Softkeys For:

Threshold
Checkers v2.1
Microtype
COMPress Software
Uptown Trivia
Murder by the Dozen
Winter Games
The Windham Classics

Core:
The Animator

Feature:
Enhancing the Monitor
by Adding 65C02
Disassembly
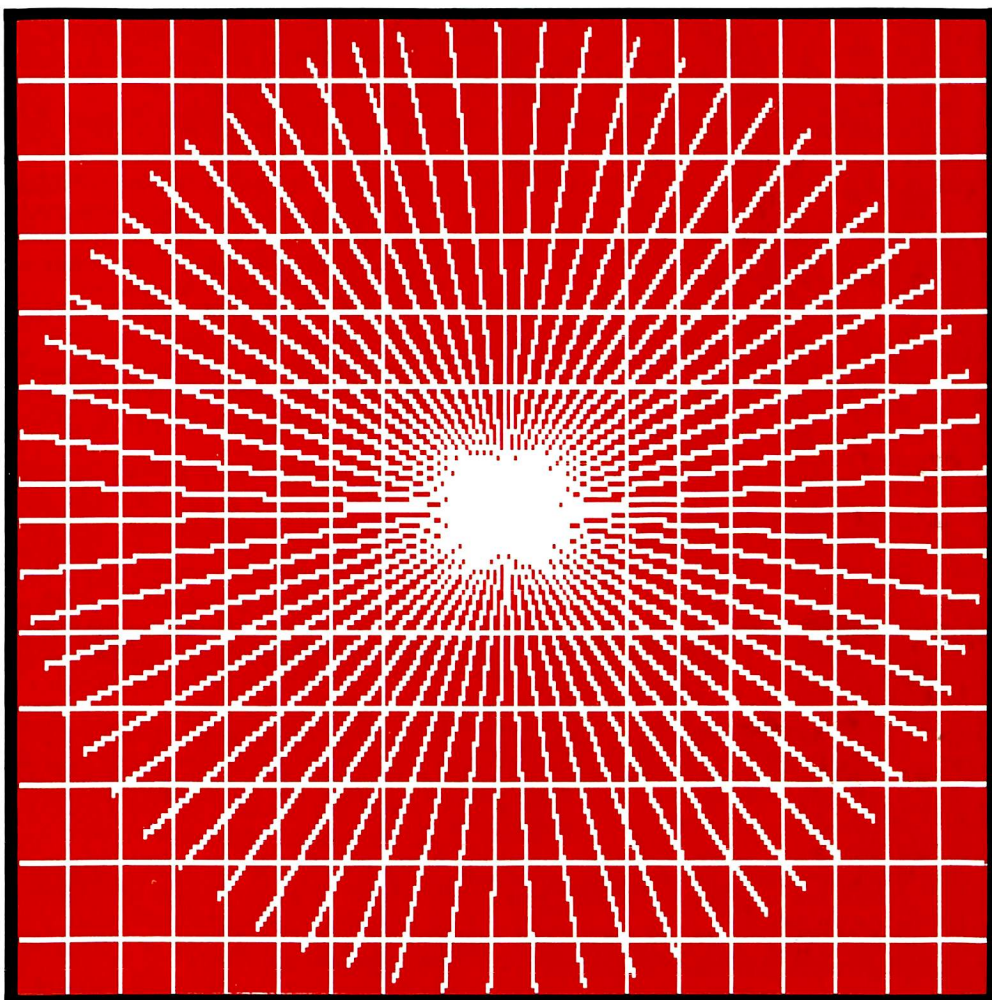
(Page 16)

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

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

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

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

PR#6

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

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

6⃞P

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

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

Software Recommendations: The following programs (or similar ones) are strongly recommended for readers who wish to obtain the most benefit from our articles:

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

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

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

RESET Into The Monitor: Some softkey procedures require that the user be able to enter the Apple's system monitor during the execution of a copy protected program. Check the following list to see what hardware you will need to obtain this ability.

Apple][Plus - Apple //e - Apple compatibles: 1) Place an Integer BASIC ROM card in one of the Apple slots. 2) Use a non-maskable interrupt (NMI) card such as Replay or Wildcard.

Apple][Plus - Apple compatibles: 1) Install an F8 ROM with a modified RESET vector on the computer's

motherboard as detailed in the "Modified ROM's" article of COMPUTIST No. 6 or the "Dual ROM's" article in COMPUTIST No. 19.

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

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

Keying In Applesoft Programs: BASIC programs are printed in COMPUTIST in a format that is designed to minimize errors for readers who key in these programs. To understand this format, you must first understand the formatted LIST feature of Applesoft.

An illustration- If you strike these keys:

10 HOME:REMCLEAR   SCREEN

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

10 HOME : REM CLEAR SCREEN

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

10 DATA 67,45,54,52

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

10 DATA   67,45,54,52

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

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

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

Keying In Hexdumps: Machine language programs are printed in COMPUTIST as both source code and hexdumps. Only one of these formats need be keyed in to get a machine language program. Hexdumps are the shortest and easiest format to type in.

To key in hexdumps, you must first enter the monitor:

CALL -151

Now key in the hexdump exactly as it appears in the magazine ignoring the four-digit checksum at the end of each line (a ''$'' and four digits). If you hear a beep,

you will know that you have typed something incorrectly and must retype that line.

When finished, return to BASIC with a:

E003G

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

Keying In Source Code The source code portion of a machine language program is provided only to better explain the program's operation. If you wish to key it in, you will need an assembler. The S-C Assembler is used to generate all source code printed in COMPUTIST. Without this assembler, you will have to translate pieces of the source code into something *your* assembler will understand. A table of S-C Assembler directives just for this purpose was printed in COMPUTIST No. 17. To translate source code, you will need to understand the directives of your assembler and convert the directives used in the source code listing to similar directives used by your assembler.

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

1) To compute CHECKSOFT checksums:

LOAD filename
BRUNCHECKSOFT

Get the checksums with

&

And correct the program where the checksums differ.

2) To compute CHECKBIN checksums:

CALL -151
BLOAD filename

Install CHECKBIN at an out of the way place

BRUN CHECKBIN,A$6000

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

xxx.xxx⃞Y

And correct the lines at which the checksums differ.

Coping with
COMPUTIST

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

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

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

renew your freedom

Check your mailing label to see if you need to renew your subscription. And if you think you might forget when that fatal time arrives, renew right now. Just use the order blank below.

if you're moving...

Let us know right away or at least 30 days in advance so that you won't miss a single issue. Just write your new address below, and include your present address label. Issues missed due to non-receipt of this Change-of-Address may be aquired at the regular back-issue rates. **Please remember, the Post Office does not forward third class mail unless requested.**

KNOW-DRIVE™

The smartest 128/512K card for the APPLE®

Memory Expansion: Increase desktop memory with **Appleworks**™ for the][+ or //e, or **Multiplan**™, or **VisiCalc**®, or **MagiCalc**™, etc. (Some programs require a preboot.) Optional piggy-back expansion (4-KD) to **KNOW-DRIVE 512K**.

Disk Simulation: **RAM WRITE PROTECT** switch. Software included for **KNOW-DOS**™ drive simulation of any Sl/Dr/Vol. Plus fast **ABCdos**™. Utilities include fast disk-RAM-disk or disk-disk copy.

Exclusive Options: **PLAY-BACK**™ captures **protected program memory** to normal disk. **BACK-to-BACK**™ adds the ability to have 3-11 protected programs in RAM at once; toggle execution instantaneously. **Interrupt/Modify/Save/Continue/Restore** at will.

Featuring −][,][+, //e and Franklin™ compatible. (Unsocketed //e RAM requires $20 adaptor.) Any slot, except //e AUX. Switch selectable hardware compatibility with 128K cards from Legend or Titan. 24 hour modem support. Two year warranty.

|		
KNOW-DRIVE 128K — $199	**KNOW-DRIVE 512K — $379**	
PLAY-BACK — $69	**BACK-to-BACK — $59**	**4-KD — $189**

Trademarks: KNOW-DRIVE, ABCdos, KNOW-DOS, PLAY-BACK, BACK-to-BACK - ABACUS Enterprises, Inc. / Appleworks, Apple - Apple Computer, Inc. / Franklin - Franklin Computer Corporation. / VisiCalc - Visicorp. / Multiplan - Microsoft Corp. / MagiCalc - Artsci. Price and specifications are subject to change without notice.   Revision D.

KD 128K w/PB w/BB $275
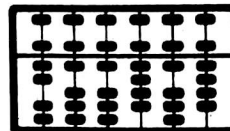
BULLSHIRT!

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

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

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

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

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

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

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

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

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

ADULT MENS: _____Small   _____Medium   _____Large   _____Xtra large

Name_____ ID#_____

Address_____

City_____ State_____ Zip_____

Country _____ Phone _____

💳 ⬛ _____-_____-_____-_____ Exp. _____

Signature_____CP29

Send check or money order to: Bullshirt; PO Box 110816-T; Tacoma, WA 98411. Washington orders add 7.8% sales tax. Foreign orders add 20% shipping and handling. US funds drawn on US bank.
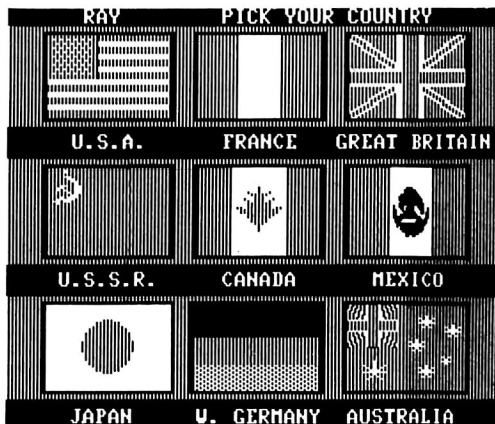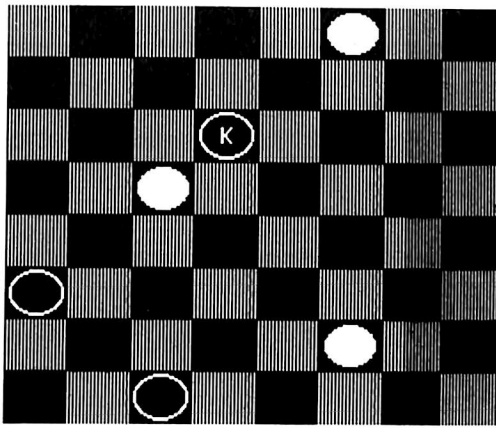
This month's cover:
A test screen used by Ray Darrah when Beta testing his
Animator program

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

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

Apple usually refers to the Apple][or][Plus Computer, and is a trademark of Apple Computers, Inc.

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

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

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

input

Space Adventure Backup

I've been a subscriber for a long time and enjoy your magazine very much.

I've owned ''Space Adventure'' by Sierra Software for several years now and have never progressed very far with it, as I have no backup to use and loathe using original disks. My original disk even came without a reinforced hub ring!

Anyway, using Copy II Plus v5.5 and two blank disks, I've been able to back it up. Now my original disk is safely stored away.

Step by Step

1) Boot the original ''Space Adventure'' disk.

2) Press ''P'' to enter dormant copy mode.

3) Use one blank disk to create dormant copy as instructed.

4) Boot Copy II Plus v5.5.

5) Select manual bit copy using default values except answer yes to synchronize tracks.

6) Copy original ''Space Adventure'' disk. Errors 5 and 6 are okay.

7) Using original disk again, re-boot and press ''P'' again.

8) Remove original disk and put away in a safe place. Replace with copied disk.

9) Select item #3 to activate dormant disk previously created.

10) The dormant disk becomes active and works as if it were the original.

I don't know why this works but it does!

Richard Ferri
Jamestown, NY

Wizardry APT Extended

This follow-up APT to the one for Wizardry on creating a ''Super Bishop'' will give you an even greater bishop. If you are unfortunate enough not to have Wizmaker, then here is what to do. After you have created a bishop, identified #9 and brought him up to the highest level, head into the maze. Once in the maze move around until you have an encounter, then cast either Haman, or Mahaman (this will dock you 1 level and some experience points), after this go back to camp and identify #9. You will now have another 200,000,000 experience points and your bishop can make level 400 or so. Then change class and you will have a super Fighter, Samurai, Lord or whatever you please. Do this 5 more times and your party will be more or less unbeatable!

As well, if any readers have figured out how to unjam the hatch that Eddie has jammed, then please send in an adventure tip for Hitchhiker's Guide to the Galaxy.

Rob Muscoby
Calgary, AB Canada

Mass Storage / Apple Gripes

I have an old program from Datamost called County Fair. It is an old arcade shooting gallery type game. I have never been able to copy or deprotect this game. Do you or any of your readers know how?

The new 3.5 inch drive from Apple says that it only supports ProDos and Pascal 1.3. Well Apple stepped in ''it'' again. ProDOS compatibility was expected. Hadn't heard of any Pascal newer than 1.2. What about CPM? I also had hoped to do some things like put the Print Shop, the Print Shop Companion, and all the graphics libraries all on one disk.

How about a patch to add to CPM to allow us to use Wordstar and it's many companion programs on the 3.5 inch format. I realize that with the catalog limitations on DOS 3.3 most would not want to use DOS 3.3 on the 3.5 inch disk, but there might be occasions where it

would be nice. For instance storing Visicalc files on the 3.5 inch disk would be very helpful.

Surely someone on your staff or one of your talented readers can help fill the gap left by Apple.

Apple has released and will continue to release some great stuff, but they always seem to fall just short of the ''best'' they could have done. Like a 3.5 inch disk that works with many operating systems instead of only two. How about a color MAC?

If Apple wants to be ''number one'', Apple will have to start putting out ''number one'' products. Not ''number three's'' that after two years the user groups ''fix'' or ''add to'' to the make number one products.

Doyle Hoover
Amarillo, TX

Mr. Hoover: We at COMPUTIST recognize the move toward mass storage in the Apple area of personal computing. In the future, we intend to concentrate more on softkeys that will break a program so far that it can be uploaded to hard disk.

A third party company (the name eludes me) is offering an upgrade for your Mac to add color graphics. This is exactly what you are saying about people ''fixing'' Apple's products a few years after their release. But then again, even a color Mac cannot possibly compete with the Amiga from Commodore.

Modified ROMs on the //e

Your answer to Mr. Keen, Jr., COMPUTIST No. 26, page 5 is right as far as it goes, but wrong in concept. I have incorporated all of Mr. Taylor's enhancements, plus a few of my own including the ability to disassemble all of the 65C02 Opcodes, and for a Mini-Assembler to assemble all the 65C02 Mnemonics. The answer to this is using the language card area, or using an integer firm card in a slot. This solves all the problems of implementing the dandy enhancements using the monitor, regarding //e's, but still leaves the problem of reset to the monitor. I have also used the integer card to use six 2732's, with one half using the enhanced F8 ROM with integer BASIC and above enhancements, and the other half with 3.3 DOS image for recovery of DOS. Following procedures will solve a couple other problems using either the original //e, or the Enhanced //e.

The following when accomplished will allow the following:

1) ⌧⌧ -Reset will re-boot without blasting two A0's into each page of memory.

input

2) ⬜🍎 -Reset will wait for another keypress:
   A. Space Bar will go to the monitor from any condition.
   B. Any other key will RUN the built in diagnostics routine.

Instructions for Patch to CD ROM (Unenhanced Version)

1) From Applesoft //e Call -151.

2) Make a patch at $300.

```
300:8D 07 C0
303:60
```

3) 300G

4) 1000:00 N 1001<1000.10FFM

5) 1100<C100.DFFFM

6) Enter the following code:

```
1242:30 0D AD 61 C0 10
1248:1B A9 00 8D F4 03 4C 64
1250:C2 2C 00 C0 10 FB AD 00
1258:C0 C9 A0 F0 03 4C 01 C4
1260:4C 69 FF FF
```

Disassembles as follows:

```
1242-   30 0D        BMI $1251
1244-   AD 61 C0     LDA $C061
1247-   10 1B        BPL $1264
1249-   A9 00        LDA #$00
124B-   8D F4 03     STA $03F4
124E-   4C 64 C2     JMP $C264
1251-   2C 00 C0     BIT $C000
1254-   10 FB        BPL $1251
1256-   AD 00 C0     LDA $C000
1259-   C9 A0        CMP #$A0
125B-   F0 03        BEQ $1260
125D-   4C 01 C4     JMP $C401
1260-   4C 69 FF     JMP $FF69
1263-   FF           ???
```

7) Save this new code

BSAVE PATCHED CD ORIGINAL ROM,A$1000,L$2000

8) The patched area is $C242.$6263. This image may now be burned into a 2764 EPROM.

Instructions for making patch to CD ROM (Enhanced Version.)

1) Except for entering the following code, the steps are the same as before. Enter the following code after completing step 5 from above:

```
12BE:10 02
12C0:80 0C AD 61 C0 10 1B A9
12C8:00 8D F4 03 80 14 2C 00
12D0:C0 10 FB AD 00 C0 C9 A0
12D8:F0 03 4C 00 C6 4C 69 FF
12E0:FF FF
```

Disassembles as follows:

```
12BE-   10 02        BPL $12C2
12C0-   80 0C        BRA $12CE
12C2-   AD 61 C0     LDA $C061
12C5-   10 1B        BPL $12E2
12C7-   A9 00        LDA #$00
12C9-   8D F4 03     STA $03F4
12CC-   80 14        BRA $12E2
12CE-   2C 00 C0     BIT $C000
12D1-   10 FB        BPL $12CE
12D3-   AD 00 C0     LDA $C000
12D6-   C9 A0        CMP #$A0
12D8-   F0 03        BEQ $12DD
12DA-   FC 00 C6     JMP $C600
12DD-   4C 69 FF     JMP $FF69
12E0-   FF           ???
12E1-   FF           ???
```

2) Save this new code

BSAVE PATCHED CD ROM ENHANCED,A$1000,L$2000

3) Patched area is $C2BE.$C2E1. This image may now be burned into a 2764 EPROM.

Donald Russell
Faribault, MN

Senior PROM Rhapsodies

I have subscribed to a multitude of magazines before, but I have never felt the need to write a letter until now. There is a piece of hardware available for the crackist that is so good and so reasonably priced that I feel all of your readers should know about it. This is the new Senior Prom for the //e and //c, which has been recently advertised in your magazine. This device is particularly important for the owner of a //c who has really been left out in the cold as far as many of the more common cracking techniques are concerned. I installed mine in 10 minutes, and I can now enter the monitor at will, move and save sections of memory, use NMI capabilities to save resident memory, and much more including a ROM resident sector editor that is available immediately upon power-on. This deceptively simple-looking device costs only $79.95 (Cutting Edge Enterprises, 317-743-4041) and comes with over 100 pages of documentation and cracking tutorials which are alone almost worth the price. I really cannot say enough in praise of this device.

By the way, all of the recent Mindscape releases (The Mist and so on, at least 4 or 5 programs) can be cracked very easily:

1) Boot System Master

2) CALL -151

3) B942:18

4) RUN COPYA and copy the disk (or you can use the disk copier in the Senior Prom with the checksum turned off!)

5) Use a sector editor to transfer track 0 sector 0 from any normal Pascal disk (I used Pascal v1.1) to track 0 sector 0 of the disk you just copied.

The program will now run and be copyable with any normal disk copier.

Joel B. Brenner, MD
Portland, OR

An Input Bug

I just received COMPUTIST No. 24 and was pleased to find my letter to you concerning protection. Evidently though I neglected to mail you the second page of my letter the first time. Following, please find the complete letter.

First let me say that this is the only Apple computer magazine worth buying that I know of! Keep up the good work.

Now, by using the 40 track init method by Yin H. Pun (Input by R. Boreiko, COMPUTIST No. 21, page 5) and a slightly modified VTOC mover by Rohn Smith (COMPUTIST No.21, pages 25 and 26) I have a copy protection method that makes a disk that none of my bit copiers can touch.

Here are the steps:

1) Boot Dos 3.3 or equal disk.

2) Type FP to clear memory.

3) Type POKE 44725.160

4) Type POKE 46063.40

5) Type POKE 48894.40

7) Insert disk and type INIT HELLO

8) Put all desired programs on disk

9) Insert disk with VTOC mover on it and BLOAD it

10) Enter monitor and type:

```
0325:27
033B:27
0345:27
3D0G
```

11) Insert 40 track disk and type CALL 769

12) Sector edit track 01 sector 09

13) Change address $01 from $11 to $27

You now have a disk that bit copiers cannot copy.

To move files from a normal disk to your protected disk do this:

input

1) Boot normal disk

2) LOAD or BLOAD program

3) Type POKE 44033.40

4) SAVE or BSAVE to protected disk.
   Change to a POKE 44033.17 to reverse this.
   If someone knows how to modify FID or COPYA to copy this whole disk, please let me know.

Thanks again for a great magazine

Wayne Watkins
Ridgeland, MS

RESET Confusion

I have a question on the "Reset into Monitor" modification you write about on page 2 of COMPUTIST No. 22. Could you please explain this in more detail. Are the instructions you are referring to for the Apple //c the ones listed in the ad on the back cover of issue number 22 under "Don Lancaster releases by Synergetics called "Absolute Reset (//c, new //e) for $19.50. If I purchase these instructions is the procedure difficult to do? Also I am aware that the modification voids the warranty which is already up in my case but do you know how it will affect my $100.00 "Apple care service contract" in any way?

Lastly, I would like to know what a "Modified F8 ROM capable of saving pages $0-$7" is listed in COMPUTIST No. 22 in "Softkey for Lode Runner" page 11. I really want to learn how to crack this disk. Is this the same as what I described above, i.e., the modification of getting into the monitor via the reset key?

I appreciate any help you could give me in this area. Thanks for your time and help concerning this matter.

Joseph Mariette
Riverside, CA

Mr. Mariette: The Don Lancaster method mentioned in the front of COMPUTIST is somewhat difficult to perform if you do not have an EPROM burner card. I don't think such a device exists for the //c. In this case, you will have to send a disk containing the code created by Don Lancaster's disk to someone who can burn EPROMs.

Even if you perform the Don Lancaster method, you will not have the ability to save pages $00 - $07 of memory as required by the Lode Runner article. For this ability, you will need to use the ROM modifications appearing COMPUTIST No. 6 or No. 19. However, neither of these modifications will work on the

//c.

For these and other reasons, we have changed the paragraph on the inside front cover to mention the Senior PROM rather than Don Lancaster's method.

Fifteen F-15's

This may sound corny to you but today was pretty exciting for me as two (!) COMPUTIST issues arrived-Numbers 23 and 24. I'd been waiting impatiently for their arrival assuming that some "good" projects awaited me in their pages and I wasn't to be disappointed. They are both fine issues.

About F-15 (your phone must be ringing off the hook by now, but, in case some folks are bashful about telling you...I will) - Microprose must have found out that they had a real winner in F-15 and decided to change the protection-at least twice. My copy of F-15 has a completely different protection scheme than the one Larry Jasonowitz had! But, his article was nicely written and informative.

My first hint at the different protection schemes was when I saw three sets of parameters listed on the July 1985 Parameter Disk for COPY II Plus 5.0. And I found out for sure when I tried Larry's controller-Super IOB 1.5 bombed when it tried to read track $22 on the original F-15 disk since it wasn't formatted.

One thing your readers will probably have to know-once they copy or deprotect later versions of F-15 is that entry codes are required to run the game once it boots up. The program will ask for a code based on a random number it will generate from 0 to 15. The required responses are:

Program:	You:	Program:	You:
0	G	8	M
1	E	9	P
2	A	10	J
3	M	11	C
4	C	12	E
5	D	13	C
6	F	14	P
7	E	15	M

Also, this game is quite complex and would be quite difficult trying to run without reading the manual-and it, too, is quite complicated. But the game has great entertainment value and that's what it's all about.

Bring on more COMPUTIST! Please?

Ken Burnell
Saudi Arabia

readers' softkey & copy exchange

Allan J. Migdal's Softkey for ...

The Windham Classics

Alice in Wonderland
Treasure Island
Wizard of Oz

Requirements:
A few blank disks
Pronto DOS or
   a sector editor COMPUTIST No. 27
FID or equivalent
A way to reset or NMI into the monitor

I had a problem with getting a reliable backup of several Windham Classics so I decided to take a closer look. The disks, although not standard DOS, appeared to have standard prologues and epilogues. I decided to try and use Super IOB 1.5 and its RWTS swap routine (COMPUTIST No. 22). By stopping the boot process just after tracks 0 and 1 are read, the RWTS routine can be saved. I used my Wildcard and a Trackstar to halt the loading after these tracks were read in. I then moved the RWTS to $1900 so it could be saved and used with Super IOB.

The whole disk needs to be copied because some program data is on track 2, which is normally used by DOS. However, the new copy will be CATALOGable and any single file can be accessed. If one uses Copy][Plus or similar to look at a map of the disk, the problem of copying the files to a normal disk becomes apparent. The adventures can, however, be run by typing BRUN STARTUP after booting a normal DOS.

In order to make a bootable version, one must copy the files to a disk with extra room on it. Pronto-DOS initializes disks with room set aside on track 2 (it's a bit smaller); or you can use the method outlined in Eric Ondler's softkey for Microzines (COMPUTIST No. 27, pg. 14, steps 1-4). You may wish to elimimate the last four sector edits in that case.

Anyway, here's the softkey...

1) Boot your Classic disk and break into the monitor after DOS is read in (the first couple of tracks). Listening to the drive can help.

2) Move the protected RWTS to a place safe from the boot.

1900<B800.BFFFM

3) Boot a slave disk and save the RWTS somewhere.

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

4) Load in Super IOB 1.5 and install the swap controller, making sure that the name of the RWTS is changed, and that tracks $0-$22 (0-34) will be copied.

5) Copy the disk with Super IOB.

6) Initialize a disk with extra room as described above, with the name "HELLO". Then DELETE HELLO.

7) Type in the following to put a new HELLO program on the disk.

NEW
10 PRINT CHR$(4)"BRUN STARTUP"
SAVE HELLO

8) Use FID to copy all the files from the Super IOB copy of your Classic onto your newly initialized disk.

9) Copy all other sides of your Classic with COPYA or equivalent.

Good luck!

_____

Phil Pattengale's softkey for...

Dave Winfield's Batter Up!

Avant-Garde Publishing
P.O. Box 30160
Eugene, OR 97403

Requirements:
COPYA
A sector editor
Two blank disk sides

When looking at this disk, I saw that it had a basic prompt as it booted up. This told me that it used some sort of modified DOS. Well, after a little investigation, I found that they really only changed a part of the DOS data epilogue. Since this is just a one byte change, from DE AA EB to DD AA EB, I first tried a simple COPYA copy with DOS's error checking killed (B942:18). But upon booting that copy, it had some of the graphics messed up, so I decided to do it with a little more accuracy.
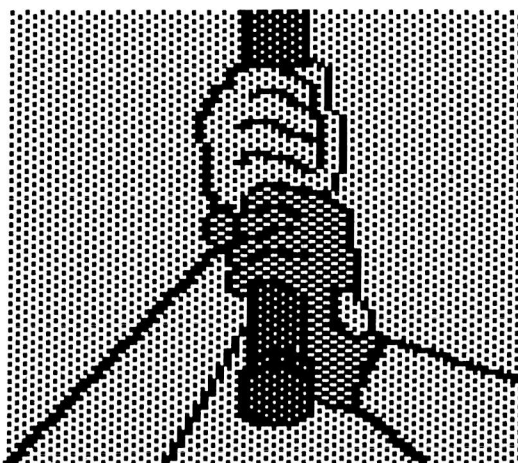

```
THE GRIP
--------

END GRIP:
--------
*GRIP BAT AT END

CHOKE GRIP:
----------
*GRIP BAT 2-4 IN.
 FROM END
```

*ALIGN KNUCKLES

*HOLD BAT FIRMLY WITH BOTTOM HAND
 AND MORE LOOSELY WITH UPPER HAND

**MAINTAIN CORRECT PRESSURE AND
 KNUCKLE ALIGNMENT

<-- TO GO BACK   <ESC> TO EXIT

readers' softkey & copy exchange

I initialized two disks, and then loaded COPYA. Next, I told DOS that the data epilogue was different (matching Batter Up!'s). Copying the disks now and initializing them (from COPYA) now would put the protected DOS on our new copy, but that would get us nowhere. So, we just use pre-formatted disks, and tell COPYA not to reinitialize them. And we've changed DOS itself so it can read the disks without any problem!

The only thing left to do is kill DOS's error checking on the disk in order to clean up any slightly loose ends (changes that don't prevent us from copying it the way we did, but won't let it boot). In order to get the CATALOG to work, we must also change a byte on the VTOC (Volume Table Of Contents) that they changed. This simply tells DOS that the catalog sectors start on Track $11.

1) Initialize 2 sides of a disk.

2) RUN COPYA from your system master disk.

3) After COPYA is loaded, hit ⌐C .

4) Make the following line changes in COPYA:

```
DEL 70
227 FT=1
```

5) Then make the following change to DOS:

```
CALL -151
B935:DD
3D0G
```

6) Use COPYA to copy both sides of Batter Up.

7) After you've made copies of both sides, boot up your favorite sector editor and make these changes to the front side, then make the second change to the back side of the copy.

Track	Sector	Byte#	From	To
0	3	$35	$DD	$DE
$11	0	$01	$00	$11

That's it! Enjoy!

─────────────────────────────── 🍎

Glen Tatum's softkey for...

Evelyn Wood's Dynamic Reader

Timeworks, Inc.

Requirements:
DOS 3.3
A copy program
Four blank disks
A way into the monitor

Evelyn Wood's Dynamic Reader is a three disk series designed to increase reading speed and comprehension. It is very challenging on high speeds. I have tried to copy the program with the usual arsenal of copy programs, and the copies do not boot, but they will run if the original is booted first. It appears that their DOS is tricky to copy.

Here is a simple way to get around that.

1) INIT a disk with DOS 3.3 and no HELLO program. Label it as "EVDOS Boot Disk" (or something).

```
FP
INIT HELLO
DELETE HELLO
```

2) Boot Dynamic Reader side A. Press ⌐C or Reset when the Applesoft prompt appears. If we try to CATALOG or LIST or anything now, the "ERROR #15" message appears. Now reset into the monitor. A56EG gives a valid catalog of this disk, so we must have some form of DOS here.

3) Move the RWTS of this DOS to a safe place.

```
4800<B800.BFFFM
```

4) Insert your disk from step 1 (labeled "EVDOS BOOT") and boot it.

```
C600G
```

5) Save the RWTS onto the disk.

```
BSAVE EVDOS,A$4800,L$800
```

6) Return to BASIC with ⌐C and type in this short program.

```
10 PRINT CHR$ (4 ) "BLOAD^ EVDOS,A$4800"
20 A$ = "B800<4800.4FFFM^ N^ D9C6G"
30 FOR A=1 TO LEN (A$ ) : POKE 511+A , ASC ( MID$
   (A$ ,A ,1 ) ) + 128 : NEXT
40 POKE 72 ,0 : CALL - 144
50 HOME : PRINT " INSERT^ DYNAMIC^ READER^ DISK^
   SIDE^ -A-^ NOW" : PRINT : PRINT "PRESS^
   SPACE^ BAR^ WHEN^ READY" ; : GET A$ : PRINT
60 IF A$ < > CHR$ (32 ) THEN 50
70 PRINT CHR$ (4 ) "RUN^ HELLO"
```

7) Save the program as HELLO.

```
SAVE HELLO
```

8) Now make a copy of Dynamic Reader sides A, B and C with whatever bit copy program you have. Or, if you use COPYA, modify DOS to ignore the errors it will find by typing the following before running COPYA:

```
CALL-151
B925:18 60
B988:18 60
```

```
BE48:18
3D0G
```

These disks will work just like the original as long as you boot your "EVDOS Boot" disk first.

─────────────────────────────── 🍎

Phil Pattengale's softkey for...

Jenny of the Prairie

Addison-Wesley Publishing Co.
Jacob Way
Reading, MA 08167

Requirements:
A blank disk
A 48K slave disk
Super IOB 1.5

Jenny of the Prairie is an animated adventure game designed to be played by young girls. The plot has it that Jenny, a young girl who was with a wagon train across the west, has wandered from the group, and must provide for herself over the winter. She gathers food, and builds a shelter. It is not an extensive adventure, but is suitable for younger children.

The only protection appears to be a modified RWTS. While the game is running, there isn't even a trap against pressing Reset. You can even hold ⌐C down during the boot and stop the Hello program before it starts. Once you're in control, just capture the RWTS and use Super IOB's swap controller. A new DOS and you're ready to go.

1) Boot the original disk and press Reset after the title page appears.

2) Enter the monitor and move the protected RWTS down to memory safe from the boot.

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

3) Boot a 48K slave disk with little or no HELLO program.

4) Save the protected RWTS to a disk with Super IOB on it.

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

5) Initialize a blank disk with the name "HELLO".

```
INIT HELLO
```

6) Put the Swap Controller below into Super IOB.

readers' softkey & copy exchange

bugs

7) Use Super IOB to copy tracks $3-$22 onto the disk you just initialized. Don't let Super IOB do the formatting. Finished!

controller

```
1000 REM JENNY OF PRAIRIE CONTROLLER
1010 TK = 3 : LT = 35 : ST = 15 : LS = 15 : CD = WR : FAST
     = 1
1020 GOSUB 360 : GOSUB 490 : GOSUB 610
1030 GOSUB 360 : GOSUB 490 : GOSUB 610 : IF PEEK
     (TRK ) = LT THEN 1050
1040 TK = PEEK (TRK ) : ST = PEEK (SCT ) : GOTO 1020
1050 HOME : PRINT "COPYDONE" : END
10010   PRINT   CHR$  (4 )   "BLOAD^
        RWTS. JENNY, A$1900"
```

Phil Pattengale's softkey for...

Learn About Sounds In Reading

American Educational Computer, Inc.

Requirements:
FID
4 blank disk sides

This is a fine educational program for children (recommended for grades K-3). Although it is protected, it is also very easy to deprotect. When this disk boots, there is a BASIC prompt (]) displayed. This means the program is using some sort of modified DOS. Well, in this case it was not too modified. In fact, if you boot a normal disk, defeat DOS's error checking and type CATALOG, the disk catalogs! This is the best kind!

Now, all we need to do is simply FID the files off the four disks to pre-formatted disks! Here is how to do it in cookbook fashion:

1) INITialize 4 disks, or 2 double-sided ones, with the name HELLO (preferably a fast DOS).

2) Kill DOS's error checking:

CALL-151
B942:18
☐C

3) BRUN FID (from your DOS 3.3 System Master), and copy all files from each disk to its corresponding blank.

BRUN FID

That's all there is to do!

Trevor Churchill's softkey for...

[Winter Games title screen graphic]

Epyx, Inc.
1043 Kiel Ct.
Sunnyvale, CA 94089

Requirements:
64K Apple][and up
Two disks or one double sided disk
COPYA
A sector editor

The amount of disk access in this sequel to Summer Games prompted me to make a backup. My nibble copiers found it impossible to copy track 0 so I set out to deprotect it. Using C.I.A. I found that it changed the end of address and data marks to DE FF, so I defeated the DOS error checking routine and copied it with COPYA. This produced a copy that kept rebooting after reading track 0. I knew that to reboot a disk there usually is a JMP to $C600 which I found on track 0, sector 5. I then changed this to a JMP $FF59 to put me in the monitor instead of re-booting. After I was in the monitor I found that it was a part of a routine at $BB00. I then used the CORE Disk Searcher to search the disk for a JSR $BB00 ($20 00 BB). I found it on Track 0, Sector 1, bytes $B, $C, and $D. Changing these to NOPs ($EA) produced a deprotected copy!

Cookbook Instructions

1) Get into the monitor.

CALL -151

2) Defeat DOS's error checking.

B942:18

3) RUN COPYA and copy both sides of the disk.

4) Sector edit track 0, sector 1 to eliminate the nibble count.

Track	Sector	Byte	From	To
0	1	$B	$20	$EA
0	1	$C	$00	$EA
0	1	$D	$BB	$EA

5) Let The Games Begin!

COMPUTIST No. 24:

Softkey for Alphabet Zoo:
The last line of the controller for this program was printed incorrectly. It should read as follows:

```
62010 DATA 222 ,171 ,237 ,170 ,170 ,213
      ,171 ,170 ,213 ,235
```

COMPUTIST No. 26:

Softkey for Gessler Spanish Software:
The Super IOB controller shown in the article (pg. 27) should be replaced with the following controller.

```
1000 REM  GESSLER CONTROLLER
1010 TK = 4 : LT = 35 : ST = 15 : LS = 15 : CD = WR
     : FAST = 1
1020 GOSUB 490 : GOSUB 270
1030 POKE 47405 ,24 : POKE 47406 ,96 : POKE
     47497 ,24 : POKE 47498 ,96
1040 GOSUB 610 : T1 = TK : TK = PEEK (TRK ) :
     RESTORE : GOSUB 310 : TK = T1
1050 GOSUB 490 : GOSUB 230 : RESTORE : GOSUB
     310
1060 POKE 47405 ,208 : POKE 47406 ,19 : POKE
     47497 ,208 : POKE 47498 ,185
1070 GOSUB 610 : IF PEEK (TRK ) = LT THEN 1090
1080 TK = PEEK (TRK ) : ST = PEEK (SCT ) : GOTO
     1020
1090 HOME : PRINT "COPYDONE" : END
5000 DATA 1^ CHANGES ,17 ,0 ,1 ,17
```

controller checksums

1000	- $356B	1060	- $2E89
1010	- $2744	1070	- $080F
1020	- $07EB	1080	- $DA02
1030	- $8288	1090	- $35F0
1040	- $E6E8	5000	- $0EAC
1050	- $B884		

Deprotecting More Stickybears:
The commands in step 14 of this article should read:

300:20 E3 03 4C D9 03
B7EB:00 tt ss
B7F0:00 60 00 00 02
300G

COMPUTIST No. 27:

War With the Daleks:
Exploring Artificial Intelligence
Due to circumstances beyond our control, the following BASIC line was omitted from page 28.

```
2550 DATA 13 ,14 ,25 ,3 ,4 ,25 ,-136 ,4 ,24
     ,25 ,-13 ,147 ,25 ,-4 ,224 ,25 ,-13
     ,14 ,215 ,-146 ,147 ,15 ,-24 ,15
     ,-136 ,137 ,-13 ,4 ,-24 ,225 ,-24 ,25
     ,-225 ,-36 ,47 ,-4 ,-36 ,15 ,-13 ,58
     ,-36 ,47 ,-47 ,58 ,-36 ,24 ,25 ,-13
     ,47 ,-47 ,15 ,-36 ,4 ,-24 ,-58
```

Antique softkey for...
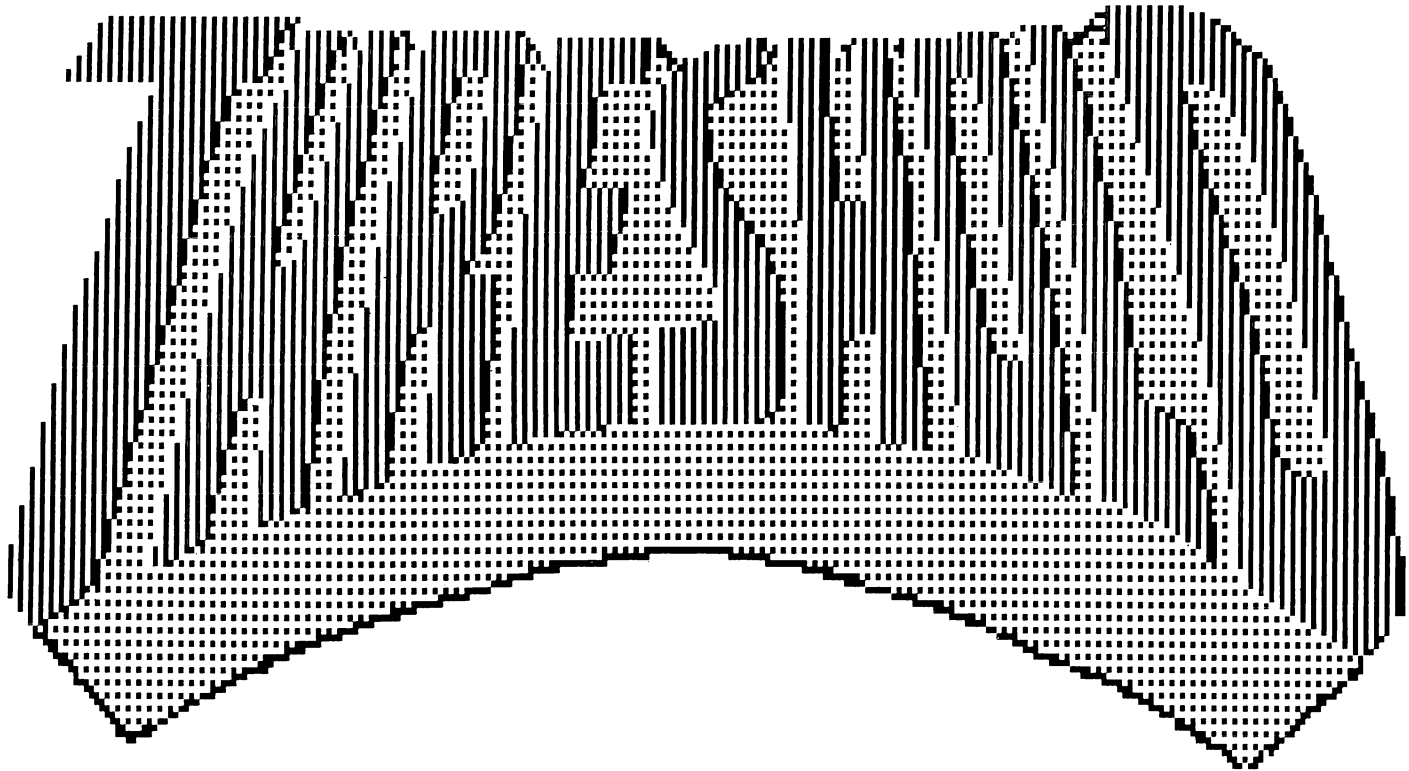
by Denny Colt

On-Line Systems

Requirements:
At least 48K
A blank disk
Sector editor (optional)
Super IOB 1.5
The 13 sector RWTS (DOS 3.2) in a binary file

Threshold is Sierra On-Line's answer to all those "zip along the bottom and blast the aliens" type games. The object is to destroy four waves of superbly animated cosmic scum and make it to the Mothership before the fuel runs out. After docking with Mother, the ship is refueled, and the cosmos is filled with four more waves of new scum. Care has to be taken when attacking, because if too many shots are fired, the laser overheats. If the action gets too hot and heavy, the ship is allowed one time warp which slows down the nasties for a few seconds.

The game is a nice piece of programming, but as a game it needs a little fine tuning. To do that, the protection needs to be removed.

Breaking In

I got out my trusty nibble editor and started to examine the disk. It seemed to be laid out in a standard 3.2 format except for track 1 which was filled with $FFs. That told me that I was working with a nonstandard boot (3.3/3.2), and a nibble count. I wanted to get rid of my problems one at a time, so I copied the disk using Copy][Plus, without the nibble counted track 1. That way I could work on removing the count without worrying about screwing up something else. I booted the copy and listened to it. After a few seconds I heard the disk arm slide over to track 1 and do its count. When it failed, the screen filled with inverse "@"'s and the computer hung up.

So I rebooted the disk. This time when I heard the slide, I pressed Reset to enter the monitor to look for the count code. I found it at $9600. I tried the easy way to remove a nibble count. I put a RTS at the start of the routine and wrote it back to the disk. **CRASH!!!**

The easy way never works for me! I examined the routine and found that mixed in the middle of the count was some code that was necessary for the program's mental health. It would have to be allowed to do the count.

Since I couldn't go in the front door, I tried to sneak in the back door. I found the end of

the nibble count, and made it loop around to the part of the code that was used if the count was okay. Scratch one nibble count.

The next problem was to convert the 3.2 format of the disk to a 3.3 format. I used the Super IOB swap controller on it, and tried to boot the disk. The computer hung. I boot code traced the original disk to find the problem. I found that it read in some code that in turn would read in a 13 sector controller, that in turn read in the DOS 3.2 second stage boot. The rest of the program was loaded by the second stage. To get around this odd boot, I simply wrote a normal 3.3 track 0 onto my copy, then wrote the Threshold 2nd stage boot controller (Track 0, Sector 1) on top of it. I booted the disk again, and it worked like a charm. The disk was cracked.

THE SOFTKEY

To see all this great theory in action follow these 2 steps.

1) INIT a blank disk with DOS 3.3.

2) Install the IOB controller at the end into Super IOB 1.5 (have DOS 3.2's RWTS ready) and use it to copy Threshold to the initialized disk. Answer "No" to the format option.

You may want to exit the room because it makes a lot of noise on track $14.

This controller includes a couple of patches to Super IOB 1.5 to make the fast copy routine and sector editor work with 13 sector disks.

You may wish to make a note of them.

The Fix

Now that the disk was cracked, I could make some improvements to Threshold. The first thing I wanted to add was a joystick mode. Although it supports paddles and the keyboard, Threshold loses it with a joystick. To enter the warp, a player has to take his/her/its hand off the joystick and press the space bar. This usually results in (a) the player dying because he let go of the joystick in the middle of a battle (Dummy!), or (b) the space bar being slapped through the bottom of the computer.

To save yourself the aggravation and damage, make the following sector edits. Change track $1C, sector $0C, starting at byte $CA, from $C9 A0 F0 67 C9 C7 F0 34 to $2C 62 C0 30 66 EA EA EA and track $1D, sector $01, byte $5C from $CB to $00. To enter the warp, press button 2 on the joystick. Hold it down to pause the game. *Note:* This takes out the keyboard mode.

These other changes will work in any mode.

To give yourself an infinite number of time warps change track $1C, sector $02, byte $BA, from $0E to $00.

To change the rate at which the laser heats up change track $1D, sector $00, byte $22, from $06 to any number from $01 to $30. A value of $01 being no heat, $30 being a bonfire.

And lastly, to prevent the game from resetting you back to the first board at the end of a game, change track $1D, sector $01, bytes $34 to $36, from $8D F5 65 to $EA EA EA. With this change in place you will now be reset to the first board after the last mothership docking you reached. (i.e. If you were on board 3, you would be reset to board 1. If you were on board 8, you would be reset to board 5.)
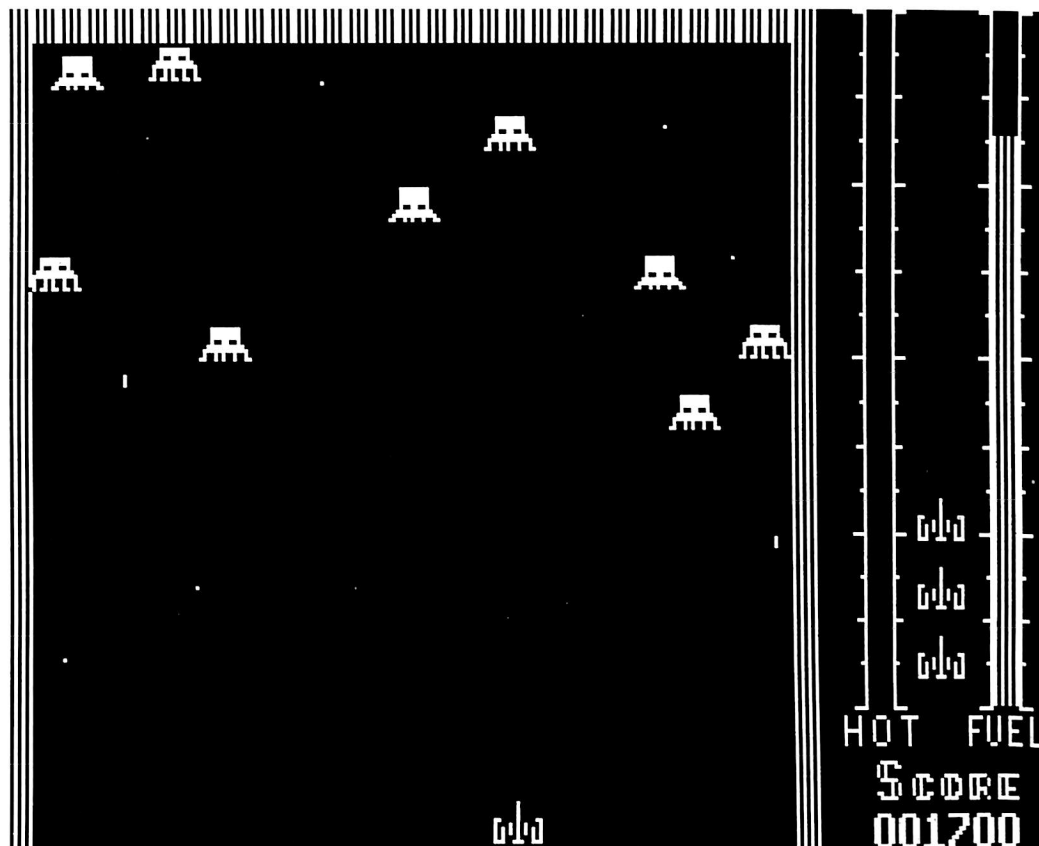
You should have a much better time playing this improved version of Threshold. You may even make it to the famous last board!
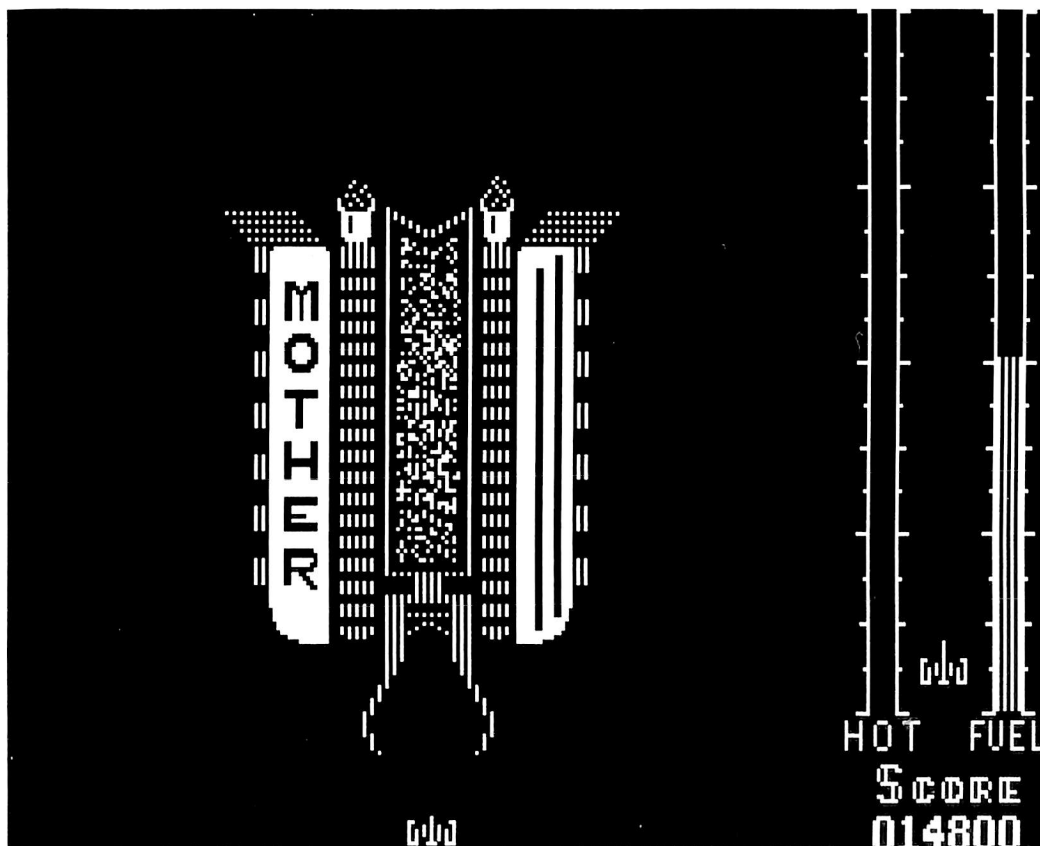
controller

```
340 POKE 9984 + (A1 - T1 ) * DOS * 256 + ABS
    (FAST * (DOS - 1 ) - A2 ) * 256 + A3 ,A4
    : NEXT : RETURN
610 PR# 0 : IN# 0 : POKE 860 ,MB : POKE 861 ,LT :
    POKE 862 ,LS : POKE 900 ,DOS - 1
1000 REM THRESHOLD CONTROLLER
1010 TK = 2 : LT = 35 : ST = 12 : LS = 12 : CD = WR : FAST
    = 1
1020 DOS = 13 : MB = 130 : POKE 775 ,96
1030 GOSUB 360 : GOSUB 490 : GOSUB 610
1040 IF PEEK (TRK ) > 32 AND TK < 33 THEN T1 = TK
    : TK = 32 : GOSUB 310 : TK = T1
1050 GOSUB 360 : GOSUB 490 : GOSUB 610
1060 TK = PEEK (TRK ) : ST = PEEK (SCT ) : IF TK <
    > LT THEN 1030
1070 TK = 0 : ST = 1 : GOSUB 360 : GOSUB 490 : GOSUB
    100
1080 GOSUB 360 : GOSUB 490 : GOSUB 100
1090 HOME : PRINT "COPYDONE" : END
1100 DATA 2ᐃ CHANGES ,32 ,0 ,8 ,131 ,32 ,0 ,9 ,195
10010 PRINT CHR$ (4) ; "BLOADᐃ RWTS .13 ,A$1900"
```

controller checksums

340	– $A295	1050	– $A25A
610	– $EDC0	1060	– $A036
1000	– $BDA7	1070	– $4340
1010	– $3E12	1080	– $D200
1020	– $CBD9	1090	– $27F8
1030	– $5A95	1100	– $E00F
1040	– $3D16	10010	– $60D6

Antique softkey for...

Checkers v2.1

by James E. Wallace

Odesta
3186 Doolittle Drive
Northbrook, IL 60062

Requirements:
Apple][Plus or equivalent
A blank initialized disk
A slave disk
A sector editor
Super IOB 1.2 or 1.5

Checkers is an excellent (but old) game that will provide entertainment for both the beginner and the experienced player. The package contains many features, including sixteen levels of skill from which to select and a very detailed manual. The disk was going to be handled by many high school students so a backup was essential.

Although I used a modified EPROM as presented in earlier issues of COMPUTIST (Nos. 6, 9, 16 and 22) to break into the program, many of you are not at liberty to modify, for instance, the school's or library's computer which you are using. I, therefore, included in the softkey procedure a jump to the monitor after several pages of the boot process have been loaded into the computer.

Protection for Checkers begins with the address field headers alternating between D5 AA 96 and D4 AA 96 for each physical sector on a track. Beginning with sector 0, this causes all even sectors to be $D5 while all odd sectors are $D4. **NOTE:** I am referring to *physical* sectors - not logical sectors.

In addition to address field header changes, the epilog has been changed from the standard DOS 3.3 DE AA EB to EF AA EB on all sectors. The protection covered thus far, of course, is easily handled by Super IOB using the Checkers Controller at the end of this

article. Unfortunately, the copy Super IOB provides will not be quite ready for immediate use. We obviously have to modify the boot code to make it happy with its new DOS 3.3 sector format. More easily said than done. Surprisingly, however, we will not have to change the code at all to make it look for $D5's all the time and not $D4's.

If you were to boot trace the program through the second stage of boot (or if you have a low tolerance for suffering, simply read and disassemble track 0 sector E from the original disk), you will find the following portion of the third stage boot process at $091A:

```
91A-   LDY  #$03
91C-   LDX  $082B         ;Fetch decoding seed
91F-   EOR  $0900,X       ;Decode next byte
922-   STA  $0500,X       ;Store in text page
925-   INX
926-   BNE  $091F         ;Continue till X=0
928-   INC  $0921         ;Change pages to read
92B-   INC  $0924         ; and write from
                          ; (self-altering code)
92E-   DEY                ;Keep going until no
92F-   BNE  $091F         ; pages are left
 .                         .
93E-   JMP  $078F         ;Go load the game.
```

This third stage of Checkers' boot process loads a portion of the primary text page (pages $05, $06, and $07) with the code that will actually load the game. The use of text page 1 is, of course, a form of program protection as the data stored there is normally lost during a reset. In order for us to have access to this code for modification, we need to instruct Super IOB to edit our copy as it is created so that it will load the final stage to pages $75 through $77 instead of the volatile text pages $05 through $07. And, as I promised those with no special reset capability, instead of JuMPing to $078F from $093E to continue, the Super IOB editor will put in a JuMP to the monitor.

After running the Super IOB copy and stopping the drive, you'll find there are two sections of code which merit attention. First is that which looks for the address field header. It looks like this:

```
7541-   LDX
7543-   LDA  $C08C,X  ;Read a byte
7546-   BPL  $7543
7548-   STA  $01      ;Store it in $01
754A-   NOP
754B-   NOP
754C-   LSR           ;Shift the byte right
754D-   EOR  #$6A     ;Is it now $6A?
754F-   BNE  $7543    ;No, try again
7551-   LDA  $C08C,X  ;Yes, read next byte
7554-   BNE  $7551
7556-   CMP  #$AA     ;Is it $AA?
...and so on...
```

The LSR at $754C is an instruction to shift the data in the accumulator to the right by one bit, introducing a zero on the left and taking away the bit on the right. Whether the computer reads a $D5 or a $D4 does not matter after an LSR has been performed on the byte - the results will always be $6A. Perhaps a picture here is worth a thousand words:

```
$D4 shifted right becomes $6A
  11010100  01101010  (carry = 0)

$D5 shifted right becomes $6A
  11010101  01101010  (carry = 1)
```

Clever, no? So we need not change anything here; to read $D5 all the time is perfectly acceptable to the existing code. Also pay attention to line $7548 which stores whichever byte is read ($D5 or $D4) in location $01.

The other section of code to examine begins at $7745:

```
7745-   JSR  $7541   ;Read address field
7748-   BCS  $7745   ;Retry on error (C=1)
774A-   LDA  $2E
774C-   STA  $0D
774E-   CMP  $0F
7750-   BNE  $7740
7752-   LDX  $2D     ;Get sector # we want
7754-   TXA          ;Move it to accum.
7755-   EOR  $01     ;Exclusive-OR it
                     ; with byte at $01.
7757-   AND  #$01    ;AND the results
                     ; with the number $1
7759-   BEQ  $7745   ;If result = 0, bad
                     ; read; go try again.
...Otherwise continue...
```

Location $2D contains the number of the physical sector about to be read. Line $7755 Exclusive-OR's the sector number with the byte stored at location $01 ($D5 or $D4, remember?). The result is then ANDed with the number $01. A final answer of 0 says an error has occurred; a 1 says all's well. Here is an example for those of you who would really like to understand the process. Remembering that even numbered sectors begin with $D5, look at what happens when Checkers reads, say, even numbered sector 8:

```
       Sector $08 = 0000 1000
    Header is $D5 = 1101 0101
                    -----------
Results of EOR'ing = 1101 1101
    AND with $01 = 0000 0001
                    -----------
  Final results = 0000 0001
                 (No Error)
```

However, if the sector header were $D4:

```
       Sector $08 = 0000 1000
    Header is $D4 = 1101 0100
                    -----------
Results of EOR'ing = 1101 1100
    AND with $01 = 0000 0001
                    -----------
  Final results = 0000 0000
                 (Error)
```

Of course, all of the address field headers on our copy will be $D5; so, to avoid errors, we need to ensure a final result of 1 even when an odd sector is about to be read. We do this by simply changing the opcode at $7755 from $45 (EOR $addr) to $49 (EOR #byte). Now the line will say EOR the number $01 - not the value stored at location $01.

Now that we understand the boot process, we're ready to modify it, right? Not quite. I'll cover very briefly the last problem encountered. During the loading of the text page with the code for the final boot, an Exclusive-OR is performed on each encoded byte as it is copied into the text page. The value is left in the accumulator as a seed for the next Exclusive-OR. Thus, if we were to attempt to modify a single byte of raw data in this portion of the code, we would alter every byte that follows our modified one.

But, not to fret. There is an alternative to having to "fix" two and a half pages of raw data just so we can change a few bytes. Remember, when we run our initial softkey copy, it will load the "converted" final stage data to pages $75 through $77. What we will do is modify this code as desired and use a sector editor to write these sectors back to the disk in its converted form. Then change the second stage of boot to have it simply load these sectors to pages $05 through $07 with no Exclusive OR'ing.

Making the copy

1) Type in the Checkers Controller listed at the end of this article. This controller will copy tracks 0-7 to normal DOS 3.3 format and edit the disk to jump into the monitor.

2) Install the controller into Super IOB 1.2 and copy the disk.

3) When the copy is complete, place it in the drive and boot it. It will only read from track 0 as it loads a portion of the boot code into pages $75 through $77, ending with a JuMP to the Apple monitor. Stop the drive by entering

C0E8

4) While still in the monitor, make the following changes to the code:

7581:00
758A:00
75F1:00
75FB:00 A0 00
7606:00
7755:49
775A:00

The above changes modify Checkers' final boot stage so that it will successfully read the new DOS 3.3 format.

5) Remove your Super IOB copy from the drive and boot a slave disk with

C600G

The remainder of this procedure is critically dependent upon the code that has been written to pages $75 through $77 not being disturbed by whichever sector editor you may use. You will also need to know what portion of memory your sector editor uses as a buffer to store the sector data it reads in. DiskEdit from COMPUTIST will not disturb pages $75-$77 and uses page $09 ($0900-$09FF) as its buffer. (CIA's Tricky Dick, by Golden Delicious Software, can also be used; its buffer is page $2E.) This procedure is based on the use of DiskEdit. If your editor has different requirements, alter the given addresses accordingly.

6) Remove the slave disk and place the disk containing DiskEdit in the drive and

RUN DISKEDIT

7) Remove the disk with DISKEDIT and again put in your Super IOB copy of Checkers. Read track 0 sector E and exit DiskEdit with "X·.

8) Enter the monitor.

CALL -151

9) Move the decoded data from $7541-75FF to replace the coded data in the disk sector (at $900- in DiskEdit).

0941<7541.75FFM

10) Now make the following changes which will restore (1) the instruction to load pages $05-$07 instead of $75-$77, but without EOR'ing, and (2) the jump to load the game instead of our jump to the monitor:

91F:BD
924:05
93F:8F 07

11) Return to BASIC and resume DiskEdit.

⬜C
RUN

12) Write the sector back to track 0 sector E of your Super IOB copy of Checkers.

13) Exit DiskEdit, call the monitor, and move the following data:

CALL -151
900<7600.76FFM

14) No changes are required here so return to BASIC and resume DISKEDIT.

⬜C
RUN

15) Write this page to track 0 sector D.

16) Exit DiskEdit, call the monitor, and move the last data:

CALL -151
900<7700.77FFM

17) Again, no changes; return to BASIC and resume DiskEdit.

⬜C
RUN

18) Write this page to track 0 sector C.
You should now have a softkeyed version of Checkers.

---

controller

```
1000 REM CHECKERS CONTROLLER
1010 TK = 0 : ST = 0 : LT = 07 : CD = WR
1020 T1 = TK : GOSUB 490 : POKE 47426 ,24
1030 GOSUB 430 : IF (ST = 15 ) OR (ST > 0 AND ST <
     8 ) THEN POKE 47445 ,212
1040 GOSUB 100 : ST = ST + 1 : POKE 47445 ,213 : IF
     ST < DOS THEN 1030
1050 IF BF THEN 1070
1060 ST = 0 : TK = TK + 1 : IF TK < LT THEN 1030
1070 GOSUB 230 : GOSUB 310 : GOSUB 490 : TK = T1
     : ST = 0
1080 GOSUB 430 : GOSUB 100 : ST = ST + 1 : IF ST <
     DOS THEN 1080
1090 ST = 0 : TK = TK + 1 : IF BF = 0 AND TK < LT THEN
     1080
1100 IF TK < LT THEN 1020
1110 HOME : PRINT "COPY^ COMPLETE" : END
5000 DATA 5^ CHANGES
5010 DATA 0 ,0 ,203 ,89
5020 DATA 0 ,0 ,205 ,255
5030 DATA 0 ,14 ,36 ,117
5040 DATA 0 ,14 ,63 ,89
5050 DATA 0 ,14 ,64 ,255
```

---

controller checksums

1000	– $356B	1090	– $973D
1010	– $63E6	1100	– $863E
1020	– $8DF2	1110	– $AAE8
1030	– $0515	5000	– $C4A4
1040	– $49A3	5010	– $63CE
1050	– $E227	5020	– $19D9
1060	– $C72A	5030	– $0CCB
1070	– $5B2D	5040	– $DB57
1080	– $4127	5050	– $7BA3

softkey for...

Microtype:
The wonderful world of Paws

by D. E. Pelzer

South-Western Publishing Co.
5101 Madison Road
Cincinnati, OH 45227

Requirements:
Apple][Plus or equivalent
Microtype disk #1 and #2
Super IOB 1.5
2 blank disks

Microtype is a typing tutor program that uses graphics, games and pictures to help improve touch typing skills. It is intended for use in the classroom environment, but works well for individual study. Apple][Plus machines need a shift-key modification or a Videx Enhancer][keyboard. The program does not allow incorrect responses during the early learning phases, which I liked in comparison with the other typing programs.

Of course, the program is protected. Letting the kids use the original was scary at best. The program couldn't be copied with Copy][Plus, Locksmith, or EDD. The publishers may be using some kind of bit insertion that won't allow reliable copies to be made.

During boot-up the Applesoft prompt appears on screen, hinting that a swap controller might work here. Locksmith Fast Copy shows that only tracks 3-22 are protected. After copying the disk with the swap controller I found that the program would not work with normal DOS. After some snooping in the protected DOS I found a JuMP call at $B944 that used a different read address field routine. The program does not come back to tracks 0-2 after loading in the modified DOS, so I made the program just skip over the altered read routine and keep running with their protected DOS. There are some routines in their DOS we need, so we will keep their DOS, but make a change so it will only read normal DOS formatted sectors.

Step by Step

1) Boot up your original disk and halt the program after the prompt appears on screen. Hit reset quickly 3 to 4 times until the drive stops and you get a prompt.

2) Enter the monitor and be sure that the 3 bytes at $B944 read 4C 02 BF (JMP $BF02).

```
CALL-151
B944L
```

If they don't match up, reboot and try again.

3) Move Microtype's RWTS to memory safe from the coming boot.

```
1900<B800.BFFFM
```

4) Boot a slave disk with no Hello program and save the RWTS on a disk with Super IOB on it.

```
C600G
BSAVE RWTS.MICROTYPE,A$1900,L$800
```

5) Install the Microtype controller on this page into Super IOB and use it to copy both Microtype disks. Don't forget to write protect!

The controller will copy tracks 0-2 using normal DOS, then use the Swap routine to copy the rest of the disk. Super IOB makes these sector edits to let Microtype read the normal format.

Track	Sector	Byte	From	To
$00	$08	$F4	$02	$4E
$00	$08	$F9	$BF	$B9

controller

```
1000 REM MICROTYPE CONTROLLER
1010 TK = 0 : LT = 3 : ST = 15 : LS = 15 : CD = WR : FAST
     = 1
1020 GOSUB 490 : GOSUB 610 : GOSUB 310 : GOSUB 490
     : GOSUB 610
1030 TK = 3 : LT = 35
1040 GOSUB 360 : GOSUB 490 : GOSUB 610
1050 GOSUB 360 : GOSUB 490 : GOSUB 610 : IF PEEK
     (TRK) = LT THEN 1070
1060 TK = PEEK (TRK) : ST = PEEK (SCT) : GOTO 1040
1070 HOME : PRINT "COPY^ DONE" : END
5000 DATA 2^ CHANGES
5010 DATA 0 ,8 ,244 ,78
5020 DATA 0 ,8 ,249 ,185
10010    PRINT    CHR$    (4    )    "BLOAD^
         RWTS.MICROTYPE,A$1900"
```

controller checksums

1000	– $356B	1060	– $EAE1
1010	– $FF63	1070	– $92EA
1020	– $E4AF	5000	– $3B3D
1030	– $1ACE	5010	– $4F3D
1040	– $8082	5020	– $24CA
1050	– $B914	10010	– $014A

softkey for Compress'...

General & Organic Chemistry Series

by Michael A. Coffey

COMPress
P.O. Box 102
Wentworth, NH 03282

COMPress puts out two excellent but expensive chemistry tutorial series, one for General Chemistry (DOS 3.3), and one for Organic Chemistry (DOS 3.2/3.3). Backing these up would be (1) logical, as disks used by students in a lab setting do not last long; and (2) economical, as $20 per disk was the backup fee until recently. The first seven disks of both series use the same type of scheme. Listed below are three different approaches to making a backup. Archiving the originals, students have been using a backup of each disk prepared by these methods for well over a year with no problems.

Bit Copiers

Bit copiers had little success on this series. EDD would backup General Chemistry #7 using T0-T2, T4-T22 normal, T3 sync. While Copy][Plus V4.4 worked for General Chemistry #2, Version 5 was successful on #2, #3, #6, and #8 using the same parameters. Nothing would touch the Organic Chemistry series.

Looking closer at the disks revealed that they have an altered DOS, an encrypted directory, and Track 3 looks like a giant sync $FF field. The HELLO program is called SSPROT$$1 and checks to see if Track 3 is correct for that disk and then calls a similar spelled program that acts as a typical turnkey program bringing up a greeting screen, menu, etc. These COMPress disks are not the only ones to use this procedure. I've come across some disks in the Vocational Education area that have the same thing. Knowing this, I tried a copy card.

Cards

Since the program checks for disk verification only once before before the main menu appears, the first seven disks of both series can be backed up by using the following:

1) Boot up the COMPress disk and allow the greeting screen to appear. Proceed to the main menu. At this point "card it". Central Point Software's Wildcard 2, (formally called the Alaska Card), and Replay II have both worked well for this.

2) Convert what you have 'carded' to a DOS 3.3 compatible file named HELLO. Place this file on a DOS 3.3 disk which I will call the backup. I recommend that you use a fast load patch or something like David DOS, otherwise, it will take forever to load upon bootup.

3) Get around the encrypted directory by using a sector editor on the COMPress disk and change the first three bytes of Track 11, Sector 0. On the General Chemistry series change them to $FF 11 0F. Set your sector editor to DOS 3.2 for the Organic series and change them to $FF 11 0C. This allows you to CATALOG the disk under normal DOS.

4) Using your favorite method, transfer all of the files on the COMPress disk to the formatted back side of the backup, except for SSPROT$$1 and the other similarly spelled program. A program that will convert DOS 3.2 files to DOS 3.3 will be necessary for the Organic series.

Even if the HELLO program is compressed, there are too many files to fit on the same side of the disk as the HELLO program, so they must be put on the back side.

You now have a disk that will give the main menu upon bootup. At this point you will have to turn the disk over to use any of the menu options. Many students will forget to turn the disk over, so you may prefer the next method.

Protected backups

The same knowledge that allowed the previous procedure can be used to produce a single sided, copy protected backup for the first seven disks of each series. This can be done by taking the program that checks for the copy protection and the track it checks from one of the COMPress series and putting it on the backup for each of the rest.

1) INIT a backup disk with normal DOS 3.3.

2) Change the name of the turnkey program from HELLO to SSPROT$$1 using any of the many available programs to do this. If you don't have one, type in and save the following as the HELLO program:

10 PRINT CHR$(4);'BRUN SSPROT$$1'

3) Use Copy II Plus or another bit copier with sync to copy Track 3 from the General Chemistry #2 disk to your backup.

4) Use a sector editor on the COMPress disk and change the first three bytes of Track 11, Sector 0. On the General Chemistry series change them to $FF 11 0F. Set your sector editor to DOS 3.2 for the Organic series and change them to $FF 11 0C.

5) Transfer SSPROT$$1 from the General Chemistry #2 disk to your backup.

6) Using your favorite method, transfer all of the files on the COMPress disk to the backup disk except SSPROT$$1. Since the Organic Chemistry series is DOS 3.2 formatted, a program that will convert DOS 3.2 files to DOS 3.3 will be necessary.

Now the program will boot up, check for its protection, and proceed. The protection is not the one on the original, but the program doesn't care as long as Track 3 and SSPROT$$1 are from the same disk. If Track 3 will not copy from General Chemistry #2, try one of the others. This last method allows a school to make its own backups which saves money and discourages students from pirating the disks and causing them legal problems.

Utilizing the Archon Editor with...

ANIMATOR

By Ray Darrah

Requirements:
Apple][Plus or better with 64K or more
Archon and an unlocked version of its editor

After using the article on page 20 of COMPUTIST No. 28 to unlock your hidden Archon Editor, you're probably saying to yourself, "Now what? I can create and edit block shapes but that's all."

Speak to yourself no longer. For presented here is my creation: a program that will allow you to animate (move about the screen) up to eight of these block shapes simultaneously.

How I Did It

The first step when writing a program of this nature is to decipher the data to be manipulated. That is, find out where in memory the block shapes are stored and how this memory relates to the intended image.

I accomplished both these tasks with little difficulty. Since the broken Archon Editor saves the shape files on a DOS 3.3 disk as a normal Binary file, finding where in memory the shapes were stored was as simple as BLOADing a shape file from the monitor and examining memory areas $AA72-$AA73 and $AA6C-$AA6D to find the address and length respectively of the last BLOADed file. This then revealed that the shape files reside at memory locations $6000-$81FF.

How the Data is Stored

Examining the data contained in a test shape file showed that the data in the shape files can be directly stored on the screen to produce the intended image.

I also noted that although the commands: "< >" and " ^ " change the editing area length and width respectively, each shape still occupies 96 bytes of memory (4 horizontal by 24 vertical).

There are 80 possible shapes stored sequentially (row two following row one and shape 2 following shape 1) starting at memory location $6000 and going through location $7DFF.

There is some data at $7E00-$7E77 but I'm not sure what it is or what it is used for so I decided to designate memory locations $7E00-$81BF as 8 sequentially stored (row two following row one and area two following area one) background areas. That is, areas in which the background (memory "behind" the shape) will be temporarily stored.

The Background Areas

With my program, you are limited to animating up to eight shapes because I have designated only eight background areas which are numbered 0-7.

Each background area is 120 bytes long because if you want to draw a block shape at an X coordinate not evenly divisible by 7 (there are 7 dots per byte) you will affect five bytes horizontally even though the shape is only four bytes wide (refer to CORE 1, graphics).

The Color Grid

Those of you familiar with the hi-res screen know that there are 280 dots horizontally and 192 dots vertically. However, the color produced by any of these dots depends on whether the dot is on an even X coordinate or an odd X coordinate. Even dots can only be blue, purple or white while odd dots can only be green, red or white. Because of this extreme limitation of Apple graphics, if a block shape were to be drawn at an odd X coordinate, the colors in it would change.

To overcome this problem (and a few others), my program only recognizes a 140 by 192 grid. Therefore, when specifying an X coordinate, be sure it is within the 0 through 139 range. Note that you can still draw a shape anywhere on the screen, you just can't position the shape horizontally as accurately as before. You will hardly notice the difference.

Calling the Animator

To draw a shape, you have to set up some memory locations and 6502 registers. They are as follows:

On Entry:

224 ($E0) - holds the X coordinate of the shape to be drawn or erased. Legal values are from 0 - 139 ($00 - $8B) however, you may wish to limit this memory location to a high of 126 ($7E) to insure that the right edge of the shape won't wrap around to the left side of the screen.

225 ($E1) - holds the Y coordinate of the shape to be drawn or erased. Legal values are from 0 - 191 ($00 - $BF) however, you may wish to limit this memory location to a high of 168 ($A8) to insure that the bottom edge of the shape won't wrap around to the top of the screen.

X register - holds the background area to use for the operation. Legal values range from 0 - 7 ($00 - $07). When a shape is drawn, the bytes "behind" the shape will be saved in the specified background area and when a shape is erased, the bytes in the specified background area will be put on to the screen instead of a shape.

Y register - holds the shape to draw or erase. Legal values are from 0 - 79 ($00 - $4F). Note that this value has no effect when erasing a shape because the background area specified by the X register will be drawn on the screen and the shape specified will not matter.

A register - holds a number designating what type of operation is desired. Legal values are from 0 - 255 ($00 - $FF). However, only three types of calls are possible. If the Accumulator (A register) contains a zero, then the shape will be erased (replaced with background information) at the location specified. If the Accumulator contains a number greater than 127 ($7F), then the shape will be drawn as a block image. That is, "holes" (zero bits) in the shape will be opaque. If the Accumulator holds some other number (less than 128 and greater than 0), then the shape will be drawn as if it were a sprite. That is, "holes" in the shape will be transparent.

230 ($E6) - holds the hi-res page number you wish to manipulate. Legal values are 32 ($20) and 64 ($40) indicating page 1 and 2 respectively.

On Exit:

When the program returns, the desired operation will be performed and all registers will have been scrambled. Also, location 225 ($E1) will have been altered. Therefore, I recommend you designate a separate area for coordinate storage and update $E0 and $E1 before each call.

The program also scrambles memory locations $00 - $09, $0D, $26 - $27, $9D - $A1 and $A5 - $A9. Note that the value in these locations previous to the call doesn't matter. Also note that the use of these locations shouldn't interfere with normal BASIC operations.

Defining a Shape

You can define a shape using the Archon Editor any way you want. However, if you change the color bit (bit 7) of a shape within one horizontal line of the shape, an unavoidable color inaccuracy will occur.

Typing It In

First key in the hexdump near the end of this article using the instructions on the inside front cover of this magazine and save it with:

BSAVE OBJ.ANIM, A$83B8, L$10E

Next, key in the BASIC program near the end of this article and save it with:

SAVE MAKE TABLES.ANIM

This program creates a 505 byte table that is to precede the Hexdump you just keyed in. I figured it would be easier for you to type in this BASIC program than 505 bytes of a Hexdump. Now execute this program which will save the tables it creates to disk under the name "TAB.ANIM."

RUN MAKE TABLES.ANIM

Now to combine the tables with the machine language. Start by loading the machine language portion:

BLOAD OBJ.ANIM

Next, load the tables that "MAKE TABLES.ANIM" has created.

BLOAD TAB.ANIM

Finally, save them both as one file.

BSAVE ANIMATOR, A$81C0, L$306

Calling Animator from BASIC

Although best results can be obtained by incorporating the Animator into a much larger machine language program, a knowledge of machine language is not needed to use the Animator.

To use the Animator from BASIC, you must first set up a machine language program that calls the Animator with the registers set correctly. Such a subroutine could look like this:

```
0300 - A2 00    LDX #$00   ;Get backgnd #
0302 - A0 00    LDY #$00   ;Get shape #
0304 - A9 00    LDA #$00   ;Get entry code
0306 - 4C B8 83 JMP $83B8  ;Call animator
```

This subroutine is set up by the following BASIC line:

63000 FOR A = 768 TO 776 : READ B : POKE A , B : NEXT : RETURN : DATA 162 ,0 ,160 ,0 ,169 , 0 ,76 ,184 ,131

After GOSUBing this line to set up the previous machine language subroutine, you must do a "CALL 768" to draw or erase a shape.

Before doing this however, you must do POKEs to the appropriate memory locations to indicate what the Animator is to do. As explained earlier in this article, location 224 & 225 contain the X & Y coordinates of the shape respectively and location 230 contains the hi-res page number. Where the article mentions X, Y and A registers, you should substitute a POKE to memory locations 769, 771 and 773 respectively.

The subroutine that calls the Animator is completely re-locatable and therefore can be put just about anywhere you want (as long as it doesn't interfere with something else). If you change its starting address, you will also have to change the locations that you POKE and CALL accordingly.

Animator Source Code

```
81C0- MOD7     .EQ $81C0   THE REMAINDER OF X/7 FOR 0 - 139
824C- XDIV     .EQ $824C   THE INTEGER ANSWER OF X/7 FOR 0-139
82D8- GBASL    .EQ $82D8   LSB OF HI-RES LINE BASE ADDRESS FOR 0 - 184 STEP 8
82F0- GBASH    .EQ $82F0   MSB OF SAME
8308- SHINDL   .EQ $8308   POINTER TO START OF ALL 80 SHAPES
8358- SHINDH   .EQ $8358   MSB OF SAME
83A8- BKINDL   .EQ $83A8   POINTER TO START OF 8 BACKGROUND AREAS
83B0- BKINDH   .EQ $83B0   MS OF SAME
0026- LINADDR  .EQ $26     CURRENT HI-RES LINE ADDRESS
00E0- XCOOR    .EQ $E0     X COORIDNATE
00E1- YCOOR    .EQ $E1     Y COORDINATE
00E6- PAGE     .EQ $E6     HI-RES PAGE ONE OR TWO ($20 OR $40)
0000- PTR1     .EQ $0      START OF CURRENT SHAPE
0002- PTR2     .EQ $2      START OF CURRENT BACKGROUND AREA
0004- CNTR1    .EQ $4      NUMBER OF LINES LEFT
0009- CNTR2    .EQ $9      WHERE WE ARE IN THE BACKGROUND AREA
009D- LINBUF   .EQ $9D     - $A1 CURRENT LINE BUFFER
00A5- MSBBUF   .EQ $A5     - $A9 CURRENT MSB BUFFER
0005- STARTX   .EQ $5      STARTING HI-RES LINE OFFSET
0006- NUMSHFT  .EQ $6      # OF TIMES TO ROTATE SHAPE
0007- SHFT1    .EQ $7      SAME FOR DECREMENT
0008- SHINDEX  .EQ $8      POINTER TO WHERE IN THE SHAPE WE ARE
000D- YSAVE    .EQ $D      TEMPORARY STORAGE FOR Y

               .OR $83B8   START AFTER ABOVE TABLES
               .TF OBJ.ANIM

83B8: 48        SETUP  PHA            SAVE STATUS OF CALL
83B9: B9 08 83          LDA SHINDL,Y  SET UP POINTER TO SHAPE
83BC: 85 00             STA PTR1
83BE: B9 58 83          LDA SHINDH,Y
83C1: 85 01             STA PTR1+1
83C3: BD A8 83          LDA BKINDL,X  SET UP POINTER TO BACKGROUND AREA
83C6: 85 02             STA PTR2
83C8: BD B0 83          LDA BKINDH,X
83CB: 85 03             STA PTR2+1
83CD: 20 6B 84          JSR LINCALC   GET ADDRESS OF FIRST LINE
83D0: A4 E0             LDY XCOOR     FIGURE OUT STARTING OFFSET IN LINE
83D2: B9 4C 82          LDA XDIV,Y
83D5: 85 05             STA STARTX
83D7: A4 E0             LDY XCOOR     AND NUMBER OF TIMES TO ROTATE BITS IN SHAPE
83D9: B9 C0 81          LDA MOD7,Y
```

Note that smooth, rapid animation is difficult, if not impossible, to accomplish through BASIC. However, there are several slow animation applications that would be adequately achieved through BASIC.

Possible Modifications

If you're not big into color, you could patch in four NOPs (EA) starting at address $8402. This will stop the Animator from forcing the color bit of the area "under" the shape to that of the shape itself. In this mode only three colors (blue, purple and white) are available but the background won't change colors in large areas where the shape is mostly transparent.

A bit of re-writing of the SPRTDRAW portion of the program could alleviate this problem even more. If you were to make this portion force the color bit only if the corresponding byte of the shape wasn't zero, this then would severely reduce the color distortion present in shapes with large transparent areas.

By inserting a small amount of code in the two places where the program saves the background area, you could add a shape-to-background collision detector. This provides an easy way of telling whether a particular shape has "hit" something on the screen.

Tips for good Animation

It is VERY important when animating more than one shape simultaneously to erase them in the reverse order they were drawn. To see why this is important; following is an account of what happens when you don't.

First, we'll say you draw shape "one" on the screen with the background saved in background area one. So far so good.

Next, let's say you draw shape "two" right over shape "one" with the background saved in background area two. The background that will be saved is primarily a picture of shape "one".

If you were to erase shape "one" right now, shape "two" would disappear from the screen, but this is expected. However, if you were then to erase shape "two," an image of shape "one" would be placed on the screen (even though you thought you had erased shape "one") because that is what background area two contains.

Finally, to smoothen your animation and to avoid flicker you could time your shape drawing binges to occur every 1/30th of a second. If have an Apple //e however, you may use the MSB of location $C109. Bit 7 of this byte is either a one or a zero depending on whether the raster scan is on the screen or not. By waiting for the right bit to pop up before doing all of your shape drawing, you could smoothen your animation by a power of ten.

Also, you could do the old hi-res page switching technique where one hi-res page is being viewed while the other is being updated.

Finally, you MUST load the Animator AFTER loading the shapes it is to animate. This is because the Animator resides in some of the same memory as the Archon shapes.

Animator Source Code Continued

```
83DC: 85 06          STA NUMSHFT
83DE: 85 07          STA SHFT1
83E0: A9 18          LDA #24       24 LINES IN EVERY SHAPE
83E2: 85 04          STA CNTR1
83E4: A9 03          LDA #3        END OF FIRST LINE OF SHAPE
83E6: 85 08          STA SHINDEX
83E8: A9 00          LDA #0        FIRST BYTE OF BACKGROUND
83EA: 85 09          STA CNTR2
83EC: 68             PLA           RESTORE STATUS OF CALL

83ED: F0 2B  WHERE2  BEQ ERASER    Z=1 THEN ERASE
83EF: 30 46          BMI BLKDRAW   N=1 THEN BLOCK DRAW

83F1: 20 88 84 SPRTDRAW JSR GET.LINE PUT HORIZONTAL LINE OF SHAPE INTO BUFFERS
83F4: A2 00          LDX #0        START WITH LEFTMOST BYTE
83F6: A4 05          LDY STARTX    PUT AT FIRST HORIZONTAL POS
83F8: 84 0D  SPRT1   STY YSAVE     SAVE HORIZONTAL POSITION
83FA: B1 26          LDA (LINADDR),Y GET BYTE FROM SCREEN
83FC: A4 09          LDY CNTR2     GET OFFSET INTO BACKGROUND
83FE: E6 09          INC CNTR2     NEXT ONE NEXT TIME
8400: 91 02          STA (PTR2),Y  SAVE SCREEN BYTE IN BACKGROUND AREA
8402: 29 7F          AND #$7F      JUST LSBS PLEASE
8404: 15 A5          ORA MSBBUF,X  MAKE MSB THAT OF SHAPE
8406: 15 9D          ORA LINBUF,X  ADD REST OF BITS
8408: A4 0D          LDY YSAVE     GET HORIZONTAL POSITION AGAIN
840A: 91 26          STA (LINADDR),Y PUT THIS ON SCREEN
840C: C8             INY           NEXT HORIZONTAL POS
840D: E8             INX           NEXT BYTE OF BUFFER
840E: E0 05          CPX #5        DONE WITH LINE?
8410: 90 E6          BCC SPRT1     NO, KEEP GOING
8412: 20 5E 84       JSR MOV.DOWN  YES, GO TO NEXT LINE
8415: C6 04          DEC CNTR1     DONE WITH ