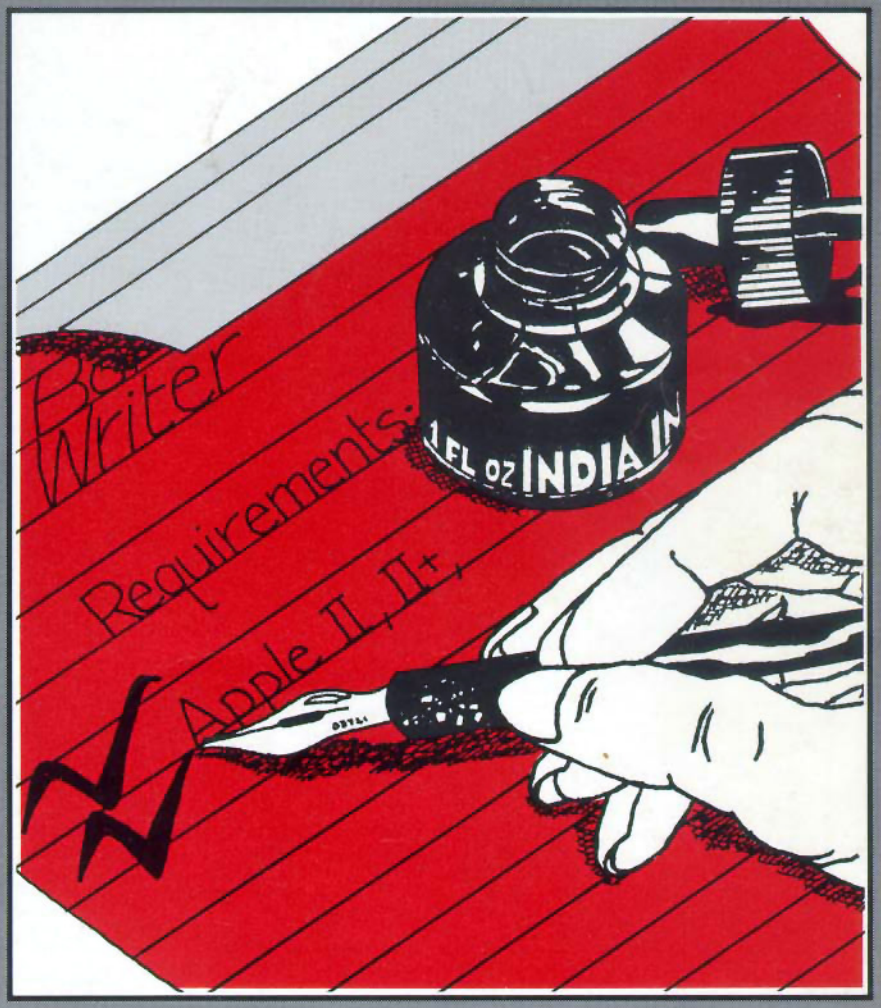
HOW TO BACKUP, UNLOCK, OR MODIFY COPY-PROTECTED SOFTWARE

Hardcore COMPUTIST

Issue No. 10 $2.50



(CA5/340) 7 issues left

Hardcore COMPUTIST
P.O. Box 44549
Tacoma, WA 98444

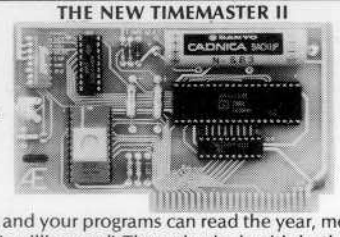
BULK RATE
U.S. Postage
PAID
Tacoma, WA
Permit No. 269

APPLIED ENGINEERING IS 100% APPLE

That's Why We're So Good At It!

THE NEW TIMEMASTER II

Automatically date stamps files with PRO-DOS



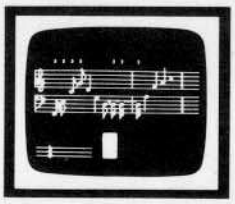
NEW 1984 DESIGN
An official PRO-DOS Clock

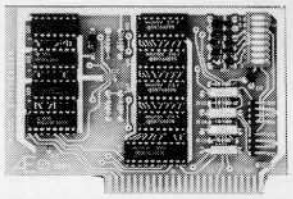
- Just plug it in and your programs can read the year, month, date, day, and time to 1 millisecond! The only clock with both year and ms.
- A rechargeable NiCad battery will keep the TIMEMASTER II running for over ten years.
- Powerful 2K ROM driver — No clock could be easier to use.
- Full emulation of most other clocks, including Thunderclock and Appleclock (but you'll like the TIMEMASTER II mode better). We emulate other clocks by merely dropping off features. We can emulate them but they can't emulate us.
- Basic, Machine Code, CP/M and Pascal software on 2 disks!
- Eight software controlled interrupts so you can execute two programs at the same time (many examples are included).
- On-board timer lets you time any interval up to 48 days long down to the nearest millisecond.

The TIMEMASTER II includes 2 disks with some really fantastic time oriented programs (over 40) including appointment book so you'll never forget to do anything again. Enter your appointments up to a year in advance then forget them. Appointment book will remind you in plenty of time. Plus DOS dater so it will automatically add the date when disk files are created or modified. The disk is over a $200.00 value along—we give the software others sell. All software packages for business, data base management and communications are made to read the TIMEMASTER II. If you want the most powerful and the easiest to use clock for your Apple, you want a TIMEMASTER II.

PRICE $129.00

Super Music Synthesizer
Improved Hardware and Software

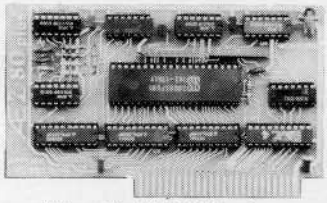




- Complete 16 voice music synthesizer on one card. Just plug it into your Apple, connect the audio cable (supplied) to your stereo, boot the disk supplied and you are ready to input and play songs.
- It's easy to program music with our compose software. You will start right away at inputting your favorite songs. The Hi-Res screen shows what you have entered in standard sheet music format.
- Now with new improved software for the easiest and the fastest music input system available anywhere.
- We give you lots of software. In addition to Compose and Play programs, 2 disks are filled with over 30 songs ready to play.
- Easy to program in Basic to generate complex sound effects. Now your games can have explosions, phaser zaps, train whistles, death cries. You name it, this card can do it.
- Four white noise generators which are great for sound effects.
- Plays music in true stereo as well as true discrete quadraphonic.
- Full control of attack, volume, decay, sustain and release.
- Will play songs written for ALF synthesizer (ALF software will not take advantage of all our card's features. Their software sounds the same in our synthesizer.)
- Our card will play notes from 30HZ to beyond human hearing.
- Automatic shutoff on power-up or if reset is pushed.
- Many many more features.

PRICE $159.00

Z-80 PLUS!



- TOTALLY compatible with ALL CP/M software.
- The only Z-80 card with a special 2K "CP/M detector" chip.
- Fully compatible with microsoft disks (no pre-boot required).
- Specifically designed for high speed operation in the Apple IIe (runs just as fast in the II+ and Franklin).
- Runs WORD STAR, dBASE II, COBOL-80, FORTRAN-80, PEACHTREE and ALL other CP/M software with no pre-boot.
- A semi-custom I.C. and a low parts count allows the Z-80 Plus to fly thru CP/M programs at a very low power level. (We use the Z-80A at fast 4MHZ.)
- Does EVERYTHING the other Z-80 boards do, plus Z-80 interrupts.

Don't confuse the Z-80 Plus with crude copies of the microsoft card. The Z-80 Plus employs a much more sophisticated and reliable design. With the Z-80 Plus you can access the largest body of software in existence. Two computers in one and the advantages of both, all at an unbelievably low price.

PRICE $139.00

Viewmaster 80

There used to be about a dozen 80 column cards for the Apple, now there's only ONE.

- TOTALLY Videx Compatible.
- 80 characters by 24 lines, with a sharp 7x9 dot matrix.
- On-board 40/80 soft video switch with manual 40 column override
- Fully compatible with ALL Apple languages and software—there are NO exceptions.
- Low power consumption through the use of CMOS devices.
- All connections are made with standard video connectors.
- Both upper and lower case characters are standard.
- All new design (using a new Microprocessor based C.R.T. controller) for a beautiful razor sharp display.
- The VIEWMASTER incorporates all the features of all other 80 column cards, plus many new improvements.

	PRICE	BUILT IN SOFTSWITCH	SHIFT KEY SUPPORT	LOW POWER DESIGN	80 COLUMN HOME	7x9 DOT MATRIX	LIGHT PEN INPUTS	40 COLUMN OVERRIDE	INVERSE CHARACTERS
VIEWMASTER	**169**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
SUP'RTERM	MORE	NO	YES	NO	NO	NO	NO	YES	YES
WIZARD80	MORE	NO	NO	NO	NO	YES	NO	YES	YES
VISION80	MORE	YES	YES	NO	NO	YES	NO	NO	NO
OMNIVISION	MORE	NO	YES	NO	NO	NO	NO	YES	YES
VIEWMAX80	MORE	YES	YES	NO	NO	YES	NO	NO	YES
SMARTERM	MORE	YES	YES	NO	NO	NO	YES	YES	NO
VIDEOTERM	MORE	NO	NO	YES	NO	YES	YES	NO	YES

The VIEWMASTER 80 works with all 80 column applications including CP/M, Pascal, WordStar, Format II, Easywriter, Apple Writer II, VisiCalc, and all others. The VIEWMASTER 80 is THE MOST compatible 80 column card you can buy at ANY price!

PRICE $179.00

MemoryMaster IIe 128K RAM Card

- Expands your Apple IIe to 192K memory.
- Provides an 80 column text display.
- Compatible with all Apple IIe 80 column and extended 80 column card software (same physical size as Apple's 64K card).
- Can be used as a solid state disk drive to make your programs run up to 20 times FASTER (the 64K configuration will act as half a drive).
- Permits your IIe to use the new double high resolution graphics.
- Automatically expands Visicalc to 95 K storage in 80 columns! The 64K config. is all that's needed, 128K can take you even higher.
- PRO-DOS will use the MemoryMaster IIe as a high speed disk drive.
- Precision software disk emulation for Basic, Pascal and CP/M is available at a very low cost. NOT copy protected.
- Documentation included, we show you how to use all 192K.

If you already have Apple's 64K card, just order the MEMORYMASTER IIe with 64K and use the 64K from your old board to give you a full 128K. (The board is fully socketed so you simply plug in more chips.)

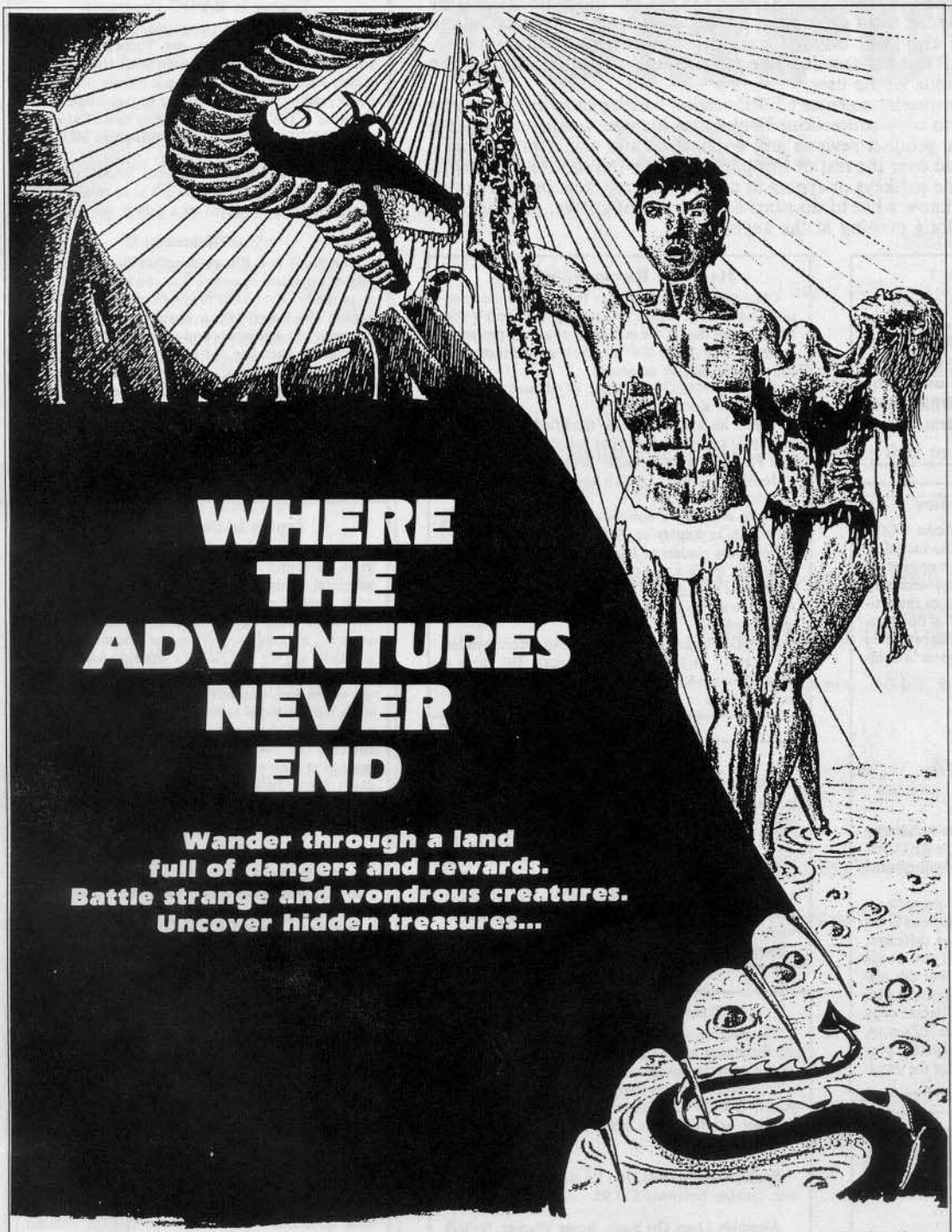
MemoryMaster IIe with 128K	**$249**
Upgradeable MemoryMaster IIe with 64K	**$169**
Non-Upgradeable MemoryMaster IIe with 64K	**$149**

Our boards are far superior to most of the consumer electronics made today. All I.C.'s are in high quality sockets with mil-spec. components used throughout. P.C. boards are glass-epoxy with gold contacts. Made in America to be the best in the world. All products work in the APPLE IIE, II, II+ and Franklin. The MemoryMaster IIe is IIe only. Applied Engineering also manufactures a full line of data acquisition and control products for the Apple; A/D converters and digital I/O cards, etc. Please call for more information. All our products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a no hassle **THREE YEAR WARRANTY.**

Texas Residents Add 5% Sales Tax
Add $10.00 If Outside U.S.A.
Dealer Inquiries Welcome

Send Check or Money Order to:
APPLIED ENGINEERING
P.O. Box 798
Carrollton, TX 75006

Call (214) 492-2027
8 a.m. to 11 p.m. 7 days a week
MasterCard, Visa & C.O.D. Welcome
No extra charge for credit cards



WHERE THE ADVENTURES NEVER END

Wander through a land full of dangers and rewards. Battle strange and wondrous creatures. Uncover hidden treasures...

In EAMON, you roam through a fantasy world where YOU control the action...and your destiny.

Enclose $4.00 for each EAMON scenario volume that you order or use our special EAMON collectors offer. Adventure scenarios are packed on double-sided disks. Minimum Order: 2 Volumes. ($8.00)

HC10

Name _____

Address _____

City _____ State _____ Zip _____

Country _____ Phone _____

VISA/MC # _____ Exp _____

Signature _____

Make checks payable to:

Computer Learning Center
P.O. Box 45202
Tacoma, WA 98445

US funds only.
No purchase orders or C.O.D.
Washington state residents add 7.8% sales tax.
Foreign orders add 20% for shipping and handling.
Please allow 4-6 weeks for delivery.

The MASTER disk is required to play any EAMON scenario.

- ☐ 01 MASTER/Beginners Cave
- ☐ 02 Lair of the Minotaur
- ☐ 03 Cave of the Mind
- ☐ 04 Zyphur River Venture
- ☐ 05 Castle of Doom
- ☐ 06 Death Star
- ☐ 07 Devil's Tomb
- ☐ 08 Abductor's Quarters
- ☐ 09 Assault of the Clone Master
- ☐ 10 Magic Kingdom
- ☐ 11 Tomb of Molinar
- ☐ 12 Quest for Trezore
- ☐ 13 Caves of Treasure Island
- ☐ 14 Furioso
- ☐ 15 The Heroes' Castle
- ☐ 16 Caves of Mondamen
- ☐ 17 Merlin's Castle
- ☐ 18 Hogarth Castle
- ☐ 19 Death Trap
- ☐ 20 The Black Death
- ☐ 21 Quest for Marron
- ☐ 22 Senators' Chambers
- ☐ 23 Temple of Ngurct
- ☐ 24 Black Mountain
- ☐ 25 Nuclear Nightmare
- ☐ 26 Assault on the Moleman
- ☐ 27 Revenge of the Moleman
- ☐ 28 Tower of London
- ☐ 29 Lost Island of Apple
- ☐ 30 Underground City
- ☐ 31 Gauntlet
- ☐ 32 House of Ill Repute
- ☐ 33 Orb of Polaris
- ☐ 34 Death's Gateway
- ☐ 35 Lair of the Mutants
- ☐ 36 Citadel of Blood
- ☐ 37 Quest for the Holy Grail
- ☐ 38 City in the Clouds
- ☐ 39 Museum of Unnatural History
- ☐ 40 Deamons Playground
- ☐ 41 Caverns of Lanst
- ☐ 42 Alternate Beginners Cave
- ☐ 43 Tomb of Y'Golonac
- ☐ 44 Operation Crab Key
- ☐ 45 Feast of Carroll
- ☐ 46 The Master's Dungeon
- ☐ 47 Crystal Mountain
- ☐ 48 Lost Adventure
- ☐ 49 The Manxome Foe
- ☐ 50 Behind the Sealed Door
- ☐ 51 Land of Death
- ☐ 52 Jungles of Vietnam
- ☐ 53 Black Castle of Nagog
- ☐ 54 Sewers of Chicago
- ☐ 55 Caverns of Doom
- ☐ 56 Valkenburg Castle
- ☐ 57 Modern Problems

Tournament Adventures

- ☐ 60 Castle of Count Fuey
- ☐ 61 Search for the Key
- ☐ 62 The Rescue Mission

EAMON Utilities

- ☐ 01 EAMON Utilities
- ☐ 02 EAMON Utilities
- ☐ 03 EAMON Utilities
- ☐ Dungeon Designer Ver 5

SPECIAL COLLECTORS'S OFFER

Every EAMON scenario & Utilities.
All for only **$200.00**

☐ YES! Send me the entire EAMON collection of 64 volumes.

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

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

New readers are advised to read over the rest of this page carefully in order to avoid frustration when following any of the softkeys or typing in any of the programs printed in this issue. Longtime readers should know what to do next: Make a pot of coffee, get out some blank disks and settle in for a long evening at the keyboard.

What Is a Softkey Anyway?

A softkey is a term which we coined to describe a procedure that removes, or at least circumvents, any copy protection that may be present on a disk. Once a softkey procedure has been performed, the disk can usually be duplicated by the use of Apple's COPYA program which is on the DOS 3.3 System Master Disk.

Following A Softkey Procedure

The majority of the articles in Hardcore COMPUTIST which contain a softkey will also include a discussion of the type of copy protection present on the disk in question and the technique(s) necessary to remove that protection. Near the end of the article, a step-by-step "cookbook" method of duplicating the disk will appear. Generally, the appropriate actions for the reader to perform will appear in boldface type. Examples are:

1) Boot the disk in slot 6

PR#6

or

2) Enter the monitor

CALL -151

It is assumed that the reader has some familiarity with his or her Apple, i.e. knowing that the RETURN key must be hit following the commands illustrated above.

Hardcore COMPUTIST tries to verify the softkeys which are published, although occasionally this is not possible. Readers should be aware that different, original copies of the same program will not always contain an identical protection method. For this reason, a softkey may not work on the copy of a disk that you own, but it may work on a different copy of the same program. An example of this is Zaxxon, by Datasoft, where there are at least 3 different protection methods used on various releases of the game.

Software Recommendations

Although not absolutely necessary, the following categories of utilities are recommended for our readers who wish to obtain the most benefit from our articles:

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

Three programs on the DOS 3.3 System Master Disk, COPYA, FID and MUFFIN, also come in very handy from time to time.

Hardware Recommendations

Certain softkey procedures require that the computer have some means of entering the Apple's system monitor during the execution of a copy-protected program. For Apple II+ owners there are three basic ways this can be achieved:

1) Place an INTEGER BASIC ROM card in one of the Apple's slots.
2) Install an old monitor or modified F8 ROM on the Apple's motherboard. The installation of a modified F8 ROM is discussed in Ernie Young's article, "Modified ROMS", which appeared in Hardcore COMPUTIST no. 6.
3) Have available a non-maskable interrupt (NMI) card such as Replay or Wildcard.

Longtime readers of Hardcore COMPUTIST will vouch for the fact that the ability to RESET into the monitor at will, greatly enhances the capacity of the Apple owner to remove copy protection from protected disks.

A 16K or larger RAM card is also recommended for Apple II or II+ owners. A second disk drive is handy, but is not usually required for most programs and softkeys.

Requirements

Most of the programs and softkeys which appear in Hardcore COMPUTIST require an Apple II+ computer (or compatible) with a minimum 48K of RAM and at least one disk drive with DOS 3.3. Occasionally, some programs and procedures have special requirements such as a sector editing program or a "nonautostart" F8 monitor ROM. The prerequisites for deprotection techniques or programs will always be listed at the beginning article under the "Requirements:" heading.

Recommended Literature

The Apple II and II+'s come bundled with an Apple Reference Manual, however this book is not included with the purchase of an Apple //e. This book is necessary reference material for the serious computist. A DOS 3.3 manual is also recommended.

Other helpful books include:

Beneath Apple DOS, Don Worth and Peter Leichner, Quality Software. $19.95.

Assembly Lines: The Book, Roger Wagner, Softalk Books. $19.95.

What's Where In The Apple, Professor Lubert, Micro Ink. $24.95.

Typing in BASIC Programs

When typing in basic programs, you will often encounter a delta ("▲") character. These are the spaces you MUST type in if you wish your checksums to match ours. All other spaces are merely printed for easier reading and don't have to be typed unless they are after a DATA statement. Any spaces after the word DATA that aren't delta characters MUST be ommitted!

It is a good idea to SAVE your BASIC program to disk frequently while typing it in to minimize the loss of data in the event of a power failure.

Checksoft

Checksoft is a Binary program that checks Applesoft programs to ensure that you have keyed them in properly. Every bin program we print has companion checksums which consist of the Applesoft program's line numbers and a hexadecimal (base 16) number for each line. After keying in a BASIC program, BRUN checksoft and compare the checksums for every line that Checksoft generates with those at the end of the program. If you use Checksoft and make a typing error, your checksums will differ from ours beginning at the line where you made the error.

Typing in Binary Programs

Binary programs are printed in two different formats, as source code and as object code in a hexadecimal dump. If you want to type in the source code, you will need an assembler. The S-C Macro Assembler is used to generate all the source code which we print, although any assembler whose use you understand will do just fine for entering source code. Binary programs can also be entered directly with the use of the Apple monitor by typing in the bytes listed in the hexdump at the appropriate addresses. Be sure to enter the monitor with a **CALL -151** before entering the hexdump. Don't type the checksums printed at the end of each line of the hexdump and don't forget to BSAVE binary programs with the proper address and length parameters listed in the article.

Checkbin

Like Checksoft, Checkbin also generates checksums, but was designed to check binary (machine language) programs.

Whenever Hardcore COMPUTIST prints a hexdump to type in, the associated Checkbin generated checksums are printed after every 8 bytes and at the end of every line.

Checksoft and Checkbin were printed in Hardcore COMPUTIST no. 1 and the Best Of Hardcore Computing and are sold on Program Library Disk No. 1 and the Best Of Hardcore Library Disk.

Let Us Hear Your Likes and Gripes

New and longtime readers of Hardcore COMPUTIST are encouraged to let us know what they like and don't like about our magazine by writing letters to our INPUT column. Our staff will also try to answer questions submitted to the INPUT column, although we cannot guarantee a response due to the small size of our staff. Also, send your votes for the softkeys you would like to see printed to our "Most Wanted List."

How-To's Of Hardcore

If you are reading our magazine for the first time, welcome to Hardcore COMPUTIST, a publication devoted to the serious user of Apple II and Apple II compatible computers. We believe our magazine contains information you are not likely to find in any of the other major journals dedicated to the Apple market.

