cannot read error". With the first step out of the way I turned to EDD III. I noticed when copying track \$11 that the track was mostly \$FFs. This is a good indication of a nibble count. I tried the manual nibble count on track \$11 but this did not yield a very reliable copy. It would not boot every time. Armed with my Apple IIc and Wildcard I decided to bypass the nibble count.

The Wildcard gives me the ability to stop the booting process during the nibble count. I booted the original after write-protecting it (I take no chances). If there is one thing you will read time and again in COMPUTIST it is to listen to your drive when it is booting, I stopped the boot with the Wildcard when I heard the drive head move to track \$11. This tells where in memory the nibble count is taking place.

Looking at the code in memory, it appeared that the nibble count routine started at \$2400. The first three bytes were \$4E 03 24. I got out my disk search utility and searched for these three bytes. I found them on track \$10 sector \$0 starting at byte \$00, but the rest of the sector seemed to be garbage. I rebooted the original to take another look at the code. It was then that I noticed that the code at \$2400 was self-modifying. The apparent garbage alters itself to make executable code. Starting at \$2400 each instruction alters the next until it reaches \$2412. At \$2419 each byte from \$241F through \$24FF are (ROR) ROTated one bit Right and at \$241F the same thing happens to memory locations \$2500 through \$25FF.

```

                (after decoding)
2400- LSR $2403  decode next step
2403- ROR $2406  " " "
2406- ROR $2409  " " "
2409- ROR $2419  " elsewhere
240C- ROR $240F  " next step
240F- ROR $2415  " elsewhere
2412- TXA       save current X
2413- LDX #51F
2415- ROR $241D  decode a step
2418- CLC       set carry bit to 0
2419- ROR $2400,X decode
241C- INX       many
241D- BNE $2419 bytes
241F- ROR $2500,X decode
2422- INX       more
2423- BNE $241F bytes
2425- TAX       restore X
2426- LDA $02   push locations 0-2
2428- PHA       on stack
2429- LDA $01
242B- PHA
242C- LDA $00
242E- PHA
242F- STX $02   save X
2431- JSR $253B set up zero page
2434- LDY #520  <<<insert JUMP<<<
2436- LDA #522
2438- JSR $24C4 start n-count
243B- LDA $C08C,X read byte
243E- BPL $243B
2440- PHA       waste
2441- PLA       time
2442- CMP #5D5  is it a $D5?
2444- BNE $2438 start again if not
    
```

The Softkey

The actual nibble count starts at \$243B. If the count is not correct the program will branch to \$24B6 and crash. If no error occurs the boot continues at \$24A0. All that is needed to do is put a JMP \$24A0 at \$2434 so the boot will continue. This does not seem too difficult. Just change the bytes starting at \$2433 to 4C A0 24

Side

(JMP \$24A0). But, remember, the code from track \$10 sector \$00 is altered at \$2419. The bytes will be RORed. So bytes \$33-\$36 on track \$10 sector \$0 must be changed from \$4B 40 41 52 to \$4A 99 40 48. Note that I changed four bytes rather than three. This is because \$4A (01001010 in binary) and \$4B (01001011) both rotated right will give us a \$25 at \$2433. However, when \$4B is rotated it causes the rightmost bit to be moved into the "carry" of the status register. Then when the byte \$99 is rotated, the bit placed in the carry from the last rotation will be moved into the byte. That would give a \$CC (11001100 bin.) A0 24 (CPY \$24A0) instead of \$4C (01001100 bin.) A0 24 (JMP \$24A0) at \$2434. This is the sector edit made by the controller at the end of this article. The controller formats the copy and copies tracks \$0-\$10, makes the sector edit on track \$10, then copies tracks \$12-\$22. The JUMP cannot be placed earlier because the stack and zero page must be set before the jump can be executed.

The sector edits made by the controller are:

Track	Sector	Byte	From	To
\$10	\$0	\$33	\$4B	\$4A
		\$34	\$40	\$99
		\$35	\$41	\$40
		\$36	\$52	\$48

Step by Step

- 1) Type in the controller below.
- 2) Install the controller into Super IOB 1.5.
- 3) RUN Super IOB and answer Yes to the Formatting option.

That's all there is to it. You have just deprotected The Other Side.

On each disk the sector arrangement may be different. If this does not work on your copy you will need a disk search utility. Make a copy with any copier that ignores disk errors (such as Locksmith Fast Copy, Disk Muncher, etc...) and search for the hex bytes \$4B 40 41 52. Change them to \$4A 99 40 48. Write the sector back to the copy. If you want to make the disk

completely COPYAable, use a nibble copier to copy track \$11 of a DOS 3.3 disk to this disk (but don't use it to store anything).

controller

```

1000 REM OTHER SIDE CONTROLLER
1010 TK=0:LT=17:ST=15:LS=15:CD=WR:FAST=1
1020 GOSUB 490:GOSUB 610
1025 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 IF PEEK(TRK)=17 THEN TK=18:LT=35:GOTO 1020
    
```

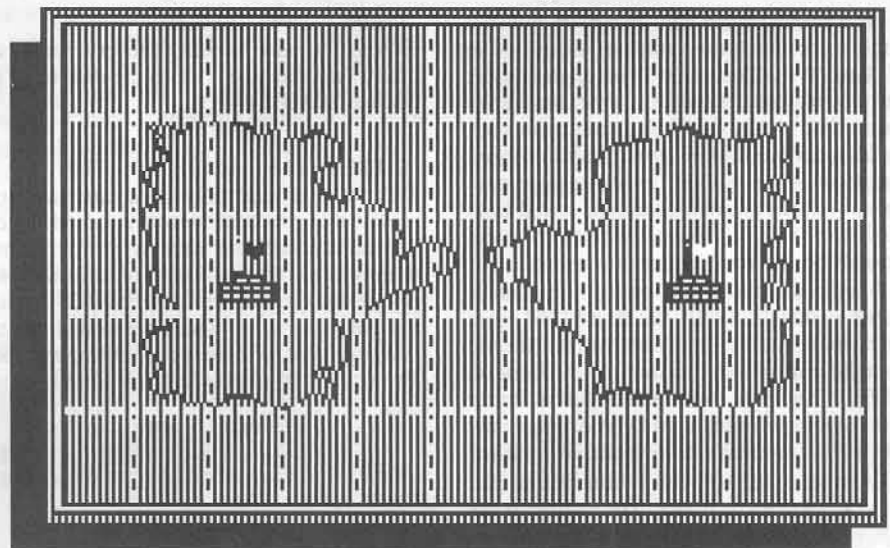
```

1060 HOME:PRINT "COPY^ DONE":END
1100 DATA 4^ CHANGES
1110 DATA 16,0,51,74
1120 DATA 16,0,52,153
1130 DATA 16,0,53,64
1140 DATA 16,0,54,72
    
```

controller checksums

1000	- \$356B	1060	- \$4D61
1010	- \$3873	1100	- \$0184
1020	- \$7BFA	1110	- \$3BC7
1025	- \$7025	1120	- \$083C
1030	- \$03C8	1130	- \$5A45
1040	- \$89FF	1140	- \$4EBB
1050	- \$26F8		

Prepare for a close-up of area 21



Looking Into

by Stephen L. Favor

Editor's note: We were going to print this article in an earlier issue, but when we recieved a real live softkey for Flight Simulator II (COMPUTIST No. 36), we put this one (which isn't a full softkey) on the back burner. We feel it can provide some useful information anyway, so here it is.

In the Gray House, the leader of serious Apple users takes the stand. "Ladies and gentlemen, friends and foes, I am pleased to announce I have just enacted legislation that outlaws Flight Simulator II. We begin hacking immediately!"

The Iron Curtain

I must compliment the writers of Flight Simulator's (FS-II) DOS. It is not a modified version of 3.3 but totally new and foreign. It forms a very nice (for SubLogic) or disgusting (to me) wall between FS-II and its users. Here is a short tutorial of what this monster does and how it does it.

FS-II doesn't store data as sectors but as a single track. Each time routines are called in this Read/Write Track (RWT) program, it seeks the proper track, looks for a header in the format \$92 94 XX (XX being unique for each track), and reads \$1A00 bytes of 6&2 encoded data to a buffer (\$2600) in hi-res page one. Then, \$1000 bytes are decoded and stored from \$2600 to \$35FF. What happens from that point depends on the routine that was called.

For the curious mind, here's how data is stored on the disk itself. The first \$1500 bytes are written in a \$15 by \$100 matrix. See the example below and disassemble the code at \$23D9 once you have RWT for a better understanding. Then, \$500 bytes of sequential data follow, and that's it. How simple can it be?

Example 1: \$15 by \$100 information matrix. All values are given in hex. The top row is the high byte and the first column is the low byte of a location within the RWT buffer where information is stored. These two numbers index the sequential location on the disk of the byte that will be stored in thier address. e.g. The location \$3902 will receive byte number \$3E

from the disk. Or byte \$14C2 from the disk will be stored at \$27FD.

xx	----- page -----						
	\$26xx	27xx	28xx	...	38xx	39xx	3Axx
00	0000	0001	0002	...	0012	0013	0014
01	0015	0016	0017	...	0027	0028	0029
02	002A	002B	002C	...	003C	003E	003F
FD	14C1	14C2	14C3	...	14D3	14D4	14D5
FE	14D6	14D7	14D8	...	14E8	14E9	14EA
FF	14EB	14EC	14ED	...	14FD	14FE	14FF

There are seven subroutines that can be called to send data to and from the disk. They all are entered by a jump table (series of JuMPs) beginning at \$1EAD. Here's a list of them and what they do.

\$1EAD- JMP \$203A ;"RDQ"
This routine reads a Quarter of the \$1000 decoded bytes from the buffer to the address stored at \$1E03,4. The quarter number (\$1E01) is from \$00 to \$87. The first quarter of track one being \$00 and \$87 the last of track \$22. More on deciphering this number later.

\$1EB0- JMP \$2015 ;"WRQ"
Exactly the same as RDQ except it writes the quarter number at \$1E01 to the disk from the address at \$1E03,4.

\$1EB3- JMP \$20AF ;"RDT"
This jump is used only once, and the write routine that goes with it is never called. It goes beyond the decoded data in the buffer and decipheres a \$150 byte "tail" of its own. There is one tail on each of tracks \$01-22 totaling \$2CA0 bytes, all of which are loaded into the language card, if there is one, during the boot. The number of a tail is its track number minus one.

\$1EB6- JMP \$2122 ;"WRT"
Never called. Forget it.

\$1EB9- JMP \$238A ;CRASH
It's code like this that upsets me somewhat. If I'm correct, this one will simply replace the data on every other track with \$FDs starting with track \$02.

\$1EBC- JMP \$2000 ;drive off
\$1EC4- ;"LDRWT"
Home disk arm and load RWT to \$2000.

All that is left now is the IOB table beginning at \$1E00. Below is a list of what each byte represents.

\$1E00: Drive no.
\$1E01: Quarter: \$00-87, \$00 being the first quarter of track \$01 and \$87 the last of track \$22. Also, Tail no. from \$00 to \$21. Add one to obtain track no.
\$1E02: ?
\$1E03: Low byte of destination address
\$1E04: High byte of destination address
\$1E05: ?
\$1E06: ?
\$1E07: 0=48K, 1=64K
\$1E08: Slot no.
\$1E09-1E0C: Used by the track seeking routine.
\$1E0D: Third mark of header. Each track has a different header. All calls set this byte to the correct value according to the contents of \$1E01.
\$1E0E: 0=no error. The program will lock up at 1000 feet if this doesn't equal zero.
\$1E0F: ?

Beyond The Iron Curtain

Let's put this information to use now, and put FS-II on a normal (3.3 format) disk.

- 1) Boot FS-II.
- 2) The third time you hear the disk arm recalibrate (the grinding noise), press Reset. You don't need a modified ROM or anything.
- 3) Boot a slave disk (don't use Control/Open-Apple/Reset!).
- 4) BSAVE FS-II.RWT, AS\$2000, LS\$600.
- 5) Type in the Super IOB controller (you can ignore what comes after the REMs) and run it to move FS-II to a blank disk.

Now you have everything you need to give you a splitting head-ache!

About That Weird Boot

The Boot ROM loads a \$100 byte program and jumps to \$801 as usual. Then, a three page block is read to \$1D00 and execution transfers

Flight Simulator II

there. It is here that the unique sound of this boot appears. The disk arm homes for the second time and loads RWT to \$2000, and quarters \$22 through \$24 (track \$09, sector \$08 to track \$0A, sector \$03 on the deprotected disk) are stored to \$A7E0.

After a pair of jumps, execution ends up at the last stage of the boot, \$AA4D. The main routines this final stage uses are:

\$1EC4: Home disk arm (for the third time) and reload RWT to \$2000.
 \$AAA5: Read quarters \$00-06 to \$0200, and quarters \$07-19 to \$6000.
 \$ABB5: Read quarters \$1A-21 to \$2000.
 \$AC9B: If 64K, load tails \$00-21 to \$D000. \$8B2=01.
 \$A71C: Loaded by \$AAA5. Turn off disk drive and copy hi-res page two over hi-res page one.
 \$AD2B: Replace zero page.
 \$ACD0: If \$8B2 isn't zero, modify for 64K version.

You will also notice an abundance of the command "BCS \$AA70." If any of these branches occurs, the ominous error trap (JSR \$1F89) that blanks the screen and displays several numbers at the top left of the CRT will be invoked. The rest of the routines display the opening messages and don't seem to do any error checking.

Finding A Quarter

Below is a short assembly routine that will set up the track and sector for a given quarter. Call it and read the next four sectors, and you'll have the quarter. The formulas used are: **track = int (quarter + 4) / 4** and **sector = (quarter * 4 + 3) AND \$0F** (Note: the program performs the "AND" before adding three in case the carry flag was set when the first shift was performed.)

```
LDA QUARTER ;(LDA $1E01)
CLC ;Adjust so quarter $00
ADC #$04 ; will be Trk $01, Sct $00
LSR ;Divide
LSR ; by four
STA TRACK ; to get track no.
LDA QUARTER ;(LDA $1E01 again)
ASL ;Multiply
ASL ; by four
AND #100001100 ; and use bits 2 and 3
CLC
```

```
ADC #$03 ;Plus $03
STA SECTOR ; for sector number
```

Final Comments

So, if I know so much, why don't I finish the deprotection scheme? Well, I may, but I don't have the assurance that I will have the time or the patience. I still want to see this one fall, and I'm not greedy for fame. So all of you out there who have some extra time (a lot of it!)... Here are some tips that might help you do it.

Probably the most difficult problem will be finding a place for DOS. There are only three blocks of memory unused that I can find. \$1E00-1FFF is free except for the DOS jumps from \$1EAD to \$1EC6. Pointing these jumps to code that imitates RWT would work nicely because all calls are made via these addresses. Remember, \$1E0E must equal zero at all times.

The second pocket is where RWT resides in the language card (64K version). It starts at \$D3D0 (\$D000 to \$D3CF may be free also) with a series of jumps presumably (you better check before you mess with them) unimportant once RWT is unhooked. Last, bank one of the language card can be used if you modify or delete the course plotter. It's the only place I found any reference to bank one, but I don't know to what extent it is used.

Above all, remember that the more memory you change, the greater the chance you have of being found out. Be sneaky and try to put your new code in places Big Bother would never think of looking.

controller

```
60 HIMEM: 7167 : GOTO 10010 : REM HIMEM JUST
BELOW RWT. BUFFER WILL START AT
$4000 LEAVING ROOM FOR FIVE TRACKS.
1000 A$ = "INSERT^ DISK^ W/FS-II.RWT" : GOSUB
470 : PRINT CHR$(13) CHR$(4) "BLOAD^
FS-II.RWT" : REM LOAD RWT. IT WILL USE
ALL OF HI-RES PAGE ONE.
1010 A$ = "INSERT^ FS-II" : GOSUB 470
1020 POKE 7424,174 : REM 1D00- "LDX" ($1E08
) . LOAD X-REG WITH SLOT NUMBER
INSTEAD OF STORING SLOT NUMBER.
1030 POKE 7438,0 : POKE 7443,64 : REM 1D0D- (LDA
) "#$00" AND 1D12- (LDA) "#$40" . THE
COMMAND IMMEDIATELY FOLLOWING EACH OF
THESE COMMANDS STORES THE NUMBERS AS THE
TARGET BUFFER AT 1E .4.
```

```
1040 DATA ^ 96,162,0,181,0,157,0,28,202
,208,248,32,0,29,162,0,189,0,28,149
,0,202,208,248,96
1050 FOR I=7469 TO 7493 : READ X : POKE I,X : NEXT
: REM SHORT PROGRAM BEGINNING AT 1D2E
TO SAVE AND RESTORE THE ZERO PAGE,
RESPECTIVELY, SO CALLING 1D00 WON'T
CLOBBER DOS. CALL THIS PROGRAM FOR
ALL READS FROM THE FS-II DIS
1060 T=1
1070 POKE 7448,T*4-4 : POKE 7466,(T+5
)*4-4 : IF T+5 > 34 THEN POKE 7466
,34*4-4 : REM 7448 = STARTING
QUARTER, 7466 = ENDING QUARTER. SET
TO READ 20 QUARTERS, OR FIVE
TRACKS.
1080 CALL 7470 : REM 1D2E
1090 VL=0 : CD=RD : GOSUB 490 : POKE BUF,64 :
REM WRITE IT TO NORMAL DOS
1100 FOR TK=T TO T+4 : IF T+4 > 34 THEN FOR TK
=T TO 34
1110 FOR ST=0 TO 15 : GOSUB 100 : GOSUB 430 : NEXT
: NEXT
1120 T=T+5 : IF T < 35 THEN 1070 : REM CHECK
FOR LAST TRACK
2000 POKE 7481,175 : POKE 7482,32 : REM 1D38-
(JSR) "$20AF" . MODIFY TO GET TAILS.
2010 POKE 49385,0 : POKE 49386,0 : REM DRIVE
#1 ON. BSAVE IN LINE 2050 WILL
TURN IT OFF.
2020 POKE 7681,0 : POKE 7683,0 : POKE 7684,64
: REM START WITH TAIL #0. BUFFER AT
$4000.
2030 FOR T=1 TO 34 : CALL 7470 : NEXT : REM READ
34 TAILS (ALL OF THEM).
2040 A$ = "INSERT^ SLAVE^ DISK" : GOSUB 470
2050 PRINT CHR$(13) CHR$(4) "BSAVE^
FS-II.D000,A$4000,L$2CA0" : REM
SAVE THE TAILS.
2060 END
```

controller checksums

60	- \$7FB5	1100	- \$75A9
1000	- \$9AE2	1110	- \$5F0B
1010	- \$C06F	1120	- \$2A92
1020	- \$18BA	2000	- \$05A1
1030	- \$9DEE	2010	- \$C97D
1040	- \$FB07	2020	- \$CB93
1050	- \$62D5	2030	- \$EEB5
1060	- \$3242	2040	- \$47E0
1070	- \$D313	2050	- \$4355
1080	- \$D60E	2060	- \$8DB3
1090	- \$E81A		

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/uni-disk 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.

UTILICO MICROWARE

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

WARNING: EDD is sold for making archival copies only.



BACKUP YOUR SOFTWARE WITH LOCKSMITH 6.0™.

Locksmith, the controversial copy program that took the Apple world by storm in 1981, has evolved from a powerful bit-copy programmed into a complete disk utility system, allowing the Apple user to recover crashed disks, restore accidentally deleted files, and perform hardware diagnostics on the disk drive and memory boards. The NEW Locksmith version 6.0 is now available and includes an advanced disk recovery utility, a framing-bit analyzer, an automatic boot tracer, a sector editor, many file utilities, and of course, the most powerful bit-copy program available. A fast disk backup utility copies disks in eight seconds flat. Improvements to Locksmith Programming Language have made it more powerful and easier to use for you to write your own backup and repair procedures. Includes a library disk which contains automatic procedures to copy hundreds of Apple programs.

Locksmith requires no additional hardware, but will use any additional RAM memory that it finds, including RAM boards from Applied Engineering and Checkmate Technology.

Don't get caught with your hands tied. Order Locksmith 6.0 today.

Does copy protection have your hands tied?



NEW LOW PRICE **\$79.95**

Registered Locksmith 5.0 owners may upgrade to version 6.0 for **\$29.95**. Available from your computer dealer or directly from:



Alpha Logic Business Systems, Inc.
4119 North Union Road
Woodstock, IL 60098
(815) 568-5166



© Alpha Logic Business Systems, Inc. 1985
Locksmith and Locksmith PC are registered trademarks of Alpha Logic Business Systems, Inc.

40¢ per disk?

And that **INCLUDES** the Tyvek sleeve, too?

SS/DD

5 1/4"

for use with:

- Apple
- Commodore
- Atari
- Epson
- TRS-80

50
bulk disks
for only
\$20

- Sold only in sets of 50.
- 100% guaranteed.
- reinforced hubs.
- Tyvek sleeves & Write-protect tabs included.

Send your order to:
Disk Offer
SoftKey Publishing
PO Box 110846-T
Tacoma, WA 98411
Enclose \$4 shipping & handling for each set.

Please send me _____ sets (50 disks per set) of 5 1/4" SS/DD diskettes. I have enclosed \$20 PLUS \$4 shipping & handling for each set and made checks / money orders payable to SoftKey Publishing.

Name _____ ID# _____

Address _____

City _____ State _____ Zip _____

Country _____ Phone _____

Exp. _____

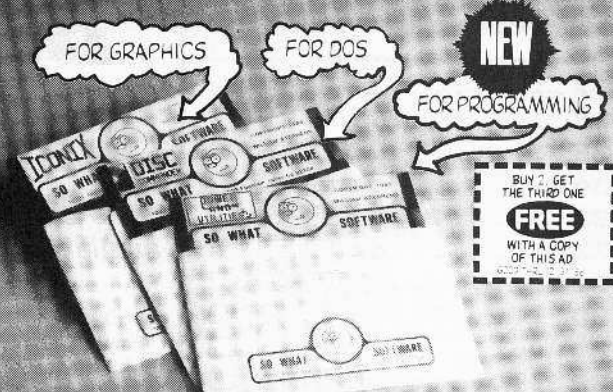
Signature _____

Domestic orders only, please include street address for UPS delivery.

Washington state orders add 7.6% sales tax.

Offer good while supplies last. Expires December 31, 1986.

So What Software Presents MORE BANG FOR THE BUCK



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 ~~5.00~~ SHIP & HANDLE FREE FOREIGN-ADD ~~2.00~~ FREE CA ~~0.50~~

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

Apple is a registered trademark of Apple Computer, Inc.

APPLE®

PROGRAMMERS

P U B L I S H W I T H

UPTIME

THE DISK MONTHLY

Gain national prominence with America's leading monthly disk publication. (We were the first to publish Bill Travers!) If you are either an Apple II or Macintosh programmer... you won't find a better way to publish your work than through UpTime, The Disk Monthly. We seek submissions for both the Apple II series and the Macintosh computers.

Call: Bill Kelly at 1-800-437-0033 today. Or, write to Bill at UpTime, 174 Bellevue Avenue, Newport, Rhode Island 02840.

New Version, Improved and Easier to use!

Graduate... to the Senior PROM!

version 3.0

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 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

Issue	Mag \$4.75	Disk \$9.95	Both \$12.95
38	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
37	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
36	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
35	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
34	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
33	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
32	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
31	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
30	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
29	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
28	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
27	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
26	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
25	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
24	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
23	NA	<input type="checkbox"/>	NA
22	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
21	NA	<input type="checkbox"/>	NA
20	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
19	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
18	NA	<input type="checkbox"/>	NA
☆ 17	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
16	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
★ Book of Softkeys Vol 3			
15	NA	<input type="checkbox"/>	NA
14	NA	<input type="checkbox"/>	NA
13	NA	<input type="checkbox"/>	NA
12	NA	<input type="checkbox"/>	NA
11	NA	<input type="checkbox"/>	NA
★ Book of Softkeys Vol 2 \$19.95			
10	NA	<input type="checkbox"/>	NA
9	NA	<input type="checkbox"/>	NA
8	NA	<input type="checkbox"/>	NA
☆ 7	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6	NA	<input type="checkbox"/>	NA
★ Book of Softkeys Vol 1 \$14.95			
4	NA	<input type="checkbox"/>	NA
3	NA	<input type="checkbox"/>	NA
Core 2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	NA	<input type="checkbox"/>	NA
1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Core 1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Core 3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Computing 3	<input type="checkbox"/>	NA	NA
Best of Hardcore Computing	NA	<input type="checkbox"/>	NA
★ Core Special \$10.00 (All three CORE magazines)			

BACK ISSUES and LIBRARY DISKS of

COMPUTIST are still available. Library disks are available for ALL issues, even for issues that are no longer available (marked NA).

LIBRARY DISKS are perfect companions for COMPUTIST

Documentation for each Library Disk is in the corresponding issue.

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

Name _____ ID# _____
 Address _____
 City _____ State _____ Zip _____
 Country _____ Phone _____
 Signature _____ CP38

HOLIDAY SAVINGS!

- Yes, I want to take advantage of the special back issue prices indicated on the Holiday savings page. I understand that the sale price is \$10 for 3 issues. (Foreign orders \$20/3 issues)
- I want to take advantage of the special library disk prices. I understand the holiday price is 5 disks for \$30. (Foreign orders add \$5 shipping & handling). I understand that I will receive a free color coded case with each set of 5 library disks ordered.
- I want to take advantage of the library disk and magazine combination prices. I understand the holiday price is \$9.95 per set, \$3 off the regular price. (Foreign shipping add \$3).
- My order is over \$100.00. I would like to take advantage of your special offer and receive a
 - free Core Special or a
 - free COMPUTIST T-shirt. MENS SIZE: XL L M S
(T-shirts will be supplied while quantities last. In the event your size is not available, a Core Special will be supplied.)

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

Canada and Mexico back-issue and library disk rates are identical to U.S. First Class unless otherwise specified.

Other Foreign: Price for each magazine includes shipping.

1 - 2 copies	3 to 4 copies	5 or more copies
\$14.25 each	\$13.25 each	12.25 each

Other Foreign disk rates are \$11.94 each (includes shipping). Special "Both" disk and magazine combinations shown do NOT apply to Foreign orders. However, Foreign Subscribers can take advantage of our NEW Combination Library Disk and Magazine subscription rates.

Book Of Softkeys: Volume 1: \$17.95 Volume 2: \$22.95

US funds drawn on US banks. All foreign orders sent AIR RATES.

Special "Both" disk & magazine combination orders apply to one issue and its corresponding disk. Prices shown are for U.S., Canada, and Mexico only.
 Some disks apply to more than one issue and are shown as taller boxes.
 ★ Book Of Softkeys volumes do not come with disks.
 ☆ We have a limited supply of these issues.

Description of Available Back Issues

37 *Softkeys* | Under Fire | Pegasus II | Take 1 (revisited) | Flight Simulator II v1.05 (part 2) | *Readers' Softkeys* | Magic Slate | Alter Ego | Rendezvous | Quicken | Story Tree | Assembly Language Tutor | Avalon Hill games | Dark Crystal | *Features* | Playing Karateka on a //c | Track Finder | Silk to Dif | *Core* | Breaking In: tips for beginners | Copy II Plus 6.0: a review | The DOS Alterer |

36 *Softkeys* | Flight Simulator II v 1.05 | AutoDuel | *Readers' Softkeys* | Critical Reading | Troll's Tale | Robot War | General Manager | Plasmania | Telarium Software | Kidwriter v1.0 | Color Me | *Features* | ScreenWriter meets Flashcard | The Bus Monitor | Mousepaint for non-Apples | *Core* | The Bard's Dressing Room | *Advanced Playing Techniques* | Championship Lode Runner |

35 *Softkeys* | Hi-res Cribbage | Olympic Decathlon | Revisiting F-15 Strike Eagle | Masquerade | The Hobbit | *Readers' Softkeys* | Pooyan | The Perfect Score | Alice in Wonderland | The Money Manager | Good Thinking | Rescue Raiders | *Feature* | Putting a New F8 on Your Language Card | *Core* | Exploring ProDOS by installing a CPS Clock Driver |

34 *Softkeys* | Crisis Mountain | Terripin Logo | Apple Logo II | Fishies 1.0 | SpellWorks | Gumball | *Readers' Softkeys* | Rescue at Rigel | Crazy 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 | Cracking 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 |

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's Softkeys* | The Report Card V1.1 | Kidwriter | *Feature* | Apple II 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 |

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 |

1 *Softkeys* | Data Reporter | Multiplan | Zork | *Features* | PARS 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,
consult our
'Holiday Specials Ad'*

The Super IOB Collection

- What could possibly be better than receiving COMPUTIST every month, typing in the Super IOB controllers and deprotecting your favorite software? How about having **all the controllers ever printed in COMPUTIST** at your fingertips? With *The Super IOB Collection* Volumes I & II, you have just that and more.
- Each volume (supplied on a DOS 3.3 disk) contains at least 60 Super IOB controllers including the standard, swap, newswap and fast controllers. In addition, each disk has the Csave program from COMPUTIST No. 13. But wait! You also get version 1.5 of Super IOB and a menu hello program that lists the available controllers and, when you select one, automatically installs it into Super IOB and RUNs the resulting program.*
- Several of the controllers deprotect the software completely with no further steps. This means that some programs are only minutes away from deprotection (with virtually no typing).
- The issue of COMPUTIST in which each controller appeared is indicated in case further steps are required to deprotect a particular program.†

Volume 1

Volume 1 of the Super IOB collection covers all the controllers appearing in COMPUTIST No. 9 through No. 26. In addition, the newswap and fast controllers from COMPUTIST No. 32 are included. The following 60 controllers are on volume 1:

Advanced Blackjack, Alphabet Zoo, Arcade Machine, Archon II, Archon, Artsci Software, Bank Street Writer, Barrons SAT, Beyond Castle Wolfenstein, BSW //c Loader, Castle Wolfenstein, Computer Preparation: SAT, Dazzle Draw, DB Master 4 Plus, Death in the Carribean, Dino Eggs, DLM Software, Electronic Arts, F-15 Strike Eagle, Fast Controller, Fathoms 40, Financial Cookbook, Gessler Software, Grandma's House, The Heist, In Search of the Most Amazing Thing, Instant Recall, Kidwriter, Lions Share, Lode Runner, Mastertype, Match Maker, Miner 2049er, Minit Man, Mufplot, Newsroom, Newswap controller, Penguin Software, Print Shop Graphic Library, Print Shop, Rendezvous with Rama, Rockys' Boots, Sargon III, Sea Dragon, Shiela, Skyfox, Snooper Troops, Standard controller, Stoneware Software, Summer Games, Super Controller, Super Zaxxon, Swap Controller, TAC, Ultima III, Word Challenge, Xyphus, Zaxxon

Volume 2

Volume 2 of the Super IOB collection covers all the controllers appearing in COMPUTIST No. 27 through No. 38. The following 65 controllers are on volume 2:

Alice in Wonderland, Alphabetic Keyboarding, Alternate Reality, Autoduel, Checkers, Chipwits, Color Me, Conan.data, Conan.prog, CopyDOS, Crisis Mountain, Disk Director, Dragonworld, Early Games, Easy as ABC, F-15 Strike Eagle, Fantavision, Fast controller, Fishies, Flight Simulator, Halley Project, Hartley Software (a), Hartley Software (b), Jenny of the Prairie, Jingle Disk, Kidwriter, Kracking Vol II, Lode Runner, LOGO II (a), LOGO II (b), Masquerade, Mastering the SAT, Microtype: The Wonderful World of Paws, Microzines 1, Microzines 2-5, Miner 2049er, Mist & View to a Kill, Murder on the Zinderneuf, Music Construction Set, Newswap controller, Olympic Decathlon, Other Side, Phi Beta Filer, Pitstop II, Print Shop Companion, RDOS, Robot War, Spy vs Spy, Standard controller, Sundog V2, Swap controller, Sword of Kadash, Synergistic Software, Tawala's last Redoubt, Terripin Logo, Threshold, Time is Money, Time Zone, Tink! Tonk!, Troll's Tale, Ultima IV, Wilderness, Word Attack & Classmate, World's Greatest Baseball, World's Greatest Football

To Order: Send \$9.95 for each volume or \$19.95 for a complete package that includes: both disks, a reprint of "Disk Inspection and the use of Super IOB" and COMPUTIST No. 32. U.S. funds drawn on U.S. banks. Foreign orders (other than Canada or Mexico) add 20% shipping. Washington state residents add 7.8% sales tax. Mail orders to: Super IOB Collection; POB 110846; Tacoma, WA 98411

*Requires at least 64K of memory.

†Although some controllers will completely deprotect the program they were designed for, some will not and therefore require their corresponding issue of COMPUTIST to complete the deprotection procedure.