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

Publisher/Editor
Charles R. Haight
Technical Editors
Gary Peterson Ray Darrah
Production & Graphics
Lynn Campos-Johnson
Circulation
Valerie Robinson Michelle Frank
Business Manager
Ken Fields
Advertising
Attn: Valerie Robinson
Advertising Department
3710 100th Street SW
Tacoma, WA 98499
Printing
Grange Printing, Inc.
Seattle, WA
Publishing
Softkey Publishing
P.O. Box 44549
Tacoma, WA 98444
USA

Address all advertising inquiries to Hardcore COMPUTIST, Advertising Department, 3710 100th St. SW, Tacoma, WA 98499. Address all manuscripts and editorials to: Hardcore COMPUTIST, Editorial Department, P.O. Box 44549, Tacoma, WA 98444.

MAILING NOTICE: Change of address must be postmarked at least 30 days prior to move. Paste your present mailing label on postal form 3576 and supply your new address for our records. Issues missed due to non-receipt of change of address may be acquired at the regular back issue rate.

SUBSCRIPTION INFORMATION: Rates for one year are as follows: U.S. $25, 1st Class, APO/FPO, and Canada $34, Mexico $39, Foreign Airmail $60, Foreign Surface Mail $40. Subscription inquiries should be directed to Hardcore COMPUTIST, Subscription Department, P.O. Box 44549, Tacoma, WA 98444.

DOMESTIC DEALER RATES sent upon request, or call (206) 581-6038.

Hardcore COMPUTIST

THIS ISSUE:

INPUT INPUT INPUT

Keyboard Recommendation

I am replying to a letter from T. Militello is Hardcore COMPUTIST Vol. 3, No. 3 (#7). I recently went to a computer show in San Diego where I saw and purchased a detachable keyboard with ten programmable keys, each capable of storing 48 keystrokes including RETURNs. The address of the company selling the keyboard is:

Super Computer (original name)
1101 S. Grand
Suite. J
Santa Ana, CA 92705

The keyboard costs $150 in the catalog, but I got it for $115 at the show (model number PC-5500). I also noticed a letter telling how to change drive speed on the RANA Elite I disk drive. I have an Apple drive and an Indus GP drive. On my Indus, I found that the variable resistor labeled "R31"near the center of the circuit board controls drive speed.

Tom Moeller
Glen Falls NY

More on the Bit Copiers

Thanks to Phillip Romine for evaluating the five main bit copier programs in your Hardcore COMPUTIST Magazine No. 8. There is no question that EDD Version III is the fastest and most reliable bit copier available today. It has been able to backup every protected disk which I own, with the exception of Zardax. And yes, although EDD III is heavily protected, it is powerful enough to copy itself quite easily. EDD III backs up Dollars and Sense without any parameter changes. By the way, on page 32 of your magazine, you have listed the EDD III parameters for track 0A to be used when backing-up Locksmith 5.0. Your parameter 48=0 should be 4B=0. Not mentioned in this article is that Locksmith 5.0 can be made into straight copyable disk by changing Byte 6F on Track F, Sector E from C6 to 0F.

With reference to Copy II Plus, Version 4.4C has recently been released. The bit copier program has been greatly improved; however, the utility section fortunately remains unchanged. The initial menu should feature the Track/Sector map next to the Sector Editor because the two are used so frequently together. Also, if one encounters an I/O Error in the Sector Editor, the program goes back to the initial menu, rather than permitting you to simply read or write from or to another track. Furthermore, the Sector Editor features inverse video and inverse flashing in the text entry portion of the screen. The inverse video and flashing makes the cursor very difficult to locate and one must rely on the Hex entry portion occupying the left side of the screen in order to make the desired entries. Central Point Software did supply interesting information that permits one to do such things as modify track 0 of Dollars and Sense so that the boot program of Dollars and Sense can be copied with any straight copy program. This update documentation also provides good discussion on software protection, and ways to easily backup such programs as Miner-2049er, Wizardry, Ultima III, Zaxxon, Bank Street Writer, etc. Your recent program listing of the sector editor that will modify protected disks is appreciated because Copy II Plus will give an I/O error on most protected tracks.

With reference to Locksmith 5.0, the published parameter lists have been distributed. This program suffers, from among other things, the need for improvement in the documentation. An example in the parameter listing is the frequent inability to discern "O"from "0". Furthermore, when performing a Syntax Check on a parameter listing that has been entered as a Text file, the cursor will show the line where the error has occured, and it frequently is a matter of spacing. An example would be "PAT1 STAT1"will yield a Syntax error unless all spaces are eliminated.

In summary, it is highly recommended that one purchase Copy II Plus for the utility programs, and there is absolutely no question that EDD III is the finest and most reliable bit copier.

Roger M. Levin
Palo Alto CA

Screenwriter 2.2 Update

The softkey that you published for Screenwriter 2.2 in Vol. 3, No.3 (#7) was not complete. It unprotects the Runoff (printout) but not the Editor. The complete softkey is as follows:

Using a disk editor (Disk Fixer 3.3, Zap, etc.), read in track 0E, Sector 03. Locate the sequence "20 00 6E"(byte offset of rest of 49). Change it to "EA EA EA". The editor part will now work.

The rest of the softkey was correct as published, that is: read in track 0F, sector 07. Locate the sequence "20 00 7F"(byte offset of 90). Change it to "EA EA EA". The runoff part will now work.

Please note that it is advisable to make these changes on a COPYAed copy of the original Screenwriter 2.2 disk; you should never change the original.

Andrew Reiffenstein
Edmonton, Alberta

Apple //e RESET

How can I "RESET into the monitor" without an Integer Firmware Card? I have an Apple //e.

Dick Armstrong
Flanders, NJ

Mr. Armstrong: *If you don't have an Integer Firmware card, about the only surefire means of halting the execution of a protected program is to have a ROM containing modified monitor code. The only place we know of that sells a modified ROM for the Apple //e is: Synergetics, 746 First Street, Box 809-AAL, Thatcher AZ, 85552. The cost is $19.95.*

Modified ROM with CP/M

The Modified ROM's article has a bug. The adapter is missing an enable line and some programs will not work. CP/M is one that will not boot up without this line. To cure this, a "Promette" from CMW. Inc., P.O. Box 33651, Dayton, Ohio 45433 will work. It has an active device to return this enable line. By the way, I will burn an 2716 EPROM with the modified rom and ship it for $7.50 each. I will also burn any program into most EPROM's at my cost. Keep up the good articles. Your mag is the best.

Paul Harvey
223 W. Bluebell Ave.
Anaheim, CA 92802

More on LOGO

Thanks to Ms. Gygi for the tip on bypassing the nibble count on Apple LOGO. On the version I have, however, bytes $79 to $7B were not as described. With a little experimenting, I found that changing the bytes $5D to $5F as described below did the same trick. Here are the details:

Modify Track $0 Sector $0A as follows:

```
Change
 Byte: $13 $14 $15 $22 $23 $24 $5D $5E $5F
 From: $20 $00 $30 $80 $8C $C0 $4C $00 $40
 To:   $EA $EA $EA $4C $55 $40 $EA $EA $EA
```

All but the last three bytes are as in Ms. Gygi's article. By the way, if you use Copy II Plus, setting parameter 3C=04 lets 4 tracks at a time be copied which will speed

up the process significantly.

Mark Saks
Philadelphia PA

Corrections to Zaxxon Softkey

In Hardcore COMPUTIST Vol. 3, No. 3 (#7) you had a softkey for Zaxxon. This contained an omission and an error, but they may both be the same thing.

First, the possible error. On page 11, the bottom two charts are identical. I suspect that you accidentally ran the same chart twice. Also, the sector editor charts didn't work for my copy of Zaxxon. I had one of the very first ones which used the Mockingboard. The technique was basically correct, but my version had different addresses for both the epilog checker and the nibble counter. Here are the changes for my version:

For Early Mockingboard Version:

Trk	Sector	Byte	From	To
$00	$07	$1F	$A9	$4C
$00	$07	$20	$00	$C0
$00	$07	$21	$85	$08
$00	$04	$4F	$CC	$DE
$00	$04	$50	$D0	$EA
$00	$04	$51	$AE	$EA

These changes work for my version. I really appreciate the depth of the article. Without it, I couldn't have found the correct locations. Please try to have more of your articles explain things this well.

Wesley R. Felty
Bothell WA

Mr. Felty: You are absolutely right, we did manage to duplicate the tables in the Zaxxon article. Thanks for pointing this out to us.

Computing in Kenya

Congratulations on a continually improving magazine. Your new format, with CORE included in each issue of Hardcore COMPUTIST is excellent. I especially appreciate the increased number of Softkeys and the more detailed, introductory-level explanations of cracking techniques. After reading Vol.3, No.3 (Hardcore COMPUTIST #7), I have a few comments and questions.

For Thomas Dragon. Your article on liberated backups with copy protection mentioned the HSD disks as one exception to the procedure because of their "incredibly complex protection system." Can you (or any other reader) give us details about breaking this system? I have HSD Stats Plus, and have been working on modifications to fix one bug in the program, customize some of the routines, and permit the program and data disks to be loaded into my RAMDISK 320 disk emulator. Using a Snapshot 2 NMI card I managed to copy each of the Applesoft programs on the distribution disk, but there is a text file with data for the graphics routine that I haven't been able to copy. How can the disk be completely broken?.

For Brian Burns. Thanks for the lower case output modification to the Infocom games. It's great! It must be an older version, since the eleven byte upper casing routine lives on track 2, sector 9, bytes 9D through A7 instead of 4C through 56. The codes for inversing and the text window seem to be in different locations, too. I found the 3F code for inversing on track 2, sector 7 at byte EA (instead of 9D), but I can't find where to change Ø1 to Ø2 to leave a blank line between the status line and the rest of the game text. Can you tell me where this code would be on my version of Zork II, or explain how I could go about locating it?

For Dr. Thomas E. Militello. If you have an Apple II Plus you should consider the Videx Enhancer][as an alternative to an add-on keyboard. It's much cheaper. List price is $149, but Conroy La Pointe (which has given me good service in the past) sells it for $99. And it's very flexible. Every key on the standard keyboard can be reprogrammed easily for one or many characters. The full ASCII character set is available and you can get alternate font chips. Keys repeat automatically when held down. Best of all, Videx offers the best customer support in the business. I wouldn't trade my][plus with the Enhancer for a //e or a fancy keyboard.

For you Most Wanted List (a great idea!), how about more softkeys for educational programs? For example, I haven't been able to back up my kids' Spinnaker or Xerox Stickybear programs. Also, could some expert reader provide a parms list for Locksmith 5.Ø? In spite of Omega's ads, there seem to be a lot of programs that the latest Locksmith fails to copy under its default settings, and Omega's support has gone from weak to deceased.

Philip R. Christensen
Radio Language Arts Project
Kenya Institute of Education
Nairobi, Kenya

Mr. Christensen: Sorry, but we do not have any information on the type of protection being used on the HSD disks. These disks are now on our "Most Wanted List."

One of our readers, John Liska of Stanford, CT, reported that on older versions of Zork the following modifications were necessary in order to obtain lower case: Step 7 - The modification needs to be made at byte $EA of track $2, sector $7. Step 8 - The modification needs to be made at byte $BC of track $2, sector $B.

You are absolutely right about Omega's Locksmith support. Their customer support is not the only thing to have died. Omega has filed for bankruptcy and the marketing rights

Continued on page 8

ATTENTION ADVENTURERS!

Adventure Data Base

Hardcore COMPUTIST is looking for more adventure hints to any of the popular adventure/fantasy games sold for the Apple II, II Plus or //e. These will be used in our regular column, ADVENTURE TIPS.

Your Clues, Please

We prefer that these hints not be dead giveaway solutions to dilemmas presented by the particular game, but instead contain just enough information to nudge the stumped adventurer towards the solution to his/her problem.

How & Where

So, if you know how to open the jewel-encrusted egg, how to plug the hole in the rowboat, where to find the key to the treasure chest, or any other tidbits of information that may be helpful to your fellow traveler, please send this information on a 3×5 postcard to:

Hardcore COMPUTIST
Attention: Adventure Tips
P.O. Box 44549
Tacoma, WA 98444

P.S. Please don't forget to include the name of the adventure game to which your hint pertains and the name of the manufacturer.

ADVENTURE TIPS

READERS' SOFTKEY & COPY EXCHANGE

More On Sensible Speller IV

By Lamont Cranston

Sensible Speller Ver 4.0c & 4.1c
Sensible Software
6619 Perham Drive West
Bloomfield, MI
$125.00

Requirements:
Apple II or II plus
Blank disk
Bit Copy program
Sector Editor

Editors Note: The Sensible Speller Softkey published in Hardcore Computist no. 9 was based upon revision 4.0d of the program. We verified the softkey using that revision. Several readers however reported that the softkey would not work on their version of the program. Apparently there are at least six different revisions, each having a different menu entry point. We are trying to gather information on the different versions, but as yet we do not have a verified procedure that will work on anything other than 4.0d.

Because of the interest in Sensible Speller we are publishing a softkey for revisions 4.0c and 4.1d. This method has not been verified by our staff. If you try this softkey, please let us know your results.

The protection scheme on Sensible Speller IV is based upon the disk having a very strange and complicated boot. The complexity of the boot requires a lot of extra code which apparently exists solely to handle the disk's odd format. Tracks $0-$1 and $4-$5 are used, tracks $2-$3 and $6-$22 are all normal DOS 3.3 format (track $3 is technically CP/M, but COPYA will still copy it). The title picture resides on tracks $6-$7 and is loaded on hi-res page 2 at $4000-$5FFF. Track $8-$B contain the SS main menu routines. Other routines, such as the spelling checker itself, are loaded later by the menu module.

Knowing this, all we have to do is load the menu program with its associated support routines and fix its expected environment. Since the useful program exists on normal DOS 3.3 tracks, we will write track $0 (on a copy of the SS disk) with the DOS 3.3 BOOT1 and RWTS sectors, then rewrite the DOS BOOT2 (track $0, sector $1) routine that's loaded at $B700 as part of RWTS so that it loads in the Sensible Speller menu. A couple of routines also have to be added to BOOT2 so that the main menu will work properly once loaded.

BOOT2 is called from BOOT1 after the RWTS has been loaded. The SS menu program loads and runs at $800-$47FF. The BOOT2 code needed for SS has to do the following things:

1) Load the SS menu program in memory at $800-$47FF.
2) Load track $2, sectors $E-$F in memory at $800-$9FF.
3) Store a $00 in the prompt register at $33.
4) Copy the motherboard F8 monitor ROM into a language card if it is present.
5) Set normal I/O.
6) Exit to the entry point of the SS menu.

Step 3 is necessary because Sensible Speller does not seem to work correctly if a prompt is found at $33 and step 4 is needed becuase SS will crash if a language card without a monitor image is found in slot 0.

The DOS BOOT2 loader is well documented in Beneath Apple DOS. Essentially it is designed to load a specified number of sectors into memory from consecutive disk sectors into consecutive pages of memory. The Input/Output Block (IOB) and Device Control Table (DCT) are present at the end of the BOOT2 sector.

I have used this patch on Sensible Speller version 4.0c and 4.1c. The source code listing is for version 4.1c (the PFS:Write compatible version). The 4.0c version (the Pascal compatible version) menu has a different entry point ($33B8), so the JMP in BOOT2 at $B790 (lines 1070 and 1790 in source code) must be changed accordingly.

If you have an earlier version of Sensible Speller you can use the Inspector to search for the entry. To do this, boot up the Inspector, set the buffer to $800 and read track $8, sector $0 and then continue holding down the "I" key until you have filled memory from $800 to $47FF with consecutive sectors from the disk. Search the buffer for a likely entry point and use it to put into the modified BOOT2 code. A conspicuous feature of the entry is that it will store its address into $3E-$3F.

The modified BOOT2 code will not load the hi-res Sensible Speller logo because this would make the code longer than one sector, although it could be rewritten to load in a second sector.

Sectors $E-$F of track 2 are loaded over the menu code because this is where SS stores its configuration data. These sectors can be read by any sector editor.

The VTOC on the copied disk can be altered to free up track $1 and track $4-$5 for data since they are not used by Sensible Speller. If you wish to implement an extended boot to load DOS or the hi-res picture, the extra data can be stored on these sectors or on the unused sectors of track $0.

The Necessary Steps

1) Boot up DOS 3.3 and INIT a slave disk

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

2) Use a bit copier to copy tracks $2-$3 and $6-$22 from the Sensible Speller original onto the disk initialized in step 1.

3) Use a sector editor to write the modified BOOT2 code onto track $0, sector $1 of the copied disk. Do not forget to change the entry point if you have a version other than 4.1c.

4) That's all. Now start misspelling!

I was motivated to deprotect Sensible Speller because of Sensible Software's heavy protection of this utility program (probably soon to get heavier), its serial numbering and restrictive licensing agreement. This licencing agreement states that you are only allowed to read the program into memory and execute it. Copying the program is, of course, verboten. Apparently no one at Sensible Software has read the copyright law which asserts the right to make archival backup copies.

Other factors which motivated me were Sensible Speller's refusal to boot on an Apple //c (the computer rightly says there is something wrong with the disk) and Sensible Speller's withdrawal of Pascal support with the release of version 4.1c. To me, it seems as if dropping an entire operating system in favor of Word Handler or Cameo seems to be a ploy to get those coveted serial number registration cards. All in all, they should have spent the money they spent on copy protection on some extra features for the program.

I did not find the program particularly difficult to crack (about 6 hours work all told). I want to emphasize that I *did buy* the original program and do not condone piracy. However, nobody locks me out of a program I have purchased, either.

Sensible Speller Boot 2 Object Code

```
B700: 8E E9 B7 8E F7 B7 A9 01
B708: 8D F8 B7 8D EA B7 AD E0
B710: B7 8D E1 B7 A9 0B 8D EC
B718: B7 A9 0F 8D ED B7 A0 48
B720: 88 8C F1 B7 A9 01 8D F4
B728: B7 8A 4A 4A 4A 4A AA A9
B730: 00 85 33 9D F8 04 9D 78
B738: 04 20 93 B7 A9 01 8D F8
B740: B7 8D EA B7 A9 02 8D E1
B748: B7 A9 02 8D EC B7 A9 0F

B750: 8D ED B7 A0 0A EA 88 8C
B758: F1 B7 A9 01 8D F4 B7 8C
B760: F1 B7 A9 01 8D F4 B7 20
```

```
B768: 93 B7 A2 FF EA 8E EB B7
B770: 20 93 FE 20 89 FE AD 81
B778: C0 AD 81 C0 A9 F8 A0 FF
B780: 84 3E 84 3F C8 84 42 84
B788: 3C 85 43 85 3D 20 2C FE
B790: 4C 17 35 AD E5 B7 AC E4
B798: B7 20 B5 B7 AC ED B7 88

B7A0: 10 07 A0 0F EA EA CE EC
B7A8: B7 8C ED B7 CE F1 B7 CE
B7B0: E1 B7 D0 DF 60 08 78 20
B7B8: 00 BD B0 03 28 18 60 28
B7C0: 38 60 AD BC B5 8D F1 B7
B7C8: A9 00 8D F0 B7 AD F9 B5
B7D0: 49 FF 8D EB B7 60 A9 00
B7D8: A8 91 42 C8 D0 FB 60 00
B7E0: 40 00 0A 1B E8 B7 00 B6
B7E8: 01 60 01 00 00 00 FB B7

B7F0: 00 47 00 FF 01 00 FE 60
B7F8: 01 00 00 00 01 EF D8 00
```

Sensible Speller
Boot 2 Source Code

```
1000 MEM1      .EQ $04F8     MEMORY SPOTS
THAT DOS USES TO KEEP TRACK OF
1010 MEM2      .EQ $0478     WHICH TRACK W
AS READ LAST AND WHAT TRACK IS NEXT
1020 READ.PGS .EQ $B793
1030 SETVID    .EQ $FE93
1040 SETKBD    .EQ $FE89
1050 LANG.CARD0 .EQ $C080  START OF MAGI
C MEMORY LOCATIONS THAT AFFECT EXTRA 16K
 OF MEMORY IN SLOT 0
1060 MOVE      .EQ $FE2C     MONITOR 'M' S
UBROUTINE
1070 START.SPELL .EQ $3517
1080 RWTS      .EQ $BD00     ENTRY TO RWTS
1090 FM.PRM    .EQ $B5BB     FILE MANAGER
PARAMETER LIST
1100 FM.VOL    .EQ $B5F9     FILE MANAGER
VOLUME NUMBER COMPLIMENTED
1110 STACK     .EQ $100      STACK AREA
1120
1130           .OR $B700
1140           .TF SSBOOT2
1150
1160           STX RWTS.PRM+1 NEXT SLOT
1170           STX RWTS.PRM+15 LAST SLOT
1180           LDA #1
1190           STA RWTS.PRM+16 NEXT DRIVE
1200           STA RWTS.PRM+2 LAST DRIVE
1210           LDA SECND.PRM NUMBER OF PA
GES
1220           STA SECND.PRM+1 NUMBER OF
SECTORS
1230           LDA #$0B
1240           STA RWTS.PRM+4 TRACK NUMBE
R
1250           LDA #$0F
1260           STA RWTS.PRM+5 SECTOR NUMB
ER
1270           LDY #$48
1280           DEY
1290           STY RWTS.PRM+9 BUFFER MSB
1300           LDA #1
1310           STA RWTS.PRM+12 COMMAND CO
DE
1320           TXA           A<=SLOT NUMBE
R FOUND*16
1330           LSR           DIVIDE SLOT
1340           LSR           NUMBER BY
1350           LSR           16
1360           LSR
1370           TAX           CORRECT OFFSE
T
1380           LDA #0
1390           STA $33
1400           STA MEM1,X    LAST TRACK RE
AD WAS PHASE 0
1410           STA MEM2,X    DON'T MOVE HE
AD ANY
1420           JSR READ.PGS
1430           LDA #1
1440           STA RWTS.PRM+16 LAST DRIVE
1450           STA RWTS.PRM+2 LAST DRIVE
1460           LDA #2
1470           STA SECND.PRM+1 NUMBER OF
SECTORS
1480           LDA #2
1490           STA RWTS.PRM+4 TRACK NUMBE
R
1500           LDA #$0F
1510           STA RWTS.PRM+5 SECTOR NUMB
ER
1520           LDY #$0A
1530           NOP
1540           DEY
1550           STY RWTS.PRM+9 BUFFER MSB
1560           LDA #1
1570           STA RWTS.PRM+12 COMMAND CO
DE
1580           STY RWTS.PRM+9 BUFFER MSB
1590           LDA #1
1600           STA RWTS.PRM+12 COMMAND CO
DE
1610           JSR READ.PGS
1620           LDX #$FF
1630           NOP
1640           STX RWTS.PRM+3 VOLUME EXPE
CTED = 255
1650           JSR SETVID    PR#0
1660           JSR SETKBD    IN#0
1670           LDA LANG.CARD0+1
1680           LDA LANG.CARD0+1
1690           LDA #$F8
1700           LDY #$FF
1710           STY $3E
1720           STY $3F
1730           INY
1740           STY $42
1750           STY $3C
1760           STA $43
1770           STA $3D
1780           JSR MOVE      F800<F800.FFF
FM
1790           JMP START.SPELL
1800 NXT.SECT LDA SECND.PRM+5 RWTS POINT
ER MSB
1810           LDY SECND.PRM+4 RWTS POINT
ER LSB
1820           JSR RWTS1
1830           LDY RWTS.PRM+5 SECTOR NUMB
ER
1840           DEY           DONE WITH TRA
CK?
1850           BPL NXT.TRK   NO,BRANCH
1860           LDY #$0F      NEXT TRACK, S
ECTOR 15
1870           NOP
1880           NOP
1890           DEC RWTS.PRM+4 TRACK NUMBE
R
1900 NXT.TRK   STY RWTS.PRM+5 SECTOR NUMB
ER
1910           DEC RWTS.PRM+9 BUFFER MSB
1920           DEC SECND.PRM+1 NUMBER OF
SECTORS
1930           BNE NXT.SECT
1940           RTS
1950 RWTS1     PHP
1960           SEI
1970           JSR RWTS
1980           BCS RWTS.ERR
1990           PLP
2000           CLC
2010           RTS
2020 RWTS.ERR PLP
2030           SEC
2040           RTS
2050 SET.WRIT LDA FM.PRM+1 SET RWTS
2060           STA RWTS.PRM+9 FOR WRITING
2070           LDA #0
2080           STA RWTS.PRM+8 RWTS BUFFER
 LSB
2090           LDA FM.VOL
2100           EOR #$FF
2110           STA RWTS.PRM+3 VOLUME EXPE
CTED
2120           RTS
2130 ZERO.BUF LDA #0         ZERO CURRENT
BUFFER
2140           TAY
2150 .1        STA ($42),Y   LOCATION $42
HOLDS THE POINTER TO THE CURRENT BUFFER
2160           INY
2170           BNE .1
2180           RTS
2190           .HS 00        UNUSED
2200 SECND.PRM .HS 40000A1BE8B700B6 DOS
SECOND STAGE BOOT LOADER PARAMETER LIST
2210 RWTS.PRM .HS 016001000000FBB7004700
FF0100FE6001
2220 *RWTS.PRM IS THE RWTS PARAMETER LIS
T
2230           .HS 0000      UNUSED
2240 DCT       .HS 0001EFD8 DEVICE CHARAC
TERISTICS TABLE
2250           .HS 00        UNUSED
```

Copying Essential Data Duplicator III

By Joseph Leathlean

Essential Data Duplicator III
Utilico Microware
3377 Solano Ave., Suite 352
Napa, CA 94558
$79.95

Requirements:
48K Apple II or II plus
One disk drive
Blank disk with no HELLO Program
Integer Firmware Card or other means to RESET into the monitor

The method of backing up the Essential Data Duplicator I which appeared in Hard-

core COMPUTIST no.8 can be modified so that the technique also works on EDD III. Only two changes need to be made.

On EDD III the main copy protection scheme starts at $3EE8 rather than at $21C9. This byte has to be changed to 60 (RTS).

The program's main entry point has also been moved from $095E. On the version dated January 25, 1984, the new entry point is $0955. On later versions the entry point is $0953. You may have to experiment with your version to find the proper entry point.

So if you want to perform the Softkey on EDD III, follow the procedure explained in the article on pages 25-26 of Hardcore COMPTIST no. 8 and make the following changes:

1) Place a 60 at address $3EE8 instead of at $21C9.

2) For the January 20, 1984 release of EDD change address $4026 in the hexdump to a 55. For later versions try changing it to a 53.

Using Super IOB To Copy Krell LOGO

By Andrew Harrison

Krell LOGO
Krell Software Corp.
1320 Stony Brook Road
Stony Brook, NY 11790
$89.95

Requirements:
Apple II plus with 64K or //e
Krell LOGO
Super IOB program
Swap Controller
One blank disk
Integer Firmware card or other means of entering the monitor

Krell LOGO version A can be copied with the use of the Super IOB program which has the Swap Controller installed. Super IOB and the Swap Controller can both be found in Hardcore COMPUTIST no. 9. You will also need some way to RESET into the monitor so that the LOGO RWTS can be saved to disk.

1) Boot up with the original Krell LOGO disk and then RESET into the monitor.

2) Move the RWTS to a safe location

```
1900<B800.BFFFM
```

3) Boot up a DOS 3.3 slave disk with no HELLO program.

```
PR#6
```

4) Save the Krell LOGO RWTS to the same disk that has the Super IOB program on it

```
BSAVE LOGORWTS,A$1900,L$800
```

5) Format a blank disk with the HELLO program listed below

```
FP
10 PRINT CHR$(4)"BLOAD
   BANNER.SHP"
20 PRINT CHR$(4)"RUN
   KRELL.START"
INIT HELLO.LOGO
```

6) Load the Super IOB and type in or EXEC in the Swap Controller

7) Change line 10010 to read

```
10010 PRINT CHR$(4)"BLOAD
   LOGORWTS,A$1900"
```

8) Run the Super IOB program copying from the original Krell LOGO disk to the disk initialized in step 5. *Do not* reformat the duplicate disk.

```
RUN
```

9) After the copy has been made, RENAME the old HELLO file on the copied disk as LOGO.START

```
RENAME HELLO,LOGO.START
```

10) The file that is now called LOGO.START has a CALL 2167 which needs to be removed in order for the copy to work properly. LOAD the LOGO.STARTUP file and remove this CALL. It should be in the first line of the program. Be sure to reSAVE the program after you have made the change.

Parameters For DB Master Version 4

By Dr. Nibbles

DB Master Version 4 Stoneware
50 Belvedere St.
San Rafael, CA 94901
$229.00

Requirements:
Apple II or II plus
Essential Data Duplicator III
or Locksmith 5.0
Blank disk

Essential Data Duplicator III

1) Drive speed is critical for making a good copy.

2) Copy tracks $0-$6, $6.5-$11.5 and $14.5-$22.5 in mode number 1.

3) Copy tracks $12.25-$13.25 in mode number 2.

Locksmith 5.0

1) Copy track $0 in sync mode.

2) Copy tracks $1-$6 in normal mode.

3) Copy tracks $6.5-$11.5 in normal mode.

4) Copy tracks $12.25-$13.25 in sync mode.

5) Copy tracks $14.5-$22.5 in normal mode.

Softkey for Canyon Climber

By John Liska

Canyon Climber
Datasoft
9421 Winnetka
Chatsworth, CA 91311
$24.95

Requirements:
Apple II with 48K
FID
One blank disk

Canyon Climber is an excellent 3-level arcade game in which you attempt to scale a hi-res version of the Grand Canyon.

To remove the copy protection, you need to make some modifications to DOS so that errors generated by the nonstandard end of address and end of data marks are ignored. The single binary file can then be transferred to a normal 3.3 disk with the use of FID.

1) First boot up with a DOS 3.3 disk and make the modifications to DOS

```
PR #6
CALL -151
B925:18 60
B988:18 60
BE48:18
3D0G
```

2) BRUN FID and use the wildcard character ("=") to transfer the one file on the original Canyon Climber disk to a standard 3.3 disk. You must this because the file name contains embedded control characters.

```
BRUN FID
```

Once you have transferred the file, use a disk editor or Copy][+ to change the file name so that its control characters are eliminated.

Continued from page 5

for Locksmith 5.0 have been acquired by Alpha Logic Business Systems.

Looking for The Royal Navy

In Hardcore Computist No. 8, Jim Willis of West Monroe, LA mentioned that he used a program called The Royal Navy Replay Disk for making backups. I've never heard of this program. Would you please tell me how I could go about getting a copy of this program?

Thank you for your excellent magazine.

Larry Newby
Cedar Grove, WI

Mr. Newby: The Royal Navy Replay disk is available for $20.00 from; Engineering Computer Applications, Inc., P.O. Box 12518, Lake Park, FL 33403.

Arcade Machine
Broderbund Software, Inc.
1938 Fourth Street
San Rafael, Ca 94901
(415) 456-6424
$59.95

Requirements:
Apple with 48K
One disk drive, with DOS 3.3
One blank disk
Old Monitor ROM or modified F8 ROM
Super IOB
Arcade Machine

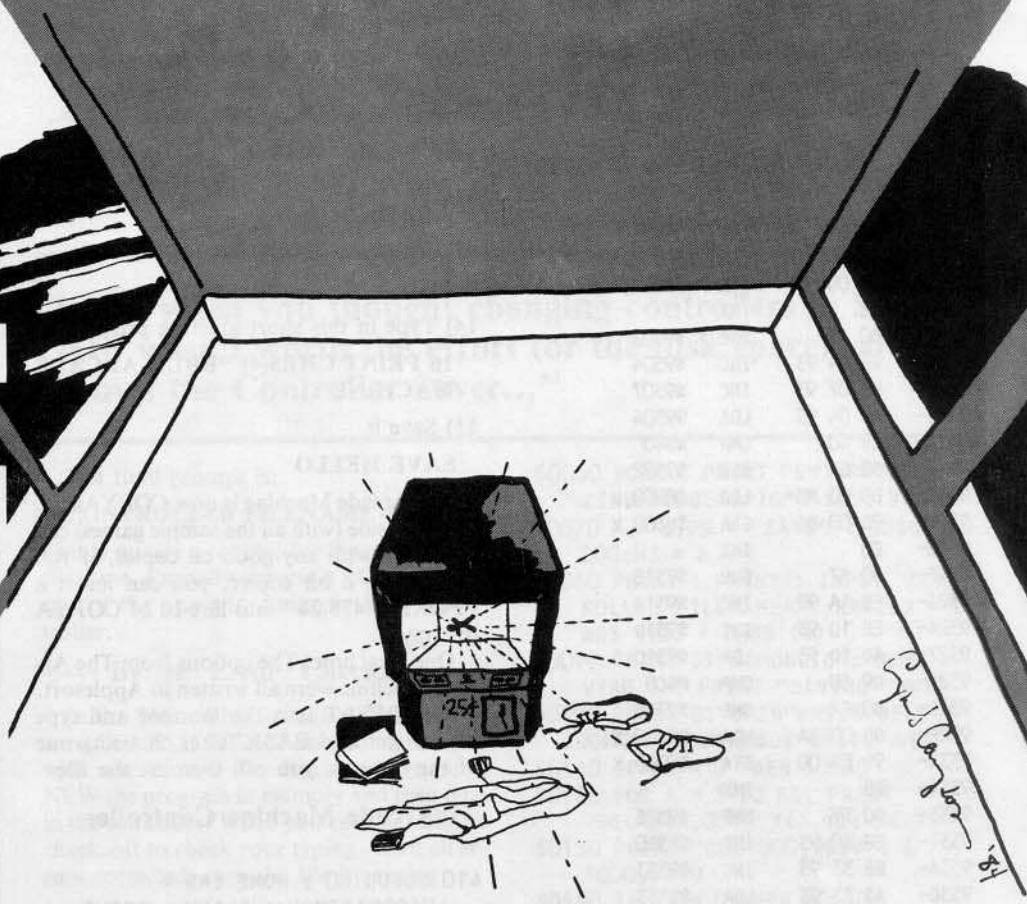
IMPORTANT IMPORTANT IMPORTANT
IMPORTANT IMPORTANT IMPORTANT

Sometimes a BASIC program can be deceiving. Often, it isn't clear (in a printout) exactly how many spaces are following a PRINT or DATA statement. Yet if you wish your checksums to be correct and the program to function as the author intended you must key in the program (spaces between quotes and after DATA statements especially) exactly.

For this reason, Hardcore COMPUTIST will be printing BASIC listings with delta characters (Δ) in all the places where you must type a space. ALL other spaces are merely inserted into the program for easy reading.

When keying in DATA statements, DON'T type any spaces after the word DATA (even if there is one printed there). If you should find an delta character after the word DATA, then type a space, otherwise DON'T! This is so your Checksoft generated checksums will match up with the ones we print for the program.

IMPORTANT IMPORTANT IMPORTANT
IMPORTANT IMPORTANT IMPORTANT



Softkey For The Arcade Machine

By Marco Hunter

Although Broderbund Software is generally very thorough with their protection schemes, they left two holes in the protection on the Arcade Machine. First of all, tracks $03 through $11 are normal DOS 3.3 tracks. These tracks store the various parts of the options available from the menu. Secondly, although the main file is heavily protected on the disk (spiral protection and what not), once loaded into memory it is fairly clean.

Since the menu options take up tracks $03 through $11 we have room for DOS (tracks 0-2), as well as room for the main file ($12-$22). Since track $11 is taken up by the menu options, we must change the location of our CATALOG. I chose to put it on track $12. Then we have a neat package (all on one disk) just like the original. When we boot our deprotected version, DOS will load the main file which from then on will access the menu options on tracks $03-$11.

Here's how.

1) First of all, boot up with a DOS that has the ability to INITialize a disk (a fast DOS is recommended).
2) Next, type in the Super IOB controller at the end of this article and SAVE it. (See the Controller Saver article on page 11 for details on checking and saving Super IOB controllers).
3) With the controller and Super IOB merged, execute the Super IOB program.

RUN

4) When Super IOB asks you if you want to format the target disk, ***you must type a "Y".*** This formatting is necessary because the controller does a special format to the disk which puts the directory on track $12 instead of $11.
5) Once Super IOB has copied tracks $03-$11, boot up The Arcade Machine.

PR#6

6) Get into the monitor when the main menu appears.
7) Move the main file and sensitive memory to safety.

2000<9600.B600M
8000<B600.C000M
8A00<0.900M

8) Boot the disk that Super IOB formatted (you should get a ?FILE NOT FOUND error, but don't worry)

6 CTRL P

9) Enter the monitor

CALL-151

10) Put a patch just before the beginning of the Arcade Machine code

8FD:4C 00 93

11) Type in this routine that moves the memory back to where it used to be

```
9300: A2 00 BD 00 20 9D 00 96
9308: E8 D0 F7 EE 04 93 EE 07
9310: 93 AD 04 93 C9 40 D0 EA
9318: BD 00 80 9D 00 B6 E8 D0
9320: F7 EE 1A 93 EE 1D 93 AD
9328: 1D 93 C9 C0 D0 EA BD 00
```

```
9330: 8A 9D 00 00 E8 D0 F7 EE
9338: 30 93 EE 33 93 AD 33 93
9340: C9 09 D0 EA 4C 00 08
```

You can check your typing by comparing it to the following disassembly:

```
9300-  A2 00      LDX  #$00
9302-  BD 00 20   LDA  $2000,X
9305-  9D 00 96   STA  $9600,X
9308-  E8         INX
9309-  D0 F7      BNE  $9302
930B-  EE 04 93   INC  $9304
930E-  EE 07 93   INC  $9307
9311-  AD 04 93   LDA  $9304
9314-  C9 40      CMP  #$40
9316-  D0 EA      BNE  $9302
9318-  BD 00 80   LDA  $8000,X
931B-  9D 00 B6   STA  $B600,X
931E-  E8         INX
931F-  D0 F7      BNE  $9318
9321-  EE 1A 93   INC  $931A
9324-  EE 1D 93   INC  $931D
9327-  AD 1D 93   LDA  $931D
932A-  C9 C0      CMP  #$C0
932C-  D0 EA      BNE  $9318
932E-  BD 00 8A   LDA  $8A00,X
9331-  9D 00 00   STA  $0000,X
9334-  E8         INX
9335-  D0 F7      BNE  $932E
9337-  EE 30 93   INC  $9330
933A-  EE 33 93   INC  $9333
933D-  AD 33 93   LDA  $9333
9340-  C9 09      CMP  #$09
9342-  D0 EA      BNE  $932E
9344-  4C 00 08   JMP  $0800
```

12) Save the main file

BSAVE ARCADE MACHINE, A$8FD,L$8D04

13) Return to BASIC

FP

14) Type in this short greeting program

10 PRINT CHR$(4) "BRUN ARCADE MACHINE"

15) Save it

SAVE HELLO

The Arcade Machine is now COPYAable. The backside (with all the sample games) can be copied with any good bit copier. If you don't have a bit copier, you can insert a "POKE 47426,24" into line 10 of COPYA to copy it.

One final note: The options from The Arcade Machine were all written in Applesoft. If you RESET into the monitor and type CTRL C to get into BASIC after choosing one of the options, you can then list the file.

Arcade Machine Controller

```
410 GOSUB 80 : HOME :A$ =
    "FORMATING" : FLASH : GOSUB
    450 : NORMAL : POKE 44033 ,18
    : POKE 44703 ,18 : POKE 44764
    ,18 : POKE 42347 ,96
415 POKE 43364 ,255 : PRINT :
    PRINT CHR$ (4 ) "INIT^HELLO,V"
    VL ",S" S2 ",D" D2 :VL = 0 :
    RETURN
1000 REM ARCADE MACHINE
1010 TK = 3 :ST = 0 :LT = 18 :CD =
    WR : POKE 47426 ,24 : GOSUB
    1120
1020 T1 = TK : GOSUB 490
1030 GOSUB 430 : GOSUB 100 :ST =
    ST + 1 : IF ST < DOS THEN 1030
1040 IF BF THEN 1060
1050 ST = 0 :TK = TK + 1 : IF TK <
    LT THEN 1030
1060 GOSUB 490 :TK = T1 :ST = 0
1070 GOSUB 430 : GOSUB 100 :ST =
    ST + 1 : IF ST < DOS THEN 1070
1080 ST = 0 :TK = TK + 1 : IF BF =
    0 AND TK < LT THEN 1070
1090 IF TK < LT THEN 1020
1100 HOME : PRINT : PRINT "DONE^
    WITH^COPY" : END
1120 POKE 44033 ,17 : POKE 44703
    ,17 : POKE 44764 ,17 : POKE
    42347 ,76
1130 CD = RD :SO = S2 :DV = D2 :
    GOSUB 80 :TK = 18 :ST = 0 :
    GOSUB 430 : GOSUB 100
1140 POKE 9985 ,18 : FOR A = 10052
    TO 10115 : POKE A ,0 : NEXT
    :CD = WR : GOSUB 80 : GOSUB
    100 :TK = 3 : RETURN
```

```
410   - $D1A9    1060  - $2E6B
415   - $0D10    1070  - $266E
1000  - $2D67    1080  - $C7FA
1010  - $D124    1090  - $F30D
1020  - $BE68    1100  - $5588
1030  - $A869    1120  - $843C
1040  - $0084    1130  - $0561
1050  - $3823    1140  - $17BE
```

DEAR AUTHOR:

Would you like to be published in **Hardcore COMPUTIST**? We would like to hear from you.

Hardcore COMPUTIST welcomes articles and submissions on a variety of subjects of interest to users of the Apple (or compatible) computers and would like to publish well-written material on the following:

- **Softkeys**
- **Hardware Modifications**
- **Advanced Playing Techniques**
- **DOS modifications**
- **Utilities**
- **Product reviews**
- **Adventure Tips**
- **Original programs of interest**
- **Do-it-yourself hardware projects**
- **General interest articles**
- **Bit-Copy Parameters**

We prefer to see your submission on a DOS 3.3 disk using an Apple (or compatible) editing program. Please enclose a double-spaced hardcopy (paper) manuscript using a dot-matrix or letter-quality printer (or typewriter). Submissions will be mailed back if adequate return packaging is included.

Hardcore COMPUTIST pays on acceptance. Rate of payment depends on the amount of editing necessary and the length of the article. Payment ranges between $10 for a short softkey, and $50 per typeset page for a full-length article. We pay more for softkeys if the original commercial disk is enclosed for verification. We guarantee the disk's return.

Softkey Publishing buys all rights as well as one-time reprint rights (for upcoming BEST OF Hardcore) on general articles, and exclusive rights on programs. We may make alternate arrangements with individual authors, depending on the merit of the contribution. At present we are not accepting fiction or poetry submissions, but **Hardcore COMPUTIST** may make an exception for an outstanding computer-related short story or poem.

For a copy of our WRITER'S GUIDE send a business-sized (20-cent) self-addressed, stamped envelope to:

Hardcore COMPUTIST
WRITER'S GUIDE
P. O. BOX 44549
TACOMA WA 98444

APT'S APT'S APT'S

Cannonball Blitz

Here is how you can reduce the hazards on the second level of play. After finishing level 1 (a difficult task), press the space bar and REPT keys simultaneously until the screen shows the next level of play. When the second level of play begins, the number of cannons will have been reduced to only two!

Loderunner

There are a couple of keys in Loderunner that may help you see reach the higher levels. They are:

CTRL SHIFT P - Adds another man to your stockpile.

CTRL SHIFT N - Transports you to the next level.

CTRL A - Aborts the current man (commit suicide before the enemy(s) get you).

The Controller Saver

By Ray Darrah

"Just when you thought changing controllers in Super IOB wasn't worth the effort (or the disk space), along comes the Controller Saver..."

Just when you thought changing controllers in Super IOB wasn't worth the effort (or the disk space), along comes the Controller Saver.

How it works

The Controller Saver (named "SAVE CONTROLLER") extracts the controller portion of Super IOB and saves it to disk as a text file. Once this is done, softkeying a disk is as easy as loading Super IOB and EXECing the controller of your choice.

Saving a controller as a text file (denoted by "T" on the left side of the catalog) takes up 1/5 the space that a different Super IOB program for each new controller would. SAVE CONTROLLER extracts lines 1000 through 9999 as well as any other lines you may have added or inserted into Super IOB.

SAVE CONTROLLER uses CRUNCHLIST II which appears in this issue on page 26 . If you haven't typed CRUNCHLIST II in yet, do so immediately.

Once CRUNCHLIST II has been BSAVEd, type in the listing on this page and

SAVE MAKE SAVER

on the same disk. Next,

RUN MAKE SAVER

This will put a file on the disk called SAVE CONTROLLER which is an EXECable image of lines 50000-50220 in the MAKE SAVER program. If you wish to alter the Controller Saver, you should make the alterations to these lines and then RUN the program and it will put the image of the new program on the disk.

How Do I Use It?

Hence forth, when you wish to save a controller, simply type

EXEC SAVE CONTROLLER

After some disk spinning and lots of prompts rolling up the screen, the computer will ask:

DELETE ANY LINES?

This is asking if you had to eliminate any line numbers of Super IOB in order for it to fit in the allocated space. If you type **"Y"**, the computer will then ask which lines you DELeted. You must reply with a range of line numbers separated by a comma. If you only deleted one line then you must type the same number twice (still separated by a comma). When you are finished, press "0,0." Now, upon the EXECing of this controller, those lines specified will be DELeted from Super IOB.

Next you are asked:

SAVE ANY EXTRA LINES?

If your controller occupies more area than lines 1000-9999, then you must type a "Y" and enter the other range(s) (just like the deletions) to be saved (besides the usual 1000 through 9999). Again, you exit this routine by typing "0,0".

The final prompt is:

CONTROLLER FILENAME=>

This is merely a request for the entry of the filename. I usually append a '.CON' to the end of the name to designate it as a controller.

Better Error Checking

In the future (this issue included), Hardcore COMPUTIST will publish controllers with checksums. To check your typing, NEW the program in memory and then type in the controller. When you're done, BRUN checksoft to check your typing. With all errors corrected, save the file by typing

EXEC SAVE CONTROLLER

(For information on delta characters (^) see IMPORTANT note on page 9.)

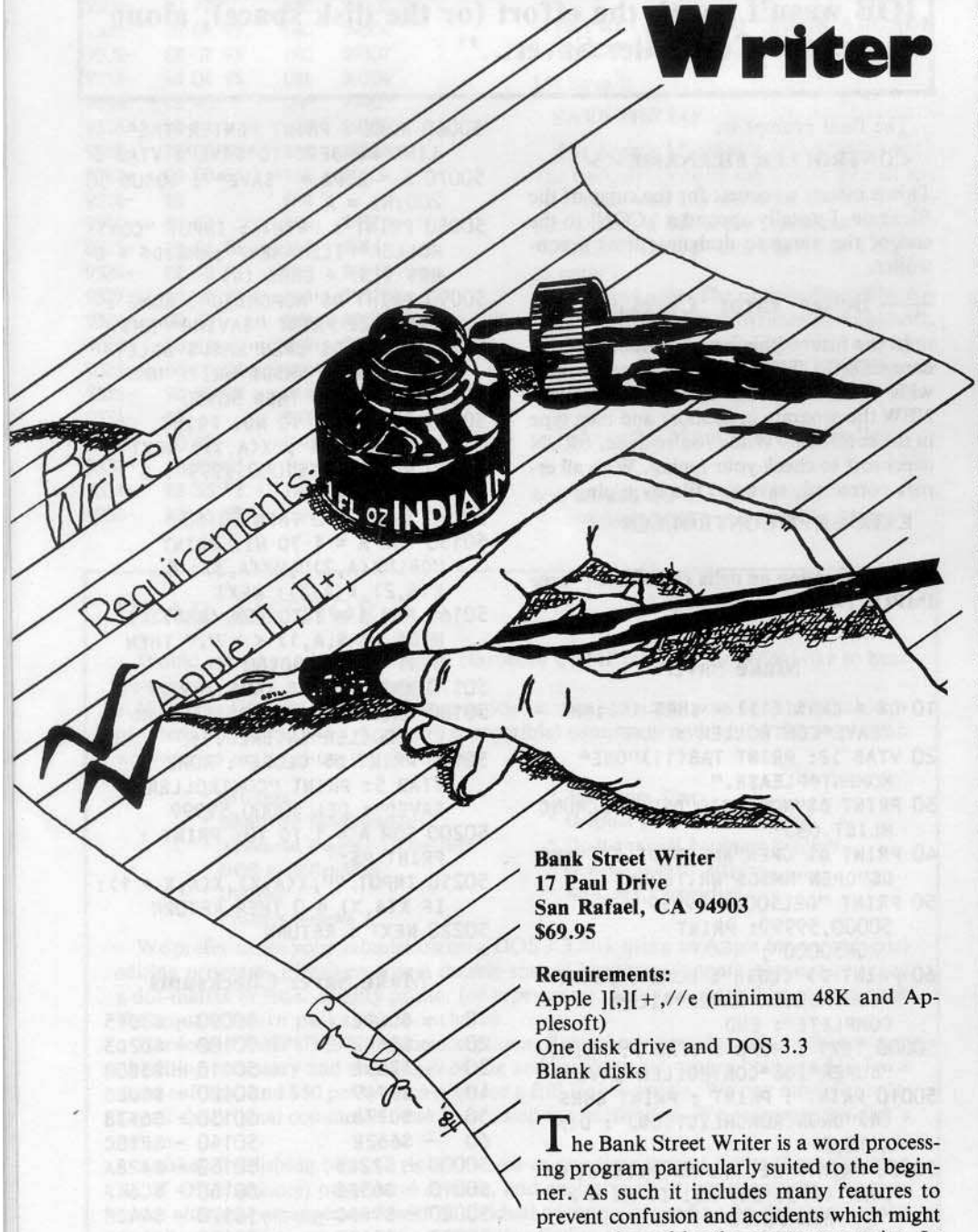
Make Saver

```
10 D$ = CHR$ (13) + CHR$ (4):NM$ =
    "SAVE^CONTROLLER": HOME
20 VTAB 12: PRINT TAB(11)"ONE^
    MOMENT^PLEASE."
30 PRINT D$"NOMONCIO"D$"BRUNCRUNC
    HLIST.OBJ"
40 PRINT D$"OPEN"NM$D$"DELETE"NM$
    D$"OPEN"NM$D$"WRITE"NM$
50 PRINT "DEL50000,59999": &
    50000,59999: PRINT
    "RUN50000";
60 PRINT D$"CLOSE": HOME : PRINT
    "CONTROLLER^SAVER^FILE^
    COMPLETE": END
50000 TEXT : HOME : VTAB 5: PRINT
    "SUPER^IOB^CONTROLLER^SAVE"
50010 PRINT : PRINT : PRINT CHR$
    (4)"BRUNCRUNCHLIST.OBJ": DIM
    X(10,3)
50020 PRINT "DELETE^ANY^LINES?^N"
    CHR$ (8);: GET A$: IF A$ < >
    "Y" THEN 50050
50030 HOME : PRINT "ENTER^THE^
    LINE^NUMBERS^TO^DELETE": VTAB
    3
50040 P$ = "DELETE^":X = 0: GOSUB
    50200:ND = A - 1
50050 HTAB 1: VTAB 10: PRINT
    "SAVE^ANY^EXTRA^LINES?^N"
    CHR$ (8);: GET A$: IF A$ < >
    "Y" THEN 50080
50060 HOME : PRINT "ENTER^THE^
    LINE^NUMBERS^TO^SAVE": VTAB 3
50070 X = 2:P$ = "SAVE^": GOSUB 50
    200:NI = A - 1
50080 PRINT : PRINT : INPUT "CONT
    ROLLER^FILENAME=>";NM$:D$ = C
    HR$ (13) + CHR$ (4)
50090 PRINT D$"NOMONCIO": HOME :
    VTAB 12: PRINT "SAVING^"NM$
50100 PRINT D$"OPEN"NM$D$"DELETE"
    NM$D$"OPEN"NM$D$"WRITE"NM$
50110 IF ND = 0 THEN 50130
50120 FOR A = 1 TO ND: PRINT
    "DEL"X(A,0)","X(A,1): NEXT
50130 PRINT "DEL1000,9999": &
    1000,9999
50140 IF NI = 0 THEN 50160
50150 FOR A = 1 TO NI: PRINT
    "DEL"X(A,2)","X(A,3): &
    X(A,2),X(A,3): NEXT
50160 FOR A = 1 TO LEN (NM$): IF
    MID$ (NM$,A,1) < > "." THEN
    NEXT : GOTO 50180
50170 NM$ = LEFT$ (NM$,A - 1)
50180 PRINT "?:?" CHR$ (34)NM$"^
    CONTROLLER^ENTERED.";
50190 PRINT D$"CLOSE": HOME :
    VTAB 5: PRINT "CONTROLLER^
    SAVED": DEL 50000,59999
50200 FOR A = 1 TO 10: PRINT :
    PRINT P$;
50210 INPUT "";X(A,X),X(A,X + 1):
    IF X(A,X) = 0 THEN RETURN
50220 NEXT : RETURN
```

Make Saver Checksums

```
10    - $D6DE
20    - $86E2
30    - $2BBE
40    - $BF49
50    - $D77B
60    - $662E
50000 - $22C5
50010 - $63F6
50020 - $9FDC
50030 - $2437
50040 - $4C9F
50050 - $21A7
50060 - $D067
50070 - $2BE3
50080 - $5673
50090 - $3DF5
50100 - $D2D3
50110 - $3B0B
50120 - $1DB6
50130 - $6F38
50140 - $F18C
50150 - $A28A
50160 - $C64F
50170 - $44C8
50180 - $46B5
50190 - $D8C3
50200 - $FAE8
50210 - $6419
50220 - $5365
```

Softkey For The Bank Street Writer



Bank Street Writer
17 Paul Drive
San Rafael, CA 94903
$69.95

Requirements:
Apple][,][+,//e (minimum 48K and Applesoft)
One disk drive and DOS 3.3
Blank disks

The Bank Street Writer is a word processing program particularly suited to the beginner. As such it includes many features to prevent confusion and accidents which might be experienced by the computer novice. In developing this softkey we have attempted to retain as many of these features as possible.

The original disk is able to operate with several different configurations of hardware. It requires (and checks for) a minimum of 48K of motherboard RAM. In addition, the presence of Applesoft is verified. If a 16K RAMCARD is installed in slot Ø, Applesoft will be placed there if necessary. If Applesoft is in motherboard ROM, the extra 16K will be used to increase the user's text memory allowing about 3,2ØØ words instead of 1,3ØØ without the card. If installed into an Apple //e, the program will use the //e's extra memory and the "open-apple, closed-apple" keys along with the four cursor movement keys.

Extensive error-trapping is performed and hitting RESET is totally non-destructive to the user's text, requiring only a press of the Return key to carry on.

After experiencing some difficulty in getting the softkey version to properly operate the disk drive, we came upon a feature not described in the manual. The program actually prevents the user from accidentally or otherwise INITing or SAVEing files to the master disk (something that our instincts prevented us from trying!).

We have made an effort to maintain as many of these user-proof features as possible in our softkey. The unlocked softkey version supports all but the following features of the original:

1) At least 48K of RAM and Applesoft must be resident in the machine at the time the softkey version is booted. Our start-up program is in Applesoft. It is possible to enter the program as an Integer Basic file and include a few statements to load Applesoft into the RAM card. The program will then work normally. All the users we know have Applesoft in ROM on the motherboard; therefore, we felt this was not a serious limitation.

2) The original boots VERY QUICKLY compared to the softkey version. Bank Street Writer (BSW) uses a powerful bulk-loading scheme to bring its files into memory. This scheme depends on the format of the data on the disk and, of course, that is what we want to change. If you're not the kind who likes to wait around, we strongly recommend the use of one of the fast DOS's such as Diversi-Dos, Hyper-DOS, ProntoDos, etc.

The Boot Process

The following paragraphs describe the boot activity of the original disk as an insight into the softkey method. You may, of course, simply follow the Softkey steps if you wish.

BSW, like all disks, has a track Ø, sector Ø that must be readable by the disk controller card's firmware at $C6ØØ (assuming the card is in slot 6). This data is loaded into $8ØØ to $8FF and executed. At this point, more data is read from track Ø and placed into memory at $1ØØØ to $17ØØ. Execution is passed to the code starting at $14ØØ.

At this point the program verifies that at least 48K of RAM is present, printing a message and stopping if it finds less. If all is well, the BSW DOS is loaded from tracks $1 and $2 into $96ØØ to $BFFF.

Next, the type of hardware configuration is determined and an identifying character is printed in the upper left corner of the

screen. There are four possible configurations which may occur. A value of Ø, 1, 2, or 3 is stored in memory location $1F depending on the type of hardware. These values represent:

Ø = No Applesoft and no ramcard in slot Ø (Screen character "Ø")
1 = Applesoft present in ROM or ramcard and 48K (Screen character "4")
2 = Applesoft in ROM and ramcard in slot Ø (Screen character "6")
3 = Apple //e (Screen character "e")

If no motherboard ROM Applesoft is found but a ramcard is detected, the type-checking code automatically loads Applesoft from tracks 3, 4 and 5 into the card and activates it. For all intents, this is the same configuration as a 48K Apple][+ ($1F=1).

If $1F=Ø after the type-check, the program informs the user that Applesoft is required and stops. For the other configurations more data is loaded into memory as follows:

$1F=	Tracks loaded..	into memory
1	$0E, $0F, $10	$6000 to $8FFF
2	$06, $07, $08	$D000 to $FFFF
	$09	$8800 to $9AFF
3	$0A, $0B, $0C	$D000 to $FFFF
	$0D	$8800 to $9AFF

Next, hi-res graphics page two is displayed and filled with the title page from tracks $18 and $19. More data is read into $4Ø0 to $7FF from track $1A. The drive head is then positioned to track $11 and the BSW DOS is cold-started with a jump to $9D84.

Track $11 contains a file directory like a normal disk. The BSW DOS loads in an Applesoft greeting program at $8Ø0 and RUNs it. The program filename is "A"followed by 7 backspace characters and 22 spaces. When we later modify normal DOS to read the catalog, this filename will not appear since the backspaces cause the visible characters to be overwritten by the spaces when printed to the screen.

Program "A"first detects if the "ESC"key has been pressed during the boot. If so, the UTILITY program is run (more on this later). Otherwise, "A"loads two other small binary files. One is called O$3Ø1 which loads at $3Ø1 and the other is INIT, loaded at $2DØ. Once these are loaded, "A"continues the boot with a CALL 2Ø48. This begins execution of the code loaded earlier at $4Ø0.

Another hardware-dependent load is performed next as follows:

$1F=	Tracks loaded	into memory
1	$1F	$9000 to $9BFF
	$20, $21	$2500 to $3CFF
	$22	$3400 to $3FFF
2	$1B, $1C	$0800 to $1FFF
3	$1D, $1E	$0800 to $1FFF

At this point, the disk access is finally completed and the drive is shut off. The next statement to be executed is at $525. This code does some initialization and patching and then starts the program with an indirect jump through $ØØØØ to either $6ØØ9 for hardware type 1 ($1F=1) or $Ø98F for types 2 and 3.

The Protection

The copy protection on the BSW disk consists essentially of changes to address and data prologue and epilogue bytes on some tracks and non-standard encoding and sectoring (for fast load) on other tracks. Half-tracks, track-arcing, track synchronizing, and nibble counting do not seem to have been used. The disk can be bit-copied (method given below) but seems to be speed sensitive which might indicate that the original was written at slower than normal speed.

The Softkey

To obtain an unprotected version of BSW, the original is allowed to load the required ranges of memory under the control of a small machine language program. At specific points in the boot, the program breaks into the monitor. A slave DOS is then booted and used to save the files in the normal DOS 3.3 format. Finally, a Basic program is written to emulate the functions of the original boot process, ending with a CALL 1317 ($525) to do the initializing and start the program.

In an effort to make things a little more sensible, notes are included with the softkey steps. There are quite a number of files to capture and many machine code patches to perform, so we suggest you *take your time and double check any typed in code.*

1) Boot DOS 3.3 into your system.
2) Enter the monitor and change the VTOC buffer (explained later)

CALL-151
B3BF:A5

3) Return to BASIC and clear the program memory

CTRL C
FP

4) Initialize a disk with this modified VTOC

INIT HELLO

5) Move the disk][controller ROM into RAM

3600<C600.C6F7M

6) Enter the following machine code

```
36F8: A9 05 8D 21 08 A9 37 8D   $0551
3700: 22 08 4C 01 08 A0 00 B9   $6AAD
3708: 2D 37 99 F8 13 C8 C0 05   $CB1F
3710: D0 F5 8D 84 14 8D 86 14   $2E64
3718: 8D A0 17 A9 4B 8D B6 14   $F3C7
3720: A9 32 8D 25 15 A9 37 8D   $B566
3728: 26 15 4C 00 14 A9 01 85   $7117
3730: 1F 60 A9 4C 8D 25 05 A9   $A619
3738: 59 8D 26 05 A9 FF 8D 27   $83C2
```

Most Wanted List

Reader response to our "Most Wanted List" has been very favorable. We have received softkeys for a number of programs previously in our list. We will be publishing the softkeys we have received just as soon as they have been evaluated and edited by our staff.

So, keep those votes and softkeys coming.

If there is a program that you have been pulling your hair out trying to back-up, let us know about it.

Hardcore COMPUTIST
Wanted List
P.O. Box 44549
Tacoma, WA 98444

If you know how to de-protect, unlock or modify any of the programs below, we encourage you to help other Hardcore COMPUTIST readers and earn some extra money at the same time. Be sure to send the information to us in article form on a DOS 3.3 diskette.

1. Apple Business Graphics
Apple Computer
2. Flight Simulator II
Sub Logic
3. Type Attack
Sirius Software
4. DB Master 4.0
Stoneware, Inc.
5. Time Is Money
Turning Point
6. Crossword MAGIC
L & S Computerware
7. Visiblend
Micro Lab
8. BPI General Ledger
Apple Computer
9. Dollars And Sense
Monogram
10. Word Juggler
Quark, Inc.
11. Catalyst
Quark, Inc.
12. Rocky's Boots
The Learning Company
13. PFS Graph
Software Publishing Corp.
14. The Statistics Series
Human Systems Dynamics
15. Millionaire
Blue Chip Software
16. Facemaker
Spinnaker
17. Story Machine
Spinnaker
18. MASTER TYPE
Scarborough Systems

```
3740: 05 4C 84 9D A9 2C 8D CA    $902C

3748: 14 8D D1 14 4C 00 14       $8B9F
```

If you can't check this code with checkbin then you may check it against the following:

```
36F8-  A9 05      LDA  #$05
36FA-  8D 21 08   STA  $0821
36FD-  A9 37      LDA  #$37
36FF-  8D 22 08   STA  $0822
3702-  4C 01 08   JMP  $0801
3705-  A0 00      LDY  #$00
3707-  B9 2D 37   LDA  $372D,Y
370A-  99 F8 13   STA  $13F8,Y
370D-  C8         INY
370E-  C0 05      CPY  #$05
3710-  D0 F5      BNE  $3707
3712-  8D 84 14   STA  $1484
3715-  8D 86 14   STA  $1486
3718-  8D A0 17   STA  $17A0
371B-  A9 4B      LDA  #$4B
371D-  8D B6 14   STA  $14B6
3720-  A9 32      LDA  #$32
3722-  8D 25 15   STA  $1525
3725-  A9 37      LDA  #$37
3727-  8D 26 15   STA  $1526
372A-  4C 00 14   JMP  $1400
372D-  A9 01      LDA  #$01
372F-  85 1F      STA  $1F
3731-  60         RTS
3732-  A9 4C      LDA  #$4C
3734-  8D 25 05   STA  $0525
3737-  A9 59      LDA  #$59
3739-  8D 26 05   STA  $0526
373C-  A9 FF      LDA  #$FF
373E-  8D 27 05   STA  $0527
3741-  4C 84 9D   JMP  $9D84
3744-  A9 2C      LDA  #$2C
3746-  8D CA 14   STA  $14CA
3749-  8D D1 14   STA  $14D1
374C-  4C 00 14   JMP  $1400
```

7) Save the boot ROM and program

BSAVE CODEBREAK,A$3600,L$14F

How CODEBREAK Works

The CODEBREAK program was developed by boot code tracing the original disk. It operates as follows:

3600.36F7 - copied from the disk controller card, reads in the track 0 sector 0 data into $800. Normally, there would be a JMP $0801 instruction at $36F8 but we have instead placed the CODEBREAK program there.

36F8.3704 - forces the BSW code to jump back to us at $3705 when it is through loading the $1000 to $17FF data from the disk.

The BSW subroutine at $13F0 calls type-checking code at $1300 and restarts the drive before it returns at $13F8. The type-checking code stores the value 0, 1, 2 or 3 into $1F according to the kind of hardware it finds. We will short circuit this code so that we can load all the files even if we only have the minimum hardware configuration.

3705.3711 - copies the small piece of code between $372D and $3731 into the BSW code starting at $13F8. This little patch will be executed after the type-checking code. It simply stores a constant into $1F, overwriting whatever the type-checking routine had placed there. To load the different hardware-dependent files, we will simply change our constant (at $372E) to a 1, 2 or 3, as needed.

3712.371A - the value in the accumulator when we reach here is a $60. This will be stored at three locations: $1484, $1486, and $17A0. The first two locations will force the first type 2 and type 3 files to be placed at $6000 instead of $D000 (in the ram card) so we don't need a ramcard to capture these files. Location $17A0 is the start of the routine that checks for minimum 48K of ram. The $60 (an RTS instruction) prevents this routine from being executed. The ramcheck code at $17A0 writes a $00 and an $FF at every page boundary from $2000 to $BF00 to verify that ram exists at these locations. We wanted to avoid clobbering anything we might want to put in memory, so we bypassed this code.

371B.371F - a $4B is patched at $14B6 to redirect the third BSW load ($8B00 to $9AFF for types 2 and 3) to start at $4B00. This prevents BSW from clobbering our relocated files at $6000 to $8FFF. $4B00 is in hi-res page two and would normally be overwritten by the logo. We will capture the logo during the type 1 file loading and then disable the logo loading for the types 2 and 3 loads.

3720.372C - used to redirect the execution back to $3732 when the BSW code is finished loading the second set of files and about to cold start the DOS.

372D.3731 - This code is copied into locations $13F8 through $13FC and it "short circuits"the error checking routine.

3732.3743 - places JMP ($4C) to $FF59 at $0525 which allows us to break into the monitor after the last files are loaded. The boot is continued by cold-starting the BSW DOS with a JMP $9D84.

3744.374E - used later to store $2C (a harmless BIT instruction) on top of two JSR instructions used to load the hi-res page two graphic. We will change the JMP $1400 at $372A to a JMP $3744 after we have saved the logo. We can then use the space from $4B00 to $5AFF for the third portion of the type 2 and 3 loads.

Whew! Let's Get Some Files

8) Put the BSW disk in the drive and execute CODEBREAK

3600G

9) When the drive stops, make the following machine code patch

1300: 2C 81 C0 20 55 13 F0 06
1308: 2C 80 C0 4C 15 13 20 3E
1310: 13 F0 1C EA EA

A 1300L should produce:

```
1300-  2C 81 C0   BIT $C081
1303-  20 55 13   JSR $1355
1306-  F0 06      BEQ $130E
1308-  2C 80 C0   BIT $C080
130B-  4C 15 13   JMP $1315
130E-  20 3E 13   JSR $133E
1311-  F0 1C      BEQ $132F
1313-  EA         NOP
1314-  EA         NOP
1315-  A2 01      LDX #$01
```

etc...

Because our softkey requires Applesoft to be installed before running, some minor changes to the BSW type-checking routine at $1300 are needed to maintain maximum compatibility. The BSW type-checker normally first checks for a ramcard by writing and verifying that every bit pattern can be stored at $D000. This would clobber Applesoft which might have been loaded there by the user before running the softkey version. To handle this situation, the above code first enables motherboard ROM with a BIT $C081 and calls the BSW "check for Applesoft" routine at $1355. This routine returns with the zero flag set if Applesoft is found. If it is found, we go to $130E where we check for a ramcard via the JSR $133E. If a ramcard is found, the code branches to the //e-checker at $132F. That code will exit with a 'type 2' if a //e is not found or with a 'type 3' if it is found.

If Applesoft is not found in motherboard ROM, execution falls through to $1308. We couldn't have gotten here with the softkey version if Applesoft were not in slot 0, so it is re-activated and the code exits indicating a 'type 1'. If the ramcard check at $130E fails, the routine also exits as a 'type 1'.

Let's Continue

10) Move the BSW code out of the way so our slave disk can boot.

1400<9000.9BFFM

11) Put the slave disk in the drive and boot it.

C600

12) We will now save as many files as possible.

BSAVE TYPECHECK,A$1300,L$69
BSAVE BSW.LOGO,A$4000,L$2000
BSAVE BSW.48K.1,A$6000,L$3000
BSAVE BSW.48K.2,A$2500,L$1B00
BSAVE BSW.48K.3,A$1400,L$AA6

13) We are ready to move on to the 'type 2' files. So hop into the monitor and load CODEBREAK

CALL-151
BLOAD CODEBREAK

14) Prepare CODEBREAK for "type 2"files

372E:Ø2

15) Enable the "hi-res graphic load disabler"

372B:44 37

16) Put the BSW disk in the drive and execute the modified CODEBREAK program

3600G

Remember, the hi-res graphics load has been disabled so don't be alarmed when you don't see the logo appear.

17) Move the file out of the way so we can boot

2800<800.1FFFM

18) Boot the slave disk again

C600G

19) Save all of the "type 2"files

```
BSAVE BSW.64K.1,A$6000,L$3000
BSAVE BSW.64K.2,A$2800,L$1800
BSAVE BSW.64K.3,A$4B00,L$FA6
```

2Ø) To get the //e files. Enter the monitor, load CODEBREAK, and tell it we want "type 3"files

```
BLOAD CODEBREAK
CALL-151
372E:Ø3
```

21) Prevent the hi-res graphic load

372B:44 37

22) Put the BSW original in the drive and startup CODEBREAK

3600G

23) Move this file out of the slave disk's boot path

2800<800.1FFFM

24) Put the slave disk in the drive and boot it by typing

C600G

25) Now save the //e files

```
BSAVE BSW.//E.1,A$6000,L$3000
BSAVE BSW.//E.2,A$2800,L$1800
BSAVE BSW.//E.3,A$4B00,L$FA6
```

What's Next?

We must now capture the initialization code which resides in pages $5 to $7. This is not as straightforward as the other files because this memory range is part of the page 1 text screen. When we jump to the monitor at $FF59 it immediately begins to print its prompt and scroll the screen. This action quickly destroys the BSW data stored there.

The data must be moved out of the text page before anything is printed. We could modify CODEBREAK to move the data before jumping to the monitor, but we hate typing in code especially when there is an interesting alternative. BSW short-circuits the character switch pointer ($36 $37) by pointing it to an RTS instruction at $FF58 just before it cold-starts its DOS. This is necessary to prevent its DOS from destroying the data in the text page, since DOS normally prints a few prompts and carriage returns to the text screen as it starts up.

We will take advantage of this by entering the monitor at $FF62 instead of $FF59. This will prevent the monitor from fixing the character switch and give us time to move the data manually. Since no characters can be printed to the screen, we will be 'blind' for a moment so type carefully.

26) Tell CODEBREAK to turn off the drive and JuMP into the monitor without resetting the character output switch

```
BLOAD CODEBREAK
CALL-151
3732:2C E8 C0 4C 62 FF
```

27) Put the BSW original in the drive and boot it with

3600G

You will not see any of the characters you type, so be careful. If you think you made a mistake, type CTRL *X and start over.*

28) Move the file out of the way (be careful)

2500<500.7FFM

29) Reset the character output switch and enter the monitor

FF59G

You should now see the monitor prompt. If you type

2500L

you should see

```
2500-  20 86 04  JSR $0486
2503-  A5 1F     LDA $1F
2505-  0A        ASL
```

etc...

If this is what you see, you've got it! If not, try steps 27 through 29 again.

3Ø) Put the slave disk back in the drive and boot it

C600G

31) Save the code beginning at $0525

BSAVE BSW.$525.$7FF,A$2525,L$2DB

Getting The CATALOGable Files

The next step is to capture the CATALOGable files from the BSW original. In order to do this, the RWTS (Read or Write a Track/Sector) routine of our slave DOS must be modified to handle the non-standard prologue and epilogue bytes of the BSW disk.

32) Enter the monitor and make the necessary changes to DOS

```
CALL-151
B8E7:D4
B8F1:D5
B8FC:D6
B935:D7
B938:18 60
B955:A5
B95F:96
B96A:BF
B991:9A
B994:18 60
```

A CATALOG of the BSW original should reveal the names of the files mentioned earlier. Notice the extra blank line at the top of the listing. This is where the "A"file's filename was printed. Since it is a tedious procedure to change DOS like this, it is convenient that we can place all these files into memory at the same time.

33) Load the files

```
BLOAD INIT,A$6300
BLOAD O$301,A$7301
BLOAD PASSWORD!,A$8000
LOAD UTILITY
```

UTILITY contains two lines that call a machine language subroutine. This routine is used by UTILITY to change the BSW DOS from non-standard to standard and back again depending on whether the BSW disk or the data disk is being accessed. UTILITY calls the routine through the now famous ampersand ("&") vector.

Happily, our slave DOS will be the same format for both the BSW softkey disk and our data disks. If we remove the "&"calls, UTILITY will work fine with our new unlocked BSW disk.

34) Prepare to edit line 12

HOME: POKE 33,33: LIST12

35) Using the ESC keys, and right arrow key, remove from this line the "&1"and "&Ø".
36) Remove the "&1"and "&Ø"from line 98.

LIST98

37) To save the files, we must switch back to standard DOS sector ID bytes. Insert the slave disk and type

```
CALL-151
B8E7:D5
B8F1:AA
B8FC:AD
B935:DE
B955:D5
B95F:AA
B96A:96
B991:DE
```

38) We can now save the files in standard format.

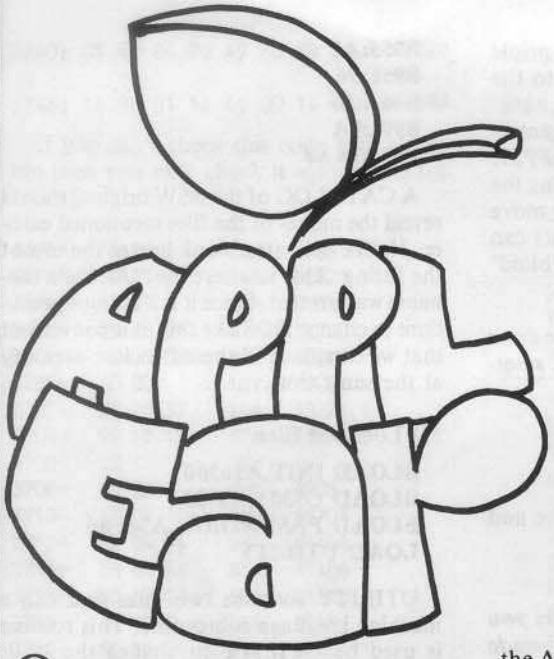
```
SAVE UTILITY
BSAVE INIT,A$6300,L$18
BSAVE O$301,A$7301,L$CF
BSAVE PASSWORD!,A$8000,L$180
```

We have finally obtained all the required files from the BSW original disk. The remaining tasks are:

1) Write a program to emulate the boot process.

2) Write two small machine language DOS patches.

Continued on page 22



VOICE SYNTHESIS and EXOTIC SOUND EFFECTS on your Apple are as easy as pie with ◄

By Ray Darrah

One of the barriers a computer encounters when dealing with the outside world is its need to process information in digital form. Unfortunately, information in the world is usually analog and cannot be directly handled by a computer. The information in an analog signal is usually transmitted as a voltage or current level, whereas the information in a digital signal is dependent upon the frequency of that signal.

Digital?! Analog!?

Since computers only understand digital information, it would be helpful if we could convert analog information to digital so the computer could act upon it. The process of converting from analog to digital generally involves measuring the amount of current (or voltage) present at a point in time and then converting it into a binary number. This binary number is merely a string of Ø's and 1's which can be stored in memory.

Unfortunately, the only thing built-in to the Apple that even comes close to performing the process of analog to digital conversion (and rather crudely I might add) are the cassette input jack and paddle button inputs.

The cassette input jack is hard-wired so that the status of the voltage (positive or negative) through it is mapped into bit 7 of location $C060. Therefore, if we were to write a program that sampled location $C060 and clicked the speaker everytime the msb (most significant bit or bit 7) changed state, we would produce a tone of the same frequency (but not of the same amplitude) as the tone being played (via tape player or amplifier) through the cassette input jack. Using this method we can roughly reproduce any sound (such as a human voice) that can be played into the cassette jack.

What About the Program?

ApplEar has the capability of playing back sounds (even at different speeds) that it has previously "heard." The user can tell ApplEar where in memory to store the recording or where in memory to start playing. In addition, a timing variable is used by ApplEar to more efficiently utilize the Apple's memory. Six versions of ApplEar have been provided to allow maximum ease when incorporating ApplEar into other programs.

How Does it Work?

ApplEar doesn't just repeatedly read location $C060 and store the status of the high bit in consecutive memory bits. Although this would provide the best sound reproduction, it would use up a great deal of memory. Instead, ApplEar reads the cassette jack and keeps a count of how long the high bit at $C060 stays set or clear. This number is then stored in the Apple's memory. With this technique, as much as ten times the amount of data (recorded material) can be stored in memory.

The bytes that are played and recorded are arranged as follows:

Bit 7 - The state of the cassette input jack at the time of sampling.

Bits Ø-6 - A number that is a multiple of the number of clock cycles (amount of time) that the input jack remained in this state.

Tell it What You Want

A string of values are passed to ApplEar that tell it everything it needs to know about the sound it has to deal with next. The location of these values in memory (currently on zero page) could be changed by altering the source code and reassembling it. The following is an explanation of various locations used by ApplEar.

$FC (252) - This location tells ApplEar from which page (block of 256 bytes) of memory to start playing or recording. To play something from the hi-res page 2, this location would hold a $40 (64 in decimal).

$FD (253) - This memory location specifies the ending page for playing or recording. To record something on the hires page 2, this location would hold a $8Ø (128 in decimal).
$FE (254) - Location $FE is used by two versions of ApplEar to indicate whether to record into or play from the above mentioned memory range. If the value in this location is greater than $7F (127 in decimal), ApplEar will record into those locations. Otherwise, it plays them through the speaker.
$FF (255) - This location holds the timing variable mentioned above. It is used by all versions of ApplEar1 but not any versions ofApplEar 2. Location $FF holds a number (from $00 through $40) that specifies how long to delay between samplings of $C060. Higher numbers produce worse sound quality and better memory usage whereas lower numbers produce better sound quality and worse memory usage.

Six Versions?

As mentioned earlier, there are six different versions of ApplEar printed at the end of this article. Three of them (ApplEar1, ApplEar1.P and ApplEar1.R) work exactly as stated above. However, the other three (ApplEar2, ApplEar2.P and Applear2.R) have *no* delay routine in them. Therefore, the ApplEar2 series provides higher quality sound but uses up memory at a faster rate. In addition, because there are no delay routines, the delay byte ($FF) is not used by the Applear2 series.

If there is nothing appended to the name of the program (ApplEar2 and ApplEar1) then that program contains BOTH the record and play routines. However, if a ".R"is appended to the name, then this is only the record routine. A ".P"at the end of the name indicates that it is only the play routine. If a ".R"or ".P"are attached to the name, then the direction byte ($FE) is not used.

Let's Get Typing

When typing in the hexdumps, use these parameters to save them:

BSAVE APPLEAR1, A$3ØØ, L$A2
BSAVE APPLEAR1.P, A$3ØØ, L$3C
BSAVE APPLEAR1.R, A$3ØØ, L$6B
BSAVE APPLEAR2, A$3ØØ, L$93
BSAVE APPLEAR2.P, A$3ØØ, L$37
BSAVE APPLEAR2.R, A$3ØØ,L$61

When recording, the computer first plays the sounds it hears through the speaker directly. This allows you to get the sound levels right and provides a pause before recording. When you are ready to record, simply press a key.

Remember to enter the proper values into locations $FC-$FF before starting the program with a

3ØØG

It is best not to set the end page byte ($FD) greater than $96 or else you may wipe out DOS which would prevent you from saving your wonderful recording. Also, you probably shouldn't set the beginning page byte ($FC) lower then $Ø8 because that is where the screen and other volatile memory exists.

ApplEar1

```
0300: A9 00 85 01 85 00 A5 FC   $47B5
0308: 85 02 24 FE 30 03 4C 8F   $FDFE
0310: 03 2C 60 C0 10 FB 8D 30   $65E7
0318: C0 2C 60 C0 30 FB 8D 30   $1C7F
0320: C0 2C 00 C0 10 EB 8D 10   $B53F
0328: C0 A9 00 F0 00 EA EA EA   $079F
0330: EA AD 60 C0 10 15 30 00   $76EB
0338: A2 80 E8 E0 FF F0 20 A4   $8C94
0340: FF 88 D0 FD 2C 60 C0 30   $3C27
0348: F1 10 14 EA A2 00 E8 E0   $25FB

0350: 7F F0 0C A4 FF 88 D0 FD   $56D3
0358: 2C 60 C0 10 F1 30 00 A4   $C494
0360: 00 8A 91 01 C8 84 00 D0   $8F0E
0368: C0 E6 02 A5 02 C5 FD 90   $EB28
0370: B8 60 29 80 C5 FB 85 FB   $6D83
0378: F0 05 8D 30 C0 D0 03 EA   $2D6C
0380: EA EA 8A 29 7F A4 FF 88   $12B3
0388: D0 FD AA EA CA D0 F3 A4   $5EC5
0390: 00 B1 01 AA C8 84 00 D0   $B369
0398: D9 E6 02 A4 02 C4 FD 90   $CBA4

03A0: D1 60                     $8553
```

ApplEar2

```
0300: A9 00 85 01 85 00 A5 FC   $47B5
0308: 85 02 24 FE 30 03 4C 80   $F2F9
0310: 03 2C 60 C0 10 FB 8D 30   $DAD0
0318: C0 2C 60 C0 30 FB 8D 30   $1378
0320: C0 2C 00 C0 10 EB 8D 10   $0A08
0328: C0 A9 00 F0 00 EA EA EA   $0898
0330: EA AD 60 C0 10 10 30 00   $DCD4
0338: A2 80 E8 E0 FF F0 16 2C   $79E9
0340: 60 C0 30 F6 10 0F EA A2   $812C
0348: 00 E8 E0 7F F0 07 2C 60   $0152

0350: C0 10 F6 30 00 A4 00 8A   $10CB
0358: 91 01 C8 84 00 D0 CA E6   $3E9B
0360: 02 A5 02 C5 FD 90 C2 60   $B534
0368: 29 80 C5 FB 85 FB F0 05   $CC49
0370: 8D 30 C0 D0 03 EA EA EA   $E4EB
0378: 8A 29 7F AA EA CA D0 F8   $EE42
0380: A4 00 B1 01 AA C8 84 00   $A897
0388: D0 DE E6 02 A4 02 C4 FD   $44C4
0390: 90 D6 60                  $9352
```

ApplEar1.P

```
0300: A9 00 85 01 85 00 A5 FC   $47B5
0308: 85 02 D0 1D 29 80 C5 FB   $AB76
0310: 85 FB F0 05 8D 30 C0 D0   $07BF
0318: 03 EA EA EA 8A 29 7F A4   $8843
0320: FF 88 D0 FD AA EA CA D0   $5E73
0328: F3 A4 00 B1 01 AA C8 84   $E29E
0330: 00 D0 D9 E6 02 A4 02 C4   $0EA6
0338: FD 90 D1 60               $6232
```

ApplEar2.P

```
0300: A9 00 85 01 85 00 A5 FC   $47B5
0308: 85 02 D0 18 29 80 C5 FB   $FA54
0310: 85 FB F0 05 8D 30 C0 D0   $06BD
0318: 03 EA EA EA 8A 29 7F AA   $D766
0320: EA CA D0 F8 A4 00 B1 01   $134E
0328: AA C8 84 00 D0 DE E6 02   $C9BA
0330: A4 02 C4 FD 90 D6 60      $C64C
```

ApplEar1.R

```
0300: A9 00 85 01 85 00 A5 FC   $47B5
0308: 85 02 2C 60 C0 10 FB 8D   $A5AC
0310: 30 C0 2C 60 C0 30 FB 8D   $9D55
0318: 30 C0 2C 00 C0 10 EB 8D   $6DCC
0320: 10 C0 A9 00 F0 00 EA EA   $108E
0328: EA EA AD 60 C0 10 15 30   $B1A7
0330: 00 A2 80 E8 E0 FF F0 20   $3C13
0338: A4 FF 88 D0 FD 2C 60 C0   $6575
0340: 30 F1 10 14 EA A2 00 E8   $7164
0348: E0 7F F0 0C A4 FF 88 D0   $F7F6

0350: FD 2C 60 C0 10 F1 30 00   $3C5B
0358: A4 00 8A 91 01 C8 84 00   $3C94
0360: D0 C0 E6 02 A5 02 C5 FD   $9A0C
0368: 90 B8 60                  $600A
```

ApplEar2.R

```
0300: A9 00 85 01 85 00 A5 FC   $47B5
0308: 85 02 2C 60 C0 10 FB 8D   $A5AC
0310: 30 C0 2C 60 C0 30 FB 8D   $9D55
0318: 30 C0 2C 00 C0 10 EB 8D   $6DCC
0320: 10 C0 A9 00 F0 00 EA EA   $108E
0328: EA EA AD 60 C0 10 10 30   $B9A2
0330: 00 A2 80 E8 E0 FF F0 16   $021D
0338: 2C 60 C0 30 F6 10 0F EA   $100F
0340: A2 00 E8 E0 7F F0 07 2C   $9F13
0348: 60 C0 10 F6 30 00 A4 00   $E350

0350: 8A 91 01 C8 84 00 D0 CA   $15BE
0358: E6 02 A5 02 C5 FD 90 C2   $690B
0360: 60                        $B9D9
```

ApplEar1 Source Code

```
1000 CASS.IN  .EQ $C060     CASSETTE INPU
T JACK
1010 IO       .EQ $C000     START OF APP
LE I/O
1020 STAT     .EQ $FB       LAST MSB STAT
US
1030 LEAST.PAGE .EQ $FC     START PLAYING
/RECORDING FROM HERE
1040 MOST.PAGE .EQ $FD      STOP PLAYING/
RECORDING WHEN YOU REACH HERE
1050 DIRECTION .EQ $FE      RECORD OR PLA
Y
1060 DELAY    .EQ $FF       TIMING VARIAB
LE
1070 INDEX    .EQ $0
1080 MEM.PTR  .EQ $1
1090
1100          .OR $0300
1110          .TF APPLEAR1
1120
1130 START    LDA #0        ALWAYS START
AT BEGINNING OF PAGE
1140          STA MEM.PTR
1150          STA INDEX
1160          LDA LEAST.PAGE FROM ADDRES
S
1170          STA MEM.PTR+1
1180          BIT DIRECTION PLAY OR RECO
RD?
```

Continued on page 20

Recently Developed Chips Promise New Life For Old Apples:

The 65SC802 & 65SC816

By Gary Peterson

Every time a new computer that uses a 16 bit (or 32 bit or 64 bit or etc...) microprocessor is introduced, I wonder how much longer I can continue to hang on to my Apple II plus which was designed before anyone had the faintest idea of what to do with a microcomputer. Of course, I could always upgrade to something more powerful, like the MacIntosh, but what am I supposed to do with all my Apple II software, donate it to the Goodwill? Perhaps I am more sentimental than most people, but for some reason I can't bear the thought of parting with the assemblers, word processors, utilities and games that I have acquired over the years. Somehow I doubt that I am the only Apple owner around who harbors such hangups.

"Every time a new computer that uses a 16 bit (or 32 bit or etc...) microprocessor is introduced, I wonder how much longer I can hang on to my Apple II Plus... Of course, I could always upgrade to something more powerful, like the MacIntosh, but what am I supposed to do with all my Apple II software? Donate it to the Goodwill?"

Thankfully, it looks as though a couple of new microprocessors just now becoming available may offer relief for those of us losing sleep over this Apple hardware/software dilemma. These new 16 bit microprocessors designed by William Mensch have the ability to execute software written for the older 8 bit 65Ø2's used in the Apple II and Apple /// series of computers (and Atari and Commodore among others).

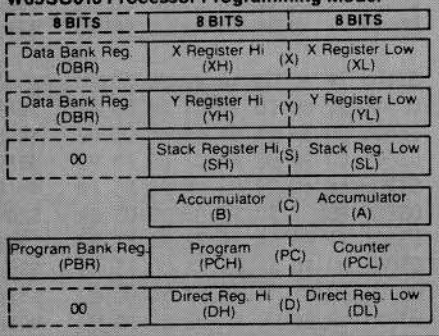
William Mensch was one of the original designers of the 65Ø2 at MOS Technology, a company he helped found. Mr. Mensch has since gone on to found another company, The Western Design Center, where he designed the 65CØ2 microprocessor being used in the Apple //c.

The 65CØ2 offers advantages over the original 65Ø2 such as low power consumption and expanded instruction set, but is still an 8 bit microprocessor. Mensch's latest microprocessors are also low power CMOS and have even more instructions along with the ability to perform 16 bit manipulations. One of the major attributes of these chips is their ability to execute code which was written for a normal 65Ø2. Western Design is marketing two different microprocessors which are 65Ø2 compatible; the 65SC8Ø2 and the 65SC816.

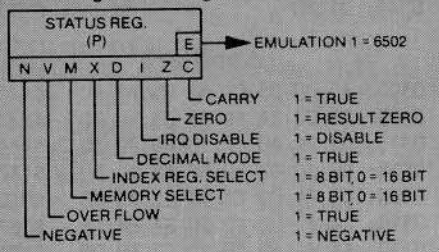
The 65SC8Ø2 is internally a 16 bit microprocessor which is, pin for pin, compatible with a 65Ø2. The 65Ø2's original address range of 65535 (16 bits) has been retained. This implies that a one chip swap on the Apple motherboard could give the computer 16 bit capabilities.

The 65SC816 is internally a 16 bit microprocessor, but it has an address range►

W65SC816 Processor Programming Model



Status Register Coding



Addressing Modes

Twenty-four addressing modes are available to the user of the W65SC816 family of microprocessors. The addressing modes are described in the following paragraphs.

1. Immediate Addressing [imm]
With immediate addressing the operand is contained in the second byte (second and third byte for 16 bit data) of the instruction.

2, 3. Absolute and Absolute Long Addressing [a], [al]
For absolute addressing the second byte of the instruction specifies the eight low order bits of the effective address while the third byte specifies the eight high order bits. For absolute long addressing the fourth byte specifies the bank address. The full 16.7 megabyte address space is addressed in the long mode. In the short mode the bank address is specified by the data bank register.

4. Direct Addressing [d]
Direct addressing allows for shorter code and execution times by only fetching a second byte of instruction. The second byte is added to the direct register (D) value. When the direct register low (DL) is zero fastest execution occurs. The bank address is always zero.

5. Accumulator Addressing [acc]
This form of addressing is represented with a one byte instruction and performs an operation on the accumulator(s).

6. Implied Addressing [imp]
In the implied addressing mode the address of the operand is implicitly stated in the operation code of the instruction.

7, 8. Direct Indirect Indexed and Direct Indirect Indexed Long Addressing [(d), y], [(dl), y]
This form of addressing is usually referred to as Indirect, Y. The second byte of the instruction is added to the direct register and points to a memory location in bank zero. The contents of this memory location and the byte following (the next byte is the bank address for the long mode) are added to the Y index register with the result being the effective address. For the short mode the bank address is specified by the data bank register. Note that when DL equals zero execution is fastest.

9. Direct Indexed Indirect Addressing [(d,x)]
With direct indexed indirect addressing (usually referred to as Indirect, X) the second byte of the instruction is added to the contents of the direct register and then adding the X register value. The result of these additions points to a memory location on bank zero whose contents is the low order byte of the effective address with the byte following the high byte of the effective address. The bank address of the effective address is specified by the data bank register.

10, 11. Direct Indexed with X and Direct Indexed with Y Addressing [d,x], [d,y]
Direct indexed with X usually referred to as Direct, X and direct indexed with Y usually referred to as Direct, Y are two byte instructions. The second byte is added to the direct register (D) and this result is added to the appropriate index register. The bank address is always zero. Execution is fastest when the low byte of the direct register (DL) is zero.

12, 13, 14. Absolute Indexed with X, Absolute Indexed Long with X, and Absolute Indexed with Y Addressing [a,x], [al,x], [a,y]
Absolute indexed addressing is used in conjunction with the X and Y index registers and is referred to as Absolute, X Absolute Long, X and Absolute, Y. The effective address is formed by adding the contents of the X or Y register to the second and third bytes of the instructions. The bank address is specified by the data bank register except in the long mode the fourth byte specifies the bank address.

15, 16. Program Counter Relative and Program Counter Relative Long Addressing [r], [rl]
Program counter relative addressing, usually referred to as relative and relative long addressing is used only with the branch instructions. The second byte is added to the program counter which for relative creates a +128 or -127 byte offset. The second and third bytes are added to the program counter to create +32768 or -32767 byte offset for the branch always long operation.

17. Absolute Indirect Addressing (Jump Instruction Only) [(a)]
The second and third bytes of the instruction contains the low and high order address bytes of a memory location located in bank zero. This memory location and the byte following contain the effective address which is loaded into the program counter. The destination bank address is specified by the program bank register except for the JML instruction the third byte fetched is the destination bank address.

18, 19. Direct Indirect and Direct Indirect Long Addressing [(d)], [(dl)]
In this form of addressing the second byte of the instruction is added to the direct register and the result points to a memory location in bank zero. The contents of this location and the following location (the next location is the bank address for the long mode) is the effective address. The bank address is specified by the data bank register for the direct indirect mode.

20. Absolute Indexed Indirect Addressing (Jump and Jump to Subroutine) [(a,x)]
With absolute indexed indirect addressing the second and third bytes of the instruction are added to the X index register contents. The result points to the low and (byte following) high order bytes which are loaded into the program counter. The bank address is specified by the program bank register.

21. Stack Addressing [s]
This addressing mode uses the stack register to address memory locations. The instructions which use the stack addressing include push, pull, interrupts, jump to subroutine, return from interrupt and return from subroutine. The bank address is always zero. Vectors are always pulled from bank 00. (See Compatibility Issues for 6502 Emulation).

22. Stack Relative Addressing [sr]
With stack relative addressing the second byte of the instruction is added to the stack register value. This effective address points to a data memory location on the stack. For 16 bit data the next location on the stack is the high byte of data. This addressing mode, in conjunction with using the push instructions, may be used to pass data to subroutines using the stack. The new TSC and TCS instructions provide fast stack modification. The direct register can be used for user stack functions. The bank register is always zero.

23. Stack Relative Indirect Indexed Addressing [(sr),y]
With stack relative indirect indexed with Y the second byte of the instruction is added to the stack register value. The address formed by this addition points to the low byte (the next location contains the high byte) of an indirect address. The Y register is added to this address to form the effective data address. This addressing mode, in conjunction with using the push effective address (PEA, PEI, PER) instructions, may be used to pass data addresses to subroutines using the stack. The new TSC and TCS instructions provide fast stack register modification. The direct register can be used for user stack functions. The data bank register is the bank address for the effective address.

24. Block Move Addressing [xyc]
This addressing mode is used for multiple byte moves forward (MVP) or backward (MVN). These three byte instructions use the X register for the source address, the Y register for the destination address and the C accumulator contains the number of bytes to be moved. The destination bank address is the second byte of the instruction with the source bank specified by the third byte. The data bank register is loaded with the destination bank value (second byte of the instruction).

of 16 megabytes (24 bits). Because the chip is still in a 40 pin package, the upper 8 bit of the address had to be multiplexed with the 8 bits of the data bus and is, therefore, not pin for pin compatible with a 6502.

All of the internal registers (A,X,Y, etc.) of these new chips are now 16 bit except the status register (8 bits) and program counter (24 bits). The chips have 11 new addressing modes and a new register, the Direct Register, which is used in conjunction with the new addressing modes. A variety of new instructions, such as block memory move and branch always (relative) have also been implemented. For low power requirements both the new microprocessors use CMOS technology. A new bit has been added to the status register to indicate whether the chip should emulate the 6502 or work in 16 bit mode.

Table 2. Instruction Set

W65SC816 Instructions (256 OP Codes)

A. The Original 6502 Instruction Set (151 Op Codes)

1.	ADC	Add Memory to Accumulator with Carry
2.	AND	"AND" Memory with Accumulator
3.	ASL	Shift Left One Bit (Memory or Accumulator)
4.	BCC	Branch on Carry Clear
5.	BCS	Branch on Carry Set
6.	BEQ	Branch on Result Zero
7.	BIT	Test Bits in Memory with Accumulator
8.	BMI	Branch on Result Minus
9.	BNE	Branch on Result Not Zero
10.	BPL	Branch on Result Plus
11.	BRK	Force Break
12.	BVC	Branch on Overflow Clear
13.	BVS	Branch on Overflow Set
14.	CLC	Clear Carry Flag
15.	CLD	Clear Decimal Mode
16.	CLI	Clear Interrupt Disable Bit
17.	CLV	Clear Overflow Flag
18.	CMP	Compare Memory and Accumulator
19.	CPX	Compare Memory and Index X
20.	CPY	Compare Memory and Index Y
21.	DEC	Decrement Memory by One
22.	DEX	Decrement Index X by One
23.	DEY	Decrement Index Y by One
24.	EOR	"Exclusive-or" Memory with Accumulator
25.	INC	Increment Memory by One
26.	INX	Increment Index X by One
27.	INY	Increment Index Y by One
28.	JMP	Jump to New Location
29.	JSR	Jump to New Location Saving Return Address
30.	LDA	Load Accumulator with Memory
31.	LDX	Load Index X with Memory
32.	LDY	Load Index Y with Memory
33.	LSR	Shift One Bit Right (Memory or Accumulator)
34.	NOP	No Operation
35.	ORA	"OR" Memory with Accumulator
36.	PHA	Push Accumulator on Stack
37.	PHP	Push Processor Status on Stack
38.	PLA	Pull Accumulator from Stack
39.	PLP	Pull Processor Status from Stack
40.	ROL	Rotate One Bit Left (Memory or Accumulator)
41.	ROR	Rotate One Bit Right (Memory of Accumulator)
42.	RTI	Return from Interrupt
43.	RTS	Return from Subroutine
44.	SBC	Subtract Memory from Accumulator with Borrow
45.	SEC	Set Carry Flag
46.	SED	Set Decimal Mode
47.	SEI	Set Interrupt Disable Status
48.	STA	Store Accumulator in Memory
49.	STX	Store Index X in Memory
50.	STY	Store Index Y in Memory
51.	TAX	Transfer Accumulator to Index X
52.	TAY	Transfer Accumulator to Index Y
53.	TSX	Transfer Stack Pointer to Index X
54.	TXA	Transfer Index X to Accumulator
55.	TXS	Transfer Index X to Stack Register
56.	TYA	Transfer Index Y to Accumulator

B. New W65SCXXX Instructions (13 Op Codes)

1.	BRA	Branch Relative always
2.	PLX	Pull X from Stack
3.	PLY	Pull Y from Stack
4.	PHX	Push X on Stack
5.	PHY	Push Y on Stack
6.	STZ	Store Zero in Memory (Direct, Direct, X; Abs; Abs, X)
7.	TRB	Test and Reset Memory Bits Determined by Accumulator A (Direct and Absolute)
8.	TSB	Test and Set Memory Bits Determined by Accumulator A (Direct and Absolute).

C. New W65SCXXX Addressing Modes (14 Op Codes)

2.	BIT	Test Bits in Memory with Accumulator (Direct, X; Absolute, X; Immediate)
2.	DEC	Decrement (Accumulator)
3.	Group I	Instructions (Direct Indirect (8 Op Codes))
4.	INC	Increment (Accumulator)
5.	JMP	Jump to New Location (Absolute Indexed Indirect)

D. Group I Instructions with New Addressing Modes (48 Op Codes)

- Direct Indirect Long Indexed with Y (8 Op Codes)
- Direct Indirect Long (8 Op Codes)
- Absolute Long and Absolute Long Indexed with X (16 Op Codes)
- Stack Relative (8 Op Codes)
- Stack Relative Indirect Indexed Y, (8 Op Codes)

1.	ADC	Add Memory to Accumulator with Carry
2.	AND	"AND" Memory with Accumulator
3.	CMP	Compare Memory and Accumulator
4.	EOR	"Exclusive-or" Memory with Accumulator
5.	LDA	Load Accumulator with Memory
6.	ORA	"Or" Memory with Accumulator
7.	SBC	Subtract Memory from Accumulator with Borrow
8.	STA	Store Accumulator in Memory

E. New Push and Pull Instructions (7 Op Codes)

1.	PEA	Push Effective Absolute Address or Immediate Data Word on Stack
2.	PEI	Push Effective Indirect Address or Direct Data Word on Stack
3.	PER	Push Effective Program Counter Relative Indirect Address or Program Counter Relative Data Word on Stack
4.	PLB	Pull Data Bank Register from Stack
5.	PLD	Pull Direct Register from Stack
6.	PHB	Push Data Bank Register on Stack
7.	PHD	Push Direct Register on Stack
8.	PHK	Push Program Bank Register on stack

F. Status Register Instructions (2 Op Codes)

1.	REP	Reset Status Bits Defined by Immediate Byte 1 = Reset 0 = Do not change
2.	SEP	Set Status Bits Defined by Immediate Byte 1 = Set 0 = Do not change

G. New Register Transfer Instructions (8 Op Codes)

1.	TCD	Transfer C Accumulator to Direct Register D
2.	TDC	Transfer Direct Register D to C Accumulator
3.	TCS	Transfer C Accumulator to Stack Register
4.	TSC	Transfer Stack Register to Accumulator C
5.	TXY	Transfer X to Y
6.	TYX	Transfer Y to X
7.	XBA	Exchange B and A
8.	SCE	Exchange Carry Bit C with Emulation Bit E.

H. New Branch, Jump and Return Instructions (6 Op Codes)

1.	BRL	Branch Relative Long Always (16 Bit Relative—32768 to + 32767) (Addressing Mode)
2.	JML	Jump Indirect Long
3.	JMP	Jump Absolute Long
4.	JSL	Jump to Subroutine Long (Uses RTL for Return)
5.	JSR	Jump to Subroutine (Indexed Indirect)
6.	RTL	Return from Subroutine Long

I. New Block Move Instructions (2 Op Codes)

1.	MVN	Move Block from Source (X Addressed) to Destination (Y Addressed). Block Length Defined by C. X, Y are Incremented
2.	MVP	Move Block from Source (X Addressed) to Destination (Y Addressed). Block Length Defined by C. X, Y are Decremented

J. New Co-Processor Operations (1 Op Code)

1.	COP	Co-Processor Instruction with Associated COP Vector and ABORT Input Supports Co-Processing Function i.e., Floating Point Processors, etc

K. New System Control Instructions (3 Op Codes)

1.	STP	Stop-the-clock Instruction Stops the Oscillator Input (or 02 Input) During 02 1 This Mode Is Released When RES Goes to a Zero. System Initialization May Be Desired; However, if After RESET One Performed an RTI, Program Execution Begins With the Instruction Following the STP Op Code in Program Sequence.
2.	WAI	Wait for Interrupt Pulls RDY Low and Is Cleared by IRQ or NMI Active Input
3.	WDM	There is One Reserved Op Code Defined as WDM Which Will Be Used For Future Systems. The W65SC816 Performs a No-Operation

On power-up, the emulation flag of the chip is set to a 1 so that it will act just like a normal 6502. When the emulation flag is set to 0 the extended capabilities of the microprocessor are enabled. This emulation flag actually occupies the same position in the status register as the carry bit. A new one byte instruction, SCE, exchanges the carry bit with the emulation bit. To enable the 16 bit capabilities of the chip, some code like the following would be required:

```
PHP ; Push Current Status Register on
      Stack
CLC ; Set Carry bit to 0
SCE ; Exchange Carry bit and Emulation
      bits
PLP ; Pull former Status Register from
      Stack
```

Some of the new instructions which have been implemented on the Western Design chips should make life easier for assembly language programmers. Forward and backward block memory moves are now available with the MVN and MVP instructions. Other useful new commands allow the swapping of the X and Y register (TXY and TYX), decrementing and incrementing the Accumulator (DEC and INC), relative branching always forward or backward 32767 bytes (BRL), pushing/pulling the X and Y registers to/from the stack (PLX,PLY,PHX,PHY) and storing zeroes in memory (STZ). Multiplication and division instructions are not available as they are on other microprocessors such as the 68000. This seems unfortunate, although the designer probably had good reason for it.

Many of the existing 6502 instructions have several new addressing modes available (the new chips have a total of 24 different addressing modes) and, combined with the new instructions, the result is that all 256 opcodes have been defined. There are no more of those unimplemented opcodes which cause unpredictable results when executed on a normal 6502.

It seems as though the impact of these two new chips could be quite substantial. It will be very interesting to see how the new chips will do in the marketplace against already established microprocessors such as the 68000 and the 8086.

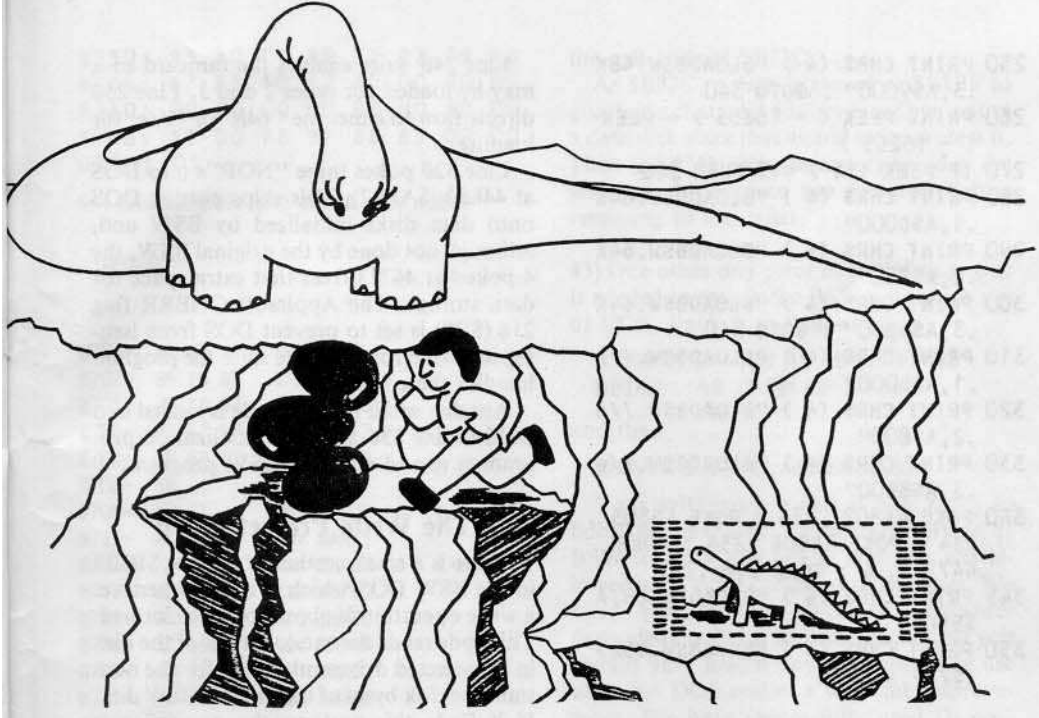
New computers based on the 65816 could offer MacIntosh-like capabilities on a computer capable of running all existing Apple II software. Because the chip is based on CMOS technology this computer could be transportable like the //c. The market for such a computer could be quite substantial.

Continued on page 23

The charts and diagrams accompanying this article are from the advanced data sheet on the 65816 and are reproduced with the kind permission of The Western Design Center.

Continued from page 17

```
1190            BMI .1          RECORD MUSIC
1200            JMP LOAD.IT     TOO FAR TO BR
ANCH
1210
1220 .1         BIT CASS.IN     PLAY BEFORE
1230            BPL .1          BEFORE RECORD
ING
1240            STA IO+$30      CLICK!
1250 .2         BIT CASS.IN     SO TO ADJUST
1260            BMI .2          SOUND LEVELS
1270            STA IO+$30      CLICK!
1280            BIT IO          UNTIL KEYPRES
S
1290            BPL .1
1300            STA IO+$10      RESET KEYLATC
H
1310
1320 RECORD     LDA #0          TIMING STUFF
1330            BEQ .1          ..ALWAYS
1340 .1         NOP
1350            NOP
1360            NOP
1370            NOP
1380            LDA CASS.IN     TEST
1390            BPL PLUS.COUNTER
1400            BMI MINUS.COUNTER TIMING
1410
1420 MINUS.COUNTER
1430            LDX #$80        BIT 7=1
1440 .1         INX             COUNTING
1450            CPX #$FF        FINISHED?
1460            BEQ STORE.IT    YES!
1470            LDY DELAY       WAIT A WHILE
1480 .2         DEY
1490            BNE .2
1500            BIT CASS.IN
1510            BMI .1          IF STILL MINU
S
1520            BPL STORE.IT
1530
1540 PLUS.COUNTER
1550            NOP             TIMING
1560            LDX #0          BIT 7=0
1570 .1         INX             COUNT THEM
1580            CPX #$7F        AS MANY AS CA
N HANDLE?
1590            BEQ STORE.IT    YES, BRANCH
1600            LDY DELAY       WAIT
1610 .2         DEY             AWHILE
1620            BNE .2
1630            BIT CASS.IN     STILL0?
1640            BPL .1          YES, BRANCH
1650            BMI STORE.IT    TIMING
1660
1670 STORE.IT
1680            LDY INDEX       WHERE TO STOR
E
1690            TXA
1700            STA (MEM.PTR),Y DUMMY LOCA
TION
1710            INY
1720            STY INDEX
1730            BNE RECORD      NO PAGE FLIP
1740            INC MEM.PTR+1   NEXT PAGE
1750            LDA MEM.PTR+1
1760            CMP MOST.PAGE
1770            BCC RECORD      DONE?
1780            RTS
1790
1800 PLAY       AND #$80        INTERESTED IN
 B7
1810            CMP STAT        SAME?
1820            STA STAT        FOR NEXT TIME
1830            BEQ NO.CLICK    SAME, DON'T C
LICK
1840            STA IO+$30      CLICK!
1850            BNE PLAY.LOOP   ..ALWAYS
1860 NO.CLICK   NOP             TIMING
1870            NOP
1880            NOP             TIMING STUFF
1890 PLAY.LOOP  TXA             START PLAYING
 HERE
1900            AND #$7F
1910            LDY DELAY       WAIT
1920 .1         DEY             FOR A WHILE
1930            BNE .1
1940            TAX             A VALUE
1950            NOP             TIMING
1960            DEX             DONE?
1970            BNE PLAY.LOOP   PLAY.LOOP KE
EP PLAYING
1980
1990 LOAD.IT    LDY INDEX       MEMORY OFFSET
T
2000            LDA (MEM.PTR),Y GET BYTE
2010            TAX             PUT IT IN X
2020            INY             NEXT BYTE
2030            STY INDEX
2040            BNE PLAY
2050            INC MEM.PTR+1   NEXT MSB
2060            LDY MEM.PTR+1   DON'T MESS W
ITH A
2070            CPY MOST.PAGE   DONE?
2080            BCC PLAY
2090            RTS
```



Dino Eggs By Microfun: A REVIEW

MicroLab, Inc.
2699 Skokie Valley Road
Highland Park, IL 60035
$40.00

Hardware Required:
Apple II, II plus, //e or compatible
One disk drive
Joystick

While visiting the pre-historic past, Time Master Tim (formerly a citizen of an average 2047 A.D. metropolis) accidentally infects the dinosaurs with 21st century measles. The dinosaurs are in danger of becoming prematurely extinct (thus changing history and the way all things are to be) if Tim doesn't devote his life to saving the entire dinosaur population by transporting their eggs and them (via time warp) to the 21st century.

Each of the ten levels of Dino Eggs consists of a randomly generated, four-ledged cliff. Along the ledges, boulders, wood and dinosaur eggs can be found. The boulders can be rolled down the cliff to smash hostile life forms.

At the beginning of every level, the first thing Tim must do is start a fire. This is accomplished by picking up a piece of wood and dropping it on another piece of wood. Failure to start a fire invites an attack from the Dino Mom.

Dino Mom has a large multicolored foot but the rest of her body is never seen. Mrs. Brontosaurus may not be the brightest creature around, but she does know when someone is trying to steal her offspring. In order to deter would-be dino-nappers, she violently stomps the ground in random locations. Tim must perform some quick evasive manuevering to avoid a Dino Mom attack. Since a single stomp from a 30 ton reptile is usually sufficient to flatten small humanoids, players are advised to keep a fire going at all times.

Dino Mom is not the only hazard Tim encounters in the Mesozoic era; snakes, spiders and centipedes are abundant. These creatures become more numerous and dangerous as the level of play advances.

Snakes can be avoided by jumping but centipedes, which travel overhead, must be ducked. Boulders can be used to smash these rather revolting life forms.

Snakes attack constantly throughout the games, but the centipedes only appear on levels 3 through 9. The spiders become considerably more nasty on these levels and periodically form a living trap for the young Dinos. If a baby wanders into one of them, they attempt to kidnap it.

To kill the spider, walk through its web strand when it is below you. The only way to save a baby dinosaur before a spider eliminates it is to cut the spider's strand while it has a hold of the baby.

The two things which will instantly destroy Tim are contact with the fire or a stomp from the Dino Mom. Contact with any of the other creatures starts a biological time bomb which can only be deactivated if Tim enters a time portal. Failure to enter a time portal initiates a unique process in which Tim de-evolves to the level of a spider.

Tim may either transport dinosaur eggs or hatched baby dinosaurs back to his era. Dinosaur eggs must be physically placed in the time portal in order to do this.

Although Time Master Tim doesn't have to carry baby Dinos to the time portal like he does the eggs, he must jump over them and press one of the joystick buttons while in mid-air. Sometimes this action doesn't register, so you might have to do it a couple of times. Once it takes effect, a time cage is placed around the baby who is now impervious to spider attacks. Furthermore, the baby can't hurt you or wander into fire like it loves to do.

The author of this game has done a very good job of shape-conflict processing. That is: objects destroy everything it is logical to destroy and leave unharmed everything else. For example, in the game you must often tumble boulders down the cliff to see what's underneath them. Usually there are eggs, wood or a power flower and sometimes there is nothing at all. As a boulder rolls down the hillside, it kills everything in its path. This works nicely for removing a couple of primordial nasties but has the side effect of wiping out baby reptilians and any fires it should land on.

In my opinion, Dino Eggs is one of the most enjoyable arcade games on the market today and I highly recommend it to anyone between the ages of three and three million.

Reviewed by Ray Darrah

APT's APT's APT's

Castle Wolfenstein
Contributed by
Eric Holman Whitaker

Hold up an SS Stormtrooper and search him. After you have confiscated his bullets (search him twice to be sure), you can leave the room without fear of the SS following you. Castle Wolfenstein trys to get the SS to follow you and then shoot you on the run.

If the SS gent is out of bullets, the game decides that he is not much good and does not send him trailing after you.

Continued from page 15

The Emulation Program

Many of the BSW files were relocated before they were saved to our slave disk. These files can be loaded back into position simply by specifying the Address parameter of the BLOAD command. However, there are four files that must be given special consideration. The BSW.48K.3 file is loaded by the BSW original between $9000 and $9BFF.

This would destroy our slave DOS's file buffer even if we set MAXFILES 1. A close examination of the file reveals that it contains no meaningful data past $9AA5 which is the bottom of the first DOS file buffer. When we saved the file in step 11, the Length parameter was shortened to allow us to BLOAD the file without overwriting the file buffer. For types 2 and 3, the same problem occurs with BSW.64K.3 and BSW.//e.3 and the same solution has been used.

The BSW.64K.2 and BSW.//e.2 files both need to be loaded in the range $800 to $1FFF. This is normally Applesoft program space and would conflict with our HELLO program. The solution is to cause DOS to load our HELLO program into the unused space starting at $2000 instead of the normal $800. Since our program will be fairly short, we can load it at $2000 for all three hardware types and not conflict with BSW.48K.2 at $2500.

39) After booting the slave disk, type in the following Applesoft program. When you have checked for and corrected any errors, save it.

SAVE HELLO

```
100 IF PEEK (104 ) < > 32 THEN
    POKE 8192 ,0 : POKE 104 ,32 :
    PRINT CHR$ (4 ) "RUNHELLO"
110 TEXT : HOME : NORMAL : IF PEEK
    (49152 ) = 155 THEN POKE 49168
    ,0 : POKE 104 ,8 : PRINT CHR$
    (4 ) " RUNUTILITY"
120 IF PEEK (49152 ) = 197 OR PEEK
    (49152 ) = 229 THEN POKE 49168
    ,0 : PRINT CHR$ (4 ) "FP"
130 POKE 49168 ,0 : PRINT CHR$ (4
    ) "MAXFILES1"
140 PRINT CHR$ (4 ) "BRUNTYPECHEC
    K,A$1300"
175 PRINT CHR$ (4 ) "BLOADBSW.LOG
    O,A$4000"
180 POKE 49234 ,0 : POKE 49237 ,0
    : POKE 49232 ,0 : POKE 49239
    ,0
190 PRINT CHR$ (4 ) "BLOADO$301,A
    $301"
200 PRINT CHR$ (4 ) "BLOADINIT,A$
    2D0"
210 PRINT CHR$ (4 ) "BLOADWRITEPR
    OTECT,A$B700"
220 IF PEEK (31 ) > 1 THEN 260
230 PRINT CHR$ (4 ) "BLOADBSW.48K
    .1,A$6000"
240 PRINT CHR$ (4 ) "BLOADBSW.48K
    .2,A$2500"
250 PRINT CHR$ (4 ) "BLOADBSW.48K
    .3,A$9000" : GOTO 340
260 PRINT PEEK ( - 16255 ) + PEEK
    ( - 16255 )
270 IF PEEK (31 ) = 3 THEN 310
280 PRINT CHR$ (4 ) "BLOADBSW.64K
    .1,A$D000"
290 PRINT CHR$ (4 ) "BLOADBSW.64K
    .2,A$800"
300 PRINT CHR$ (4 ) "BLOADBSW.64K
    .3,A$8B00" : GOTO 340
310 PRINT CHR$ (4 ) "BLOADBSW.//e
    .1,A$D000"
320 PRINT CHR$ (4 ) "BLOADBSW.//e
    .2,A$800"
330 PRINT CHR$ (4 ) "BLOADBSW.//e
    .3,A$8B00"
340 POKE 44802 ,234 : POKE 44803 ,
    234 : POKE 44804 ,234 : POKE
    4472 3 ,4 : POKE 216 ,255
345 PRINT CHR$ (4 ) "BLOADPATCH,A
    $BFC8"
350 PRINT CHR$ (4 ) "BRUNBSW.$525
    .$7FF,A$525"
```

This program does not create any variables. In particular, the CHR$(4) must be used repeatedly rather than defining a "D$ = CHR$(4)" since strings would be stored below HIMEM and conflict with files to be loaded there. The program length has been minimized by deleting unnecessary spaces in DOS commands since it must fit between $2000 and $24FF.

The HELLO program begins by determining where it has been loaded. Location 104 ($68) contains the high byte of the start of Applesoft program pointer which is normally $08. If this value is not 32 ($20) then the program pokes a zero at 8192 ($2000) and changes the pointer to point there. A DOS command to RUN HELLO then re-loads the program at $2000 and starts it running. Of course, the second time through, 104 will contain 32 and the program carries on.

Line 110 tests for the "ESC"key waiting at the keyboard and, if found, resets the start of Applesoft pointer to $0800 and RUNs UTILITY.

As a handy exit, Line 120 checks for upper or lower-case E at the keyboard and, if found, gives the DOS FP command to reset the pointer to $0800, clear the program and stop. This is convenient when you want to examine files on the disk; just hit "E'during the boot, then load whatever you wish to look at.

Line 130 clears the keyboard of any other keys and then sets MAXFILES 1. Line 140 runs TYPECHECK to determine the hardware configuration and lines 150 to 190 load the files common to all three hardware types. Line 160 sets the softswitches to display the hi-res page 2 logo. The WRITEPROTECT file in Line 190 is created below.

Line 200 checks location 31 ($1F) and skips to line 140 if type 2 or 3 has been determined by TYPECHECK. For type 1, lines 210 to 230 load the "48K"files and GOTO 320 to finish execution.

Line 240 write enables the ramcard so it may be loaded for types 2 and 3. Line 250 directs flow to either the "64K"or "//e"file loading.

Line 320 pokes three "NOP"s into DOS at 44802 ($AF02). This skips putting DOS onto data disks initialized by BSW and, although not done by the original BSW, the 4 poked at 44723 frees that extra space for data storage. The Applesoft ONERR flag 216 ($D8) is set to prevent DOS from issuing its own error messages since the program handles these itself.

Another small piece of code is loaded into DOS in line 330 and the initialization program is run to start the BSW program.

The Write Protect Patch

There is a small section of code at $B700 in the BSW DOS which is called whenever a write operation is about to be performed. This code reads the catalog track of the disk in the selected drive and checks for the non-standard disk bytes of the original BSW disk. If it finds this pattern, the carry flag is cleared, location $9AAC is loaded with a $4D and the routine returns to the caller. If the pattern is not found, the carry flag is set and the routine returns.

We originally thought this might be some kind of copy protection. It turned out to be a routine to prevent the user from writing to the original disk when he should not. Simply putting a write protect tab on the disk would provide similar protection, but this disk is likely to be used by children and it is easier to remove the write-protect tab than to change the format of the disk (as we have seen). In addition, the UTILITY program is used to customize BSW to the user's needs. It stores the initialization data on the BSW original disk and the write-protect tab would need to be removed for this operation.

It seemed logical to find a way to support this valuable feature. DOS maintains a buffer at $B3BB which contains the VTOC (Volume Table Of Contents) for the most recently read disk. This buffer is read from track $11, sector $0 on any standard DOS disk. It is also written to a disk when that disk is INITed.

Not all bytes in the VTOC are used by DOS. Before we INITed our slave disk, we changed one of these unused bytes ($B3BF) to an $A5. This byte will be used by our own routine at $B700 to determine if the disk about to be written is our softkey version of BSW.

Following is the code needed to implement this feature on the softkey disk.

40) Enter the monitor and type this code in

CALL-151

```
B700: A0 09 B9 EB B7 99 3D B7
B708: B9 47 B7 99 EB B7 88 10
B710: F1 AC C1 AA AD C2 AA 20
B718: B5 B7 B0 10 AC BF B3 C0
B720: A5 F0 03 38 B0 06 A9 4D
B728: 8D AC 9A 18 A9 00 8D BF
```

```
B730: B3 A0 09 B9 3D B7 99 EB
B738: B7 88 10 F7 60 00 00 00
B740: 00 00 00 00 00 00 00 00
B748: 11 00 FB B7 BB B3 00 00
B750: 01
```

41) Disassemble and check this code

B700L

```
B700-  A0 09      LDY #$09
B702-  B9 EB B7   LDA $B7EB,Y
B705-  99 3D B7   STA $B73D,Y
B708-  B9 47 B7   LDA $B747,Y
B70B-  99 EB B7   STA $B7EB,Y
B70E-  88         DEY
B70F-  10 F1      BPL $B702
B711-  AC C1 AA   LDY $AAC1
B714-  AD C2 AA   LDA $AAC2
B717-  20 B5 B7   JSR $B7B5
B71A-  B0 10      BCS $B72C
B71C-  AC BF B3   LDY $B3BF
B71F-  C0 A5      CPY #$A5
B721-  F0 03      BEQ $B726
B723-  38         SEC
B724-  B0 06      BCS $B72C
B726-  A9 4D      LDA #$4D
B728-  8D AC 9A   STA $9AAC
B72B-  18         CLC
B72C-  A9 00      LDA #$00
B72E-  8D BF B3   STA $B3BF
B731-  A0 09      LDY #$09
B733-  B9 3D B7   LDA $B73D,Y
B736-  99 EB B7   STA $B7EB,Y
B739-  88         DEY
B73A-  10 F7      BPL $B733
B73C-  60         RTS
```

B73D.B750

```
B73D- 00 00 00
B740- 00 00 00 00 00 00 00 00
B748- 11 00 FB B7 BB B3 00 00
B750- 01
```

42) Save the code

BSAVE WRITEPROTECT, A$B700,L$51

WRITEPROTECT uses the RWTS routine to read the VTOC from the disk into the VTOC buffer at $B3BB. In order to do this without disturbing BSW's use of RWTS, $B700 to $B710 save the current RWTS parameters into temporary storage at $B73D and replace them with the VTOC-reading parameters from the table at $B747. Then the RWTS is called to read the VTOC.

If an error occurs, it may well be because the disk is unformatted so we exit as 'OK-to-write' by branching to $B72C. If no error occurs, the value at $B3BF is compared to our constant ($A5). If our protected disk was in the drive, this byte would match and the branch to $B726 would be taken. Otherwise, it is safe to write the disk so the carry is set and we branch to $B72C.

At $B726, $4D is stored at $9AAC (like the original), the carry is cleared indicating 'Not-OK-to-write' and execution drops into the exit code at $B72C.

At $B72C, a zero is stored at $B3BF to guarantee that a $A5 is never written out to a data disk since that would write protect it. Then at $B731, the parameters saved at the start of the routine are restored before returning to the caller.

43) One other tiny piece of initializing code is needed to satisfy the BSW program's use of DOS. Enter it as follows

BFC8:20 DC AB A9 10 8D F0 B3
BFD0: A9 23 8D EF B3 60

and then

BSAVE PATCH,A$BFC8,L$14

Your softkey slave disk is now ready to boot! We have used this softkey version of BSW to write this article and that has allowed us to weed out many of the bugs. We were able to test all three hardware-dependent versions to some degree, so you should have few, if any problems. The use of a fast DOS makes a welcome improvement. We have successfully used Diversi-DOS but have not tried any of the others, although they should work just as well.

The tutorial on the back of the BSW master disk is not protected and may be copied with the normal copy methods, but for maximum protection we recommend that you mark the disk as a "master" so BSW cannot write to it. This can be done by using steps 1 to 5 of this softkey to INIT the disk on which you wish to copy the tutorial. Then, use the DOS 3.3 FID program to copy all the files from the original tutorial disk to the INITed back-up disk.

An alternative method is to use a disk-zap type program to modify the fifth byte of track $11, sector $0 to an $A5. This method can be used to protect other disks that might be accidentally written by someone using the BSW program.

As a final note, we would also like to offer a method to 'nibble copy' the BSW original disk. We are not sure if all the parameter changes and use of different programs is necessary but we have used this method and found it to be successful.

1) Track 0 is the hardest to copy; the track 0, sector 0 is imbedded in a long, non-standard data field. Use Copy II+ with the following parameter changes: 9=0, A=3, E=D5, F=B5, 10=D5, 31=0.

2) For tracks $1 to $19, use Locksmith 4.1 with these parameters: 44=A5, 45=96, 46=BF. When tracks $1 to $19 have been written, go back and re-copy track $11.

3) For tracks $1A to $22, change the Locksmith 4.1 parameters to: 44=F6, 45=FA, 46=FB.

As mentioned earlier, the original seems to have been recorded at a slower than normal speed so you may have to slow down your drive to get a good copy with this method. On the other hand, once you have a functioning softkey version, why bother?

Checksums

```
100 - $76C4    250 - $1D1F
110 - $2F60    260 - $19F1
120 - $5BF9    270 - $DB34
130 - $094F    280 - $0682
140 - $F655    290 - $985A
175 - $93F3    300 - $419F
180 - $49E8    310 - $49ED
190 - $73E2    320 - $C5D4
200 - $8EE6    330 - $B641
210 - $F39F    340 - $4BB9
220 - $B617    345 - $2313
230 - $1979    350 - $71B1
240 - $7409
```

Continued from page 19

Add-on co-processor boards based on the 65816 should also become available for the Apple. Rumors are cheap, but Apple is said to have a 65SC816-based computer in the works.

Although the 6580 2 can't address any more memory than a normal 6502, its pin for pin compatibility with the 6502 offers a route by which an Apple owner can upgrade to a 16 bit computer by simply dropping in a new microprocessor. The upgraded Apple will work just like it always did until software switches the 6580 2 out of emulation mode.

Don't rush out to your computer dealer or electronic parts supplier just yet, however, to ask for one of these new chips. As of this writing (2nd week of June '84), fully functional samples of the chips are not yet available, although Western Design says they should be available by the end of June. The partially functional samples of the chips which are now available do not operate correctly in the emulation mode and could not be used to replace a regular 6502. Hopefully, by the time you read this, fully operational samples of the 65SC802 and 65SC816 will be available in single unit quantities for $95.00 each, directly from Western Design. Western Design's address is: **The Western Design Center, Inc., 2166 East Brown Road, Mesa AZ 85203, (602)-962-4545.**

The only Apple assembler currently available which supports the new 65802 and 65816 instructions is the ORCA/M Macro Assembler from Hayden Software. Early versions of this assembler did not support the 65816, so if you decide to purchase the assembler make sure it is the latest version.

Examining Protected Applesoft BASIC Programs

By Clay Harrell

Many protected programs are written in APPLESOFT. Of course, most publishers are sly enough to protect against breaking out of their program with CTRL C or RESET. Also, most protect against re-entering BASIC from the monitor by changing the typical BASIC re-entry point (at $3DØ) so that it points to disaster. Many programs also set the autorun flag at $D6 so if you manage to break out of the program into BASIC, anything you type will RUN their BASIC program. I will describe how to beat all these protection schemes, assuming you have an old style F8 monitor ROM or some means to enter the Apple's monitor at will.

Is It BASIC?

First, we must determine if the protected program is written in APPLESOFT. If after you boot the program a BASIC prompt appears, this is a good indication that at least parts of the program are written in BASIC. Furthermore, if the program prints a lot of text on the screen, or requires a good deal of user input it is likely that the program is written in BASIC. The reason for this is that printing text on the screen and inputing data from the keyboard is easily accomplished from BASIC using PRINT and INPUT statements. Doing this from Assembly language however, requires a great deal more work. Also, we should realize why a programmer uses Assembly language. The only real advantage to Assembly language is speed. If speed is not critical, most non-sadistic programmers will use BASIC.

With this in mind, take note of how the program runs and displays information on the screen. If it runs at about the same speed as the BASIC programs you have written, it is probably written in BASIC. Remember, Assembly language is considerably faster than BASIC in every respect.

Finally, read the package the program came in. The package may say what language it was written in. If it doesn't, a dead give away is in the hardware requirements. If the program requires APPLESOFT in ROM, then at least part of the program is undoubtedly written in APPLESOFT.

Viewing the Code

Now that you have figured out that your protected program is written in BASIC, it is time to LIST their code. The first step is to RESET into the monitor after the program has started running.

Now you can try to enter the immediate BASIC mode by typing:

3DØG

The code at $3DØ normally jumps to the BASIC warm start routine but, if the protection scheme is worth anything, this will not work.

Assuming that didn't work, reload the program and RESET into the monitor again. The next thing is to try typing 9D84G or 9DBFG. These are the DOS cold and warm start routines, respectively. If you are lucky enough to get a BASIC prompt, you have done well. Most of the time, however, you will not.

If in either case you succeed in getting a BASIC prompt, try LISTing the program or CATALOGing the disk. If anything you type starts the program running again, the protection has changed the RUN flag at $D6. So RESET into the monitor again.

The RUN flag is a zero page location (at $D6) which will automatically RUN the BASIC program in memory if $D6 contains a value of $8Ø or greater (128 or greater in decimal). This is easy to defeat after you have RESET into the monitor by typing:

D6:ØØ

This resets the RUN flag to normal. Now if 3DØG, 9D84G or 9DBFG previously rewarded you with a BASIC prompt, this will solve the problem of the program re-running when you type a command.

For debugging efforts, the RUN flag can be changed from within a BASIC program by issuing the code:

```
1Ø POKE 214,255
```

or by poking location 214 with anything greater than 127. From Assembly language, the code would most likely look like this:

```
800- A9 FF      LDA #$FF
802- 85 D6      STA $D6
```

or by loading a register with $8Ø or greater and storing it at $D6.

Now if 3DØG, 9D84G or 9DBFG did not produce a BASIC prompt, then the DOS in use is more elaborate. So re-load the program and RESET into the monitor after it is running.

Saving the Program

Now come the final steps in trying to examine a BASIC program. If you are using a ROM card in slot zero with an old style F8 monitor ROM to reset into the monitor, turn on the mother board ROMs and turn off the ROM card INTEGER ROMs by typing:

CØ81

Now RESET the RUN flag to normal, just to be sure. Type:

D6:ØØ

Finally, enter APPLESOFT the sure fire way

"First we must determine if the protected program is written in APPLESOFT..." "...read the package the program came in. It may say what language it was written in. If it doesn't, a dead giveaway is in the hardware requirements. If the program requires APPLESOFT in ROM, then at least part of it is undoubtedly written in APPLESOFT."

by typing:

CTRL C

You should see an APPLESOFT prompt. Now type:

LIST

and your APPLESOFT program should now appear.

Strategic Simulation's RDOS

Applying this to a real world example, try this method with one of Strategic Simulation's (SSI) releases. SSI uses a highly modified DOS called RDOS for their protection. SSI uses all the tricks mentioned to prevent you from LISTing their programs but, using the above procedure, you can LIST their BASIC programs.

In addition, the DOS used by SSI (RDOS) uses the ampersand (&) in all of its DOS commands. So if you see any ampersands from within their BASIC program, you know it is a DOS command. For example, to catalog a SSI disk, after you follow the above procedure and you are in BASIC, type:

&CAT

This will display SSI's catalog. Very different, eh!

Well, back at the ranch. If you want to SAVE your a protected APPLESOFT program to a normal DOS disk, follow these steps:

1) RESET into the monitor after the program is running.

2) If you are using a ROM card in slot zero, disable its ROMs by typing:

C081

3) Next turn off the autorun flag and move page $08 to page $95 where it will be safe

D6:00
9500<800.8FFM

4) Check where the APPLESOFT program ends by typing:

AF.B0

5) Write down the two bytes that are displayed.

6) Boot a 48K normal DOS 3.3 slave disk with no HELLO program.

PR#6

7) Enter the monitor by typing:

CALL-151

8) Restore the APPLESOFT program by typing:

800<9500.95FFM

BCD: enter the two bytes you recorded in step 4, separated by spaces.

9) Enter BASIC and save the program by typing:

3D0G
SAVE PROGNAME

What you have done is to move $800 to $8FF out of the way so you can boot a slave disk. After normal DOS is up, you restore $800 to $8FF from $9500 to $95FF, and then restore the end of APPLESOFT program pointers so DOS knows how big your BASIC program is. Finally, you just SAVEd it to your disk!

Of course there are other more automated ways of getting programs to a normal DOS 3.3 disk (such as Demuffin Plus), but this is a quick and dirty method that will often work. Keep in mind that the program may not run from normal DOS because of secondary protection within the BASIC program itself. Any curious CALLs, POKEs or PEEKs to memory above 40192 (this is memory where DOS resides) or below 256 (zero page memory) should be examined closely.

Another thing to keep in mind is that the protected disk may have more than one file on it. If the protected DOS's commands have been modified you may not be able to CATALOG the disk directly to see how many files are on it. If this is the case, you can try to execute the CATALOG command handler at $A56E. To do this, boot up with the protected disk and RESET into the monitor. Then type:

A56EG

With any luck at all, the disk's directory should be displayed. If there is more than one file you will probably want to transfer all the files to normal DOS with the use of a program such as DeMuffin Plus.

If you experiment with the techinques I have described in this article I think you will be surprised at the number of programs whose copy protection can be fairly easily removed.

ADVENTURE TIPS ADVENTURE TIPS

Cranston Manor

Sierra On-Line

How do you lure a hungry mouse to a trap?

No fireflies in that dark cave? Better check out the General Store.

Don't try "open sesame" on the vault. Read the sign.

Zork I

Infocom, Inc.

Only sticky fingers can open a lock, or an egg without breaking it.

Where would YOU hide if you were a dungeon door? In the floor, of course.

Santa may carry many things up the chimney, but you only need a light and one other item.

Is that a sword or a light saber when danger is near?

Planetfall

Infocom, Inc.

In the brig? Look for writing on the wall.

The emergency bulkhead will open on its own when needed.

The auto pilot in the pod needs driving lessons, so strap in or you'll be very dead.

Mission Asteroid

Sierra On-Line

If you were a General, how would you want your men to address you?

Fatties don't pass military physicals. If you want in, you've got to get in shape.

Phew! Someone needs Shower To Shower.

Time Zone

Sierra On-Line

If you see a pteradactyl, run for cover!

Those cavemen look hungry enough to eat a lagomorph.

Try using a rock to kill dinner for your pre-human friends.

Suspended

Infocom, Inc.

Waldo can help Iris, but first he needs to "extend" himself.

When she's in top condition, Iris can be a big help in reading those monitors.

Every Whiz Kid needs a power source.

If you want to find a specific piece of information, look in the index.

Critical Mass

Sirius Software

What's on the wall?

There's something in that envelope. Better read it.

That's a pretty old elevator. Get ready to jump!

Who likes to eat peanuts?

Zork II

Infocom, Inc.

*If you pass through the curtain of light, you will leave in the same direction as you entered the vault.

*The safe is to the south.

*Dogs are easier to control if they wear collars.

*"Babe Flathead" played baseball.

*Don't kick the bucket.

**Contributed by Cullen Johnson*

CRUNCHLIST II

By Ray Darrah

Crunchlist II is an updated version of Crunchlist which appeared in Hardcore #6. The original Crunchlist program was designed for saving disk space while creating EXECable text files. As a side effect, Crunchlist also saved time when creating and EXECing the text file that was made when using it.

Crunchlist saved disk space by LISTing line numbers (of a BASIC program) without unnecessary spaces.

The Weakness

Unfortunately, Crunchlist had one weakness: It could not list the line number associated with an expression. That is, it could list 1000 but not (999 + 1). This is because it used the same routines that the LIST command used to determine which line number(s) to list. Therefore, just like the LIST command, you couldn't use expressions when specifying line numbers.

The Revision

Crunchlist II works the same way Crunchlist did, except now you may use any legal BASIC expression to specify a line number. Because of this, you may no longer use a hyphen (-) to separate the starting and ending lines to list. If you do, Crunchlist II will just think it is a minus sign and try to evaluate the entire expression. Instead, you must use a comma to separate start and finish line numbers.

I think you'll find this new crunchlister even more versatile than the last!

Crunchlist II Hexdump

```
0300: A9 4C 8D F5 03 A9 10 8D   $2C5A
0308: F6 03 A9 03 8D F7 03 60   $BB3A
0310: 20 A4 03 20 1A D6 A5 9B   $BA48
0318: 48 A5 9C 48 20 B7 00 F0   $347D
0320: 06 20 BE DE 20 A4 03 68   $F9AD
0328: 85 9C 68 85 9B A5 50 05   $D76F
0330: 51 D0 06 A9 FF 85 50 85   $62EB
0338: 51 A0 01 B1 9B F0 36 C8   $3398
0340: B1 9B AA C8 B1 9B C5 51   $23DB
0348: D0 04 E4 50 F0 02 B0 25   $3442

0350: 84 85 20 24 ED A4 85 4C   $E174
0358: 5F 03 09 80 20 ED FD C8   $B07E
0360: B1 9B D0 12 20 8E FD A0   $AFAB
0368: 00 B1 9B AA C8 B1 9B 86   $371D
0370: 9B 85 9C D0 C4 60 10 E2   $7589
0378: E9 7F AA 84 85 A0 D0 84   $AF52
0380: 9D A0 CF 84 9E A0 FF CA   $48D0
0388: F0 07 20 2C D7 10 FB 30   $9999
0390: F6 20 2C D7 30 08 09 80   $D957
0398: 20 ED FD 4C 91 03 20 ED   $2830

03A0: FD 4C 55 03 20 7B DD 20   $603D
03A8: F2 EB A5 A0 85 51 A5 A1   $76E1
03B0: 85 50 60                  $A844
```

Crunchlist II Source Code

```
1000 ********************************
1010 *                              *
1020 *        CRUNCHLIST II         *
1030 *                              *
1040 *                              *
1050 *      BY RAY DARRAH III       *
1060 *                              *
1070 ********************************
1080
1090
1100
1110
1120
1130 *                              *
1140 * APPLESOFT ROUTINES/LOCATIONS *
1150 *                              *
1160
1170 FINDLIN  .EQ $D61A
1180 LINPRT   .EQ $ED24     ROUTINE THAT
PRINTS X,A IN DECIMAL
1190 TOKTABL  .EQ $D0D0     TABLE OF TOKE
NS SPELLED OUT
1200 GETCHR   .EQ $D72C     ROUTINE THAT
GETS NEXT CHARACTER OF COMMAND WORD
1210 FRM.EVAL .EQ $DD7B     EVALUATES AN
EXPRESSION AND PUTS IT IN THE FAC
1220 INT.CONV .EQ $EBF2     CONVERTS VALU
E IN FAC TO INTEGER AND STORES IT IN $A0
.$A1
1230 CHKCOM   .EQ $DEBE
1240
1250 *                              *
1260 *      MONITOR ROUTINES        *
1270 *                              *
1280
1290 COUT     .EQ $FDED     PRINT A AS AS
CII
1300 CROUT    .EQ $FD8E     PRINT A CARRI
AGE RETURN
1310
1320 *                              *
1330 *      ZERO PAGE LOCATONS      *
1340 *                              *
1350
1360 CHRGOT   .EQ $B7       RETRIEVE THE
LAST CHARACTER FROM BASIC
1370 CHRGET   .EQ $B1       GET A NEW BAS
IC CHARACTER
1380 LINNUM   .EQ $50       NUMBER OF LAS
T LINUMBER TO LIST
1390 LOWTR    .EQ $9B       POINTER TO CU
RRENT LINE THAT WE ARE LISTING
1400 FORPNT   .EQ $85       TEMPORARY STO
RAGE OF THE OFFSET FOR THIS LINE
1410 FAC      .EQ $9D       USED BY GETCH
R SHOULD POINT TO WHICH TOKEN YOUR GETTI
NG
1420
1430 *                              *
1440 *    PAGE THREE LOCATIONS      *
1450 *                              *
1460
1470 AMPER    .EQ $3F5      AMPERSAND JMP
 LOCATION
1480
1490 *                              *
1500 *      START OF PROGRAM        *
1510 *                              *
1520
1530          .OR $300      SQUEEZE IT IN
TO PAGE THREE
1540          .TF CRUNCHLIST.OBJ
1550          LDA #$4C      MAKE AMPERSAN
D JMP TO START
1560          STA AMPER
1570          LDA #START
1580          STA AMPER+1
1590          LDA /START
1600          STA AMPER+2
1610          RTS           HOOKUP COMPLE
TE
1620 START    JSR LINGET    SET LINNUM TO
 START OF RANGE
1630          JSR FINDLIN   POINT LOWTR T
O 1ST LINE
1640          LDA LOWTR     SAVE POINTER
1650          PHA
1660          LDA LOWTR+1
```

```
1670            PHA
1680            JSR CHRGOT    RANGE SPECIFI
ED?
1690            BEQ MAINLST   YES, LIST IT
1700            JSR CHKCOM    SKIP THE COMM
A
1710            JSR LINGET    SET LINNUM TO
 END RANGE
1720 MAINLST    PLA           MOVE LOWTR BA
CK
1730            STA LOWTR+1
1740            PLA
1750            STA LOWTR
1760            LDA LINNUM    IS THE END NU
MBER A NULL?
1770            ORA LINNUM+1
1780            BNE NXLST     IF NO, START
LISTING
1790            LDA #$FF      YES IT IS, SO
 SET THE END NUMBER TO 65535
1800            STA LINNUM
1810            STA LINNUM+1
1820 NXLST      LDY #1        LIST AN ENTI-
RE LINE
1830            LDA (LOWTR),Y IS HIGH BYTE
 OF LINK ZERO
1840            BEQ FINISHED YES, DONE LIS
TING
1850            INY           GET LINE NUMB
ER
1860            LDA (LOWTR),Y
1870            TAX
1880            INY
1890            LDA (LOWTR),Y
1900            CMP LINNUM+1
1910            BNE LSTD
1920            CPX LINNUM
1930            BEQ LST1LIN
1940 LSTD       BCS FINISHED
1950 LST1LIN    STY FORPNT
1960            JSR LINPRT
1970 LISTLOOP   LDY FORPNT
1980            JMP PRINT1
1990 SENDCHR    ORA #$80
2000            JSR COUT
2010 PRINT1     INY
2020            LDA (LOWTR),Y
2030            BNE TOKENCMP NOT END OF LI
NE SO SEE IF TOKEN
2040            JSR CROUT     EVERY LINE IS
 TERMINATED WITH A CARRIAGE RETURN
2050            LDY #0        AT END OF LIN
E TO GET NEXT LINK
2060            LDA (LOWTR),Y
2070            TAX
2080            INY
2090            LDA (LOWTR),Y
2100            STX LOWTR     POINT TO NEXT
 LINE
2110            STA LOWTR+1
2120            BNE NXLST     BRANCH IF NOT
 AT END OF PROGRAM
2130 FINISHED   RTS           COMPLETELY DO
NE LISTING SO RETURN
2140 TOKENCMP   BPL SENDCHR   IF NOT A TOKE
N, JUST PRINT IT
2150            SEC
2160            SBC #$7F      MAKE IT A IND
EX NUMBER FOR THE TABLE
2170            TAX
2180            STY FORPNT    SAVE LINE POI
INTER
2190            LDY #TOKTABL-$100
2200            STY FAC       POINT FAC TO
TABLE
2210            LDY /TOKTABL-$100
2220            STY FAC+1
```

Continued on page 30

News Of Note

Omega Microware Files For Bankruptcy

Recently we received several calls from readers asking us if we knew anything about the rumor that Locksmith's publisher, Omega Microware, had gone out of business. Omega was not answering its phone.

A call to the phone number listed in the Locksmith 5.0 manual was intercepted by a recording saying the phone had been disconnected and calls were being handled by 312-537-3528. We called that number quite a few times but there was never an answer, although occasionally there would be a busy signal. It seemed like a bad sign. We decided to call some other people in the publishing industry to see if they could enlighten us.

Central Point Software's sales manager had also heard the rumor about Omega but could not confirm it. S-C Software told us that they now had to get copies of Locksmith from a distributor in Illinois and not directly from Omega. A call to that distributor finally confirmed the rumor; Omega had filed for Bankruptcy under Chapter 7 (anyone out there speak legaleze?) and the marketing rights for Locksmith 5.0 had been acquired by Alpha Logic Business Systems of Woodstock, IL.

Alpha Logic will be marketing Locksmith 5.0 and publishing the Locksmith Newletter. Improved support for it has been promised. It is no secret that Omega's support for Locksmith had been marginal at best.

Alpha Logic plans to establish a 24-hour hotline in addition to posting parameters on the SOURCE and MCI. They also have promised to rewrite the Locksmith 5.0 manual and to release versions of Locksmith for the IBM PC and the MacIntosh.

Future editions of the newletter probably will be appearing more often than quarterly, although each will have fewer pages than the first issue (72 pages). If you have tried to order a subscription from Omega and the letter was returned, you will have to send it directly to Alpha Logic. Their address is:

Alpha Logic Business System
4119 N. Union Road
Woodstock, IL 60098

Omega's decision to declare bankruptcy must have come abruptly because the company's ads appeared in some June computer magazines. No doubt there is a very interesting story behind the demise of Omega.

Cracking Reward

Any of our readers who have access to an IBM PC or compatible might be interested in a challenge that has been put forward by Denver-based Defendisk Inc. They are offering a $10,000 reward to the first person to produce a duplicate of a disk for the IBM that has been protected by their Defendisk system.

Their literature claims that this system places a unique electronic "signature" on the disk that can be read and verified by software. Of course, protected programs will refuse to execute if the correct signature is not found. Apparently this signature is recorded so strongly that even reformatting the disk will not remove it.

This sounds like a variation of the nibble-count scheme which many of us have come to know and love. It seems like the routines which check the signature could be tracked down and removed by a diligent programmer, although with $10,000 at stake, Defendisk probably checks the disk several times and uses additional protection schemes.

If you want to get a copy of the competition disk call 1-800-522-1800, Ext. 720. The cost is $9.95 plus $2.65 for shipping and handling.

Dear Subscribers:

To help us evaluate our mailing system, please send a postcard with your name and address and the *date you received this magazine* (Hardcore COMPUTIST no. 10) to:

Hardcore COMPUTIST
Mailing Survey
P.O. Box 44549
Tacoma, WA 98444

Backing-Up Minit Man

By Clay Harrell

Penguin Software
830 4th Avenue
Geneva, IL 60134
$34.95

Requirements:
48K Apple with old (or modified) F8 monitor ROM
One disk drive with DOS 3.3
Two blank disks
Minit Man
Super IOB

Minit Man is a Choplifter type game from Penguin Software. The game plays well and has nice graphics, demos and title pages. Unfortunately, its slow loading is practically unbearable at times. My original intent was to make Minit Man load at the speed of light... But in order to do this I had to first remove the disk's copy protection. Minit Man proved to be a challenge on which I had to use some of my own programming to overcome Penguin's protection. First, lets talk about the protection used in Minit Man.

"My original intent was to make Minit Man load at the speed of light...But in order to do this I had to first remove the disk's copy protection."

You can protect a program by various means, or you can protect a disk full of programs with some sort of DOS modification. DOS modifications are the least successful of protection schemes, since someone always seems to find a way to copy all the files onto a normal DOS disk, eluding all the protection.

The classic program for dealing with modified DOSs is Demuffin Plus. It works much the same way as Apple's Muffin program works. Muffin was written to read files from a DOS 3.2 disk and then write them to a DOS 3.3 Disk. DeMuffin was a variation of Muffin, allowing the Hardcore 3.2 user to copy disks from DOS 3.3 to DOS 3.2. Demuffin Plus operates on the same principle, but uses whatever DOS is in memory to read the disk, and then writes out to an initialized DOS 3.3 disk. While this is a powerful utility, it only works with programs that are based on DOS and that have a CATALOG track with normal, or somewhat normal, files.

Now, with this tidbit of information in mind, how do we know that a disk uses a modified DOS? There are many hints, the foremost being the appearance of a BASIC prompt at the bottom of the screen during booting.

Some publishers have bypassed the routine that outputs the prompt, but you can still guess that there's a modified DOS present if the boot sounds like a normal DOS boot, but the disk won't copy with COPYA.

Minit Man's protection scheme falls into this category of modified DOS protection. Upon booting the disk, the normal DOS boot sounds are heard and a prompt appears at the bottom left of the screen. Instead of attempting to use Demuffin Plus, we are going to use a different approach to the deprotection of Minit Man, using the program "Super IOB."

The actual modification on the Minit Man disk is that the end of address and end of data markers have been changed on every track to "DA AA"instead of the usual "DE AA." In addition, the start of address markers on every other track have been changed to "D4 AA 96"instead of "D5 AA 96."

The Super IOB controller starts copying from track two (we do not want the protected DOS residing on tracks Ø through 1) of the protected disk and writes back to the same tracks of our normal DOS 3.3 Disk.

With the Super IOB controller in hand we can now start to deprotect Minit Man. Here are the first few steps:

1) Type in the following Super IOB controller and save it with the SAVE CONTROLLER program on page 11 of this issue (or just LOAD SUPER IOB and type in the controller)

(For information on delta characters (▲) see IMPORTANT note on page 9.)

Minit Man Controller

```
1000 REM MINIT MAN CONTROLLER
1010 TK = 2 :ST = 0 :LT = 35 :CD =
    WR
1020 T1 = TK : GOSUB 490 : GOSUB
    1110
1030 GOSUB 430 : GOSUB 100 :ST =
    ST + 1 : IF ST < DOS THEN 1030
1040 IF BF THEN 1060
1050 ST = 0 :TK = TK + 1 : GOSUB
    1110 : IF TK < LT THEN 1030
1060 POKE 47505 ,222 : POKE 47413
    ,222 : GOSUB 230 : GOSUB 490
    :TK = T1 :ST = 0
1070 GOSUB 430 : GOSUB 100 :ST =
    ST + 1 : IF ST < DOS THEN 1070
1080 ST = 0 :TK = TK + 1 : IF BF =
    0 AND TK < LT THEN 1070
1090 IF TK < LT THEN 1020
1100 HOME : PRINT : PRINT "DONE▲
    WITH▲COPY" : END
1110 POKE 47505 ,218 : POKE 47413
    ,218 : IF TK / 2 = INT (TK / 2
    ) THEN 230
1120 RESTORE : GOTO 190
63010 DATA 212 ,170 ,150
```

EXEC SAVE CONTROLLER

2) Copy Minit Man with the controller placed into Super IOB.

3) Boot a normal DOS 3.3 Disk (preferably with a fast DOS since the load is so slow on the original Minit Man) and initialize the disk with HELLO as the boot program. Then FID all the files to the freshly INITed disk.

INIT HELLO
BRUN FID

When the copy is done you will have the bulk of Minit Man deprotected. Now we must make the programs on the Minit Man disk compatible with DOS 3.3, since the protected DOS has some built in routines that normal DOS does not. (Notice you can now CATALOG the disk and see the files that make up Minit Man).

4) Clear any program in memory and get the HELLO program

FP
LOAD HELLO

5) Make the following changes:

1 HOME
5 HGR2: HGR
6 CALL 33Ø72

6) Save the program

SAVE HELLO

Now some other files on the disk need to be changed to complete the deprotection. Follow these steps:

7) Remove the file "PLAYGAME PROG"

DELETE PLAYGAME PROG

8) Clear any program in memory and type this short one

FP
1 PRINT CHR$(4); "EXEC PROG"

9) Save it

SAVE PLAYGAME PROG

1Ø) Replace the program "DEMOSETPROG"with the following:

DELETE DEMOSETPROG
FP
1 PRINT CHR$(4); "EXEC DEMO"
SAVE DEMOSETPROG

Now you must create some text files to load in the game and/or demo files. You may use a word processor that creates NORMAL text files or you may create an Applesoft program to create them.

Note: The filenames must be typed exactly as shown or you will get a ?FILE NOT FOUND error

11) Create the text file "PROG"that contains:

```
BLOAD DP.7.2.SHAPES
BLOAD DPBC,A8516
BLOAD DP7,A16384
BLOAD PACPICS,A2Ø48
BLOAD DP.7.2.ANNEX,A22574
```

```
BLOAD MEMSORT5C01,A$5C01
BLOAD PACMOVE,A$6000
BLOAD TRAINS,A14000
BLOAD DP.7.2.ANMX
CALL 23553
```

12) Now create the text file "DEMO"that contains:

```
BLOAD DP7.DEMO,A16384
BLOAD DPBC.DEMO,A8516
BLOAD DP.7.2.SHAPES.DEMO
BLOAD PACPICS.DEMO,A2048
BLOAD DP.7.2.ANNEX. DEMO,A22574
BLOAD MEMSORT5C01.DEMO,A$5C01
BLOAD PACMOVE.DEMO,A$6000
BLOAD TRAIN.DEMO,A14000
BLOAD DP.7.2.ANMX.DEMO
CALL 23553
```

Be sure to carefully enter the text files exactly as shown above. Have fun with lightning fast Minit Man!

Controller Checksums

```
1000 - $356B    1070 - $5C4D
1010 - $3764    1080 - $E4C6
1020 - $B2CC    1090 - $7942
1030 - $A4CD    1100 - $7EA2
1040 - $34F8    1110 - $C0B7
1050 - $7477    1120 - $CFA4
1060 - $5448   63010 - $58A1
```

Dear Readers:

We at Hardcore COMPUTIST apologize to all those who have been confused by the format and name changes our magazine has undergone in the past. To make it easier for you to identify the order of our publications and to facilitate acquiring back issues, see the following which lists all the magazines that have been published through June 1984.

	1981-82
Hardcore Computing 1	
Update 1.1- Newsletter	
Hardcore Computing 2	
Update 2.1- Newsletter	
Hardcore Computing 3	
Update 3.1- Newsletter	
Update 3.2- Newsletter	
	1983
CORE Vol.1 No.1- Graphics	April
Hardcore COMPUTIST 1	May
Hardcore COMPUTIST 2	June
CORE Vol.1 No.2- Utilities	October
Hardcore COMPUTIST 3	September
Hardcore COMPUTIST 4	November
CORE Vol.1 No.3- Games	December
	1984
Hardcore COMPUTIST 5	January
Hardcore COMPUTIST 6	February
Hardcore COMPUTIST 7 (Vol. 3 No. 3- green cover)	March
Hardcore COMPUTIST 8	April
Hardcore COMPUTIST 9	May
Hardcore COMPUTIST 10	June

Hardcore COMPUTIST has been numbered consecutively from January 1984 with the exception of the March issue. Please note that CORE is no longer printed as a separate magazine. It has been combined with Hardcore COMPUTIST and you now receive Hardcore COMPUTIST with a CORE section included in it each month.

For back issue information, see inside back cover.

Whiz Kid by Ray Darrah

Did you know that all 256 different byte values cannot be reliably read from an Apple's disk drive? This is because of a hardware weakness which is: ***A byte stored on the surface of the disk must have the high bit set and contain no more than one instance of two consecutive zero bits.*** This results in only 69 "valid bytes." Two of these bytes (D5 and AA) are reserved for marking the beginning and end of data and address fields. This leaves 67 bytes, but only 64 are needed for the type of encoding that is used. The bytes that don't have a pair of consecutive one bits in them (not counting bit 7) are also eliminated which leaves exactly 64 bytes that can be stored on disk. They are:

DOS 3.3 Legal Bytes

```
96 97 9A 9B 9D 9E 9F A6
A7 AB AC AD AE AF B2 B3
B4 B5 B6 B7 B9 BA BB BC
BD BE BF CB CD CE CF D3
D6 D7 D9 DA DB DC DD DE
DF E5 E6 E7 E9 EA EB EC
ED EE EF F2 F3 F4 F5 F6
F7 F9 FA FB FC FD FE FF
```

Writing A Sector

The method of storing all 256 values with only 64 different bytes is called 6 and 2 encoding. This means that 6 bits of the byte are stored in one byte and the other two are saved in another byte. Here is an illustration of this:

Input Bytes:	**Output Bytes:**
Ø) ABCDEFGH	Ø) ØØABCDEF
87) JKLMNPQR	87) ØØJKLMNP
174) STUVWXYZ	174) ØØSTUVWX
	341) ØØZYRQHG

The numbers before the parenthesis are the relative byte number within the string of input or output bytes. The letters in the example represent the individual bits within these bytes. Note that for every three input bytes there are four output bytes. This is because the output bytes have only six changeable bits. With a little math (256*4/3 = 341.33) we can see that a sector encoded like this would take up 342 bytes (byte Ø through 341).

After DOS has encoded the sector to be written, it takes the output bytes (called nibblized bytes), translates them and writes them on the disk surface. The translation is performed with the help of a table in DOS (at $BA29 through $BA68) which contains the valid disk bytes listed near the beginning of this article. A $ØØ is written to the disk as a $96, a $Ø1 is stored as a $97, a $Ø2 is stored as a $9A and so on up to a $3F which is stored as a $FF.

Reading a Sector

When reading a sector, this entire process is performed in an opposite manner. First of all, a byte is read from the disk. Next, it is translated through the use of a different table (at $BA96 through $BAFF). A $96 is translated to a $ØØ, a $97 becomes a $Ø1, a $9A becomes a $Ø2 and so on up to a $FF which becomes a $3F.

Now these 342 bytes have to be converted into the original 256 bytes they once were (called denibblization). This process is what you would get if you exchanged the "Input Bytes" and "Output Bytes" labels in the table above.

Confusing?

Sounds pretty confusing doesn't it? Actually, this is a slightly simplified version of what happens. In addition to all this, DOS Exclusive ORs (EORs) the bytes with themselves (flips bits in bytes) as they are read and written. This is called a checksum and is a way of checking the data's validity because it always starts with a zero when writing and it should always end with a zero when reading.

I suggest reading the above several times until it finally makes sense. For a more complete explanation of this process consult 'Beneath Apple DOS' by Don Worth and Peter Lechner.

Continued from page 27

```
2230            LDY #$FF
2240 SKPTK      DEX             COUNT TOKENS
VERSA X
2250            BEQ PRTOK
2260 TOKLP      JSR GETCHR
2270            BPL TOKLP
2280            BMI SKPTK
2290 PRTOK      JSR GETCHR      GET A CHARACT
ER OF THE TOKEN
2300            BMI TOKDONE
2310            ORA #$80
2320            JSR COUT
2330            JMP PRTOK
2340 TOKDONE    JSR COUT        SEND LAST CHA
RACTER OF TOKEN
2350            JMP LISTLOOP ;GO DO NEXT T
HING IN LINE
2360
2370 LINGET     JSR FRM.EVAL EVALUATE FORM
ULA
2380            JSR INT.CONV CONVERT IT TO
 INTEGER
2390            LDA $A0         MOVE INTEGER
VALUE
2400            STA LINNUM+1
2410            LDA $A1
2420            STA LINNUM
2430            RTS
2440
```

PIRATES - Get your own Pirate T-Shirt with the above flag on the chest! Send size & color, (Red or Gray), and $8.00 + $2.00 S/H to: Captain Clone, Box 183/HC, Granville, IL 61326

REMEMBER ELEPHANT FLOPPY DISKS

QUALITY GUARANTEED FOR A LIFETIME OF HEAVY DUTY USE

$20.00 BOX OF 10
WE PAY SHIPPING

5¼" SOFT SECTOR, SS/SD/, W/HUB RINGS

data byte

2361 TEE DRIVE
LAKE HAVASU CITY, AZ. 86403
(602) 855-1592

VISA

CHECKS AND MONEY ORDERS WELCOME
AZ. RES. ADD 5% SALES TAX

master charge

Apple][,][+, //e, Franklin users:

Do you have problems backing-up your copy-protected programs?

Do you lack parameters for your copy programs?

Are you looking for programs that you can AFFORD?

Are you hesitating to upgrade your equipment due to expensive prices quoted in other ads?

It's simple now. Just drop us a line.

Send $1.00 U.S. funds to:

Reliant
P.O. Box 33610
Sheungwan, Hong Kong

IMPORTANT: We have over 600 PC name-brand programs and various hardware offers. Programs @ U.S. $8.00/PC includes the disk and registered airmail handling.

BACKUP YOUR DISKS

EDD is the most powerful disk duplicator available for your Apple™ computer. Unlike the copycards, which only copy single load programs, EDD backs up your entire disk. EDD can back up more protected software than all other copy programs or copycards put together. Since EDD is automatic, you will no longer have to change parameters to duplicate most disks, although every parameter is fully documented in our extensive manual. We also provide updated EDD program lists.

$79.95

Runs on: 48K Apple II, II plus, IIe, or III (emulation mode) with 1 or 2, 3.3 drives

ESSENTIAL DATA DUPLICATOR III™

- EDD rarely needs parameter changing
- Automatically finds the beginning of each track
- Unlike any of the Copycards, EDD backs up the entire disk, not just what is in memory
- Accurately finds "auto-sync" bytes and their lengths
- Can copy ¼ and ¾ tracks

TO ORDER OR FOR MORE INFORMATION, CALL (707) 257-2420

UTILICO MICROWARE 3377 Solano Ave., Suite 352, Napa, CA 94558

Do you need BACK ISSUES?

Are you tired of typing in programs that are available on disk from Hardcore COMPUTIST's PROGRAM LIBRARY?

If you're reading Hardcore COMPUTIST for the first time, don't miss out on past issues.
And, take advantage of our Special Offer.

Please send me the back issues and/or library disks I have checked below:

Hardcore COMPUTIST #1	**$3.50**	☐
Hardcore COMPUTIST #2 *	**$3.50**	☐
Hardcore COMPUTIST #3 *	**$3.50**	☐
Hardcore COMPUTIST #4	**$3.50**	☐
Hardcore COMPUTIST #6	**$3.50**	☐
Hardcore COMPUTIST #7	**$3.50**	☐
Hardcore COMPUTIST #8	**$3.50**	☐
Hardcore COMPUTIST #9	**$3.50**	☐
CORE #1 Graphics	**$4.50**	☐
CORE #2 Utilities	**$4.50**	☐
CORE #3 Games	**$4.50**	☐

* **Limited supplies**

SPECIAL OFFER!

Order these magazine/disk combinations and SAVE.

CORE #1,
Hardcore COMPUTIST #1,
and Library Disk #1 all for only **$19.95** ☐
CORE #1,2,3 **$10.00** ☐

Name _____

Address _____

City _____ **St** _____ **Zip** _____

Country _____

VISA/MC _____ **Exp** _____

Signature _____

Send check or money order to:

Back Issues/Library Disk Offer
Hardcore COMPUTIST
P.O. Box 44549
Tacoma, WA 98444

Washington state residents add 7.8% sales tax. Foreign orders add 20% shipping and handling. U.S. funds only.

Library Disk #6 **$9.95** ☐
- Hardcore COMPUTIST #9:
  - Super IOB and related Controllers
  - CORE Word Search Generator
  - Sensible Speller Loader & Saver

Library Disk #5 **$9.95** ☐
- Hardcore COMPUTIST #7:
  - Corefiler
  - Disk Directory Designer
- Hardcore COMPUTIST #8:
  - Corefiler Formatter

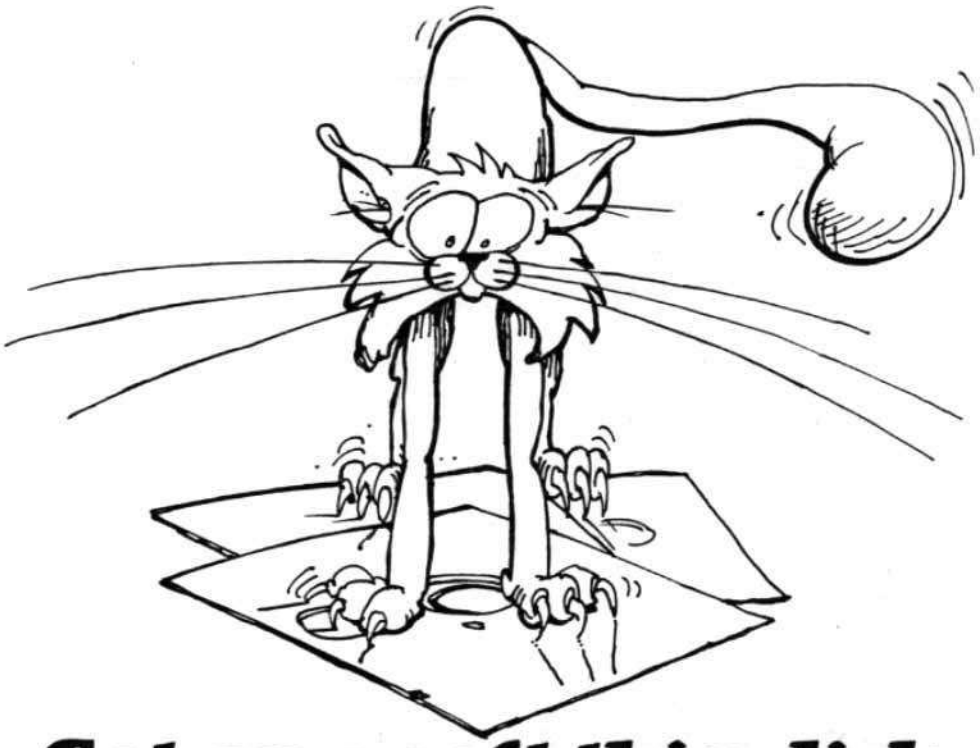
Library Disk #4 **$9.95** ☐
- Hardcore COMPUTIST #6:
  - Modified ROMs
  - Crunchlist
  - Crucial Code Finder

Library Disk #3 **$9.95** ☐
- CORE Games issue:
  - Destructive Forces
  - Dragon Dungeon

Library Disk #2 **$19.95** ☐
- CORE Utilities issue:
  - Hi-Res Utilities
  - Dynamic Menu
  - Line Find
  - GOTO Replace
  - GOTO Label
  - Fast Copy
- Hardcore COMPUTIST #3:
  - Map Maker
- Hardcore COMPUTIST #4:
  - Ultima II Character Generator

Library Disk #1 **$19.95** ☐
- CORE Graphics issue:
  - Scruncher
  - Quick Draw
  - QD.Editor
  - Design Plus
  - Faster Shapes
  - Space Raid
- Hardcore COMPUTIST #1:
  - Checksoft
  - Checkbin
- Hardcore COMPUTIST #2:
  - Page Flipper
  - String Plotter
  - Wall Draw

Disk Control **$15.00** ☐
- Disk Edit
- IOB
- Menu
- Disk View



Cat on a soft thin disk.

You need software insurance.

Diskettes are fragile, and when a protected program is damaged, the results are expensive and inconvenient. If you have a backup diskette, though, you can have your Apple, IBM or compatible computer back on line within seconds... affordably. That's software insurance.

Copy II Plus (Apple][,][Plus, //e)

This is the most widely used backup program for the Apple. Rated as "one of the best software buys of the year" by **InCider** magazine, its simple menu puts nearly every disk command at your fingertips. The manual, with more than 70 pages, describes protection schemes, and our **Backup Book™** lists simple instructions for backing up over 300 popular programs. A new version is now available that is easier to use and more powerful than before. Best of all, Copy II Plus is still only **$39.95.**

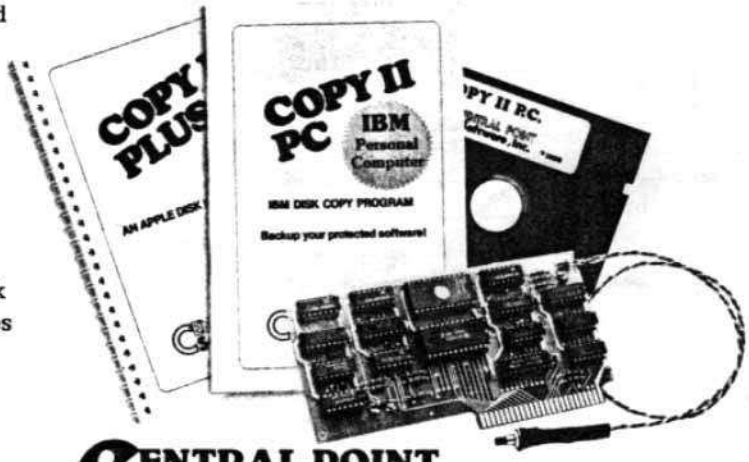
WildCard 2 (Apple][,][Plus, //e)

Designed by us and produced by Eastside Software, WildCard 2 is the easiest-to-use, most reliable card available. Making backups of your total load software can be as easy as pressing the button, inserting a blank disk and hitting the return key twice. WildCard 2 copies 48K, 64K and 128K software, and, unlike other cards, is always ready to go. No preloading software into the card or special, preformatted diskettes are required. Your backups can be run with or without the card in place and can be transferred to hard disks. **$139.95** complete.

Copy II PC (IBM)

This is **THE** disk backup program for the IBM PC and PC/XT that backs up almost anything. Others may make similar claims, but in reality, nothing out performs Copy II PC... at any price. Copy II PC even includes a disk speed check and is another "best buy" at only **$39.95.**

We are the backup professionals. Instead of diluting our efforts in creating a wide variety of programs, we specialize in offering the very best in backup products. So, protect your software investment, and get surefire relief from scratchy disks.



CENTRAL POINT Software, Inc.

The Backup Professionals

To order, call 503/244-5782, 8:00-5:30 Mon.-Fri., or send your order to: Central Point Software, 9700 SW Capitol Hwy, Suite 100, Portland, OR 97219. **Prepayment is required. Please include $2 for shipping and handling ($8** outside U.S. or Canada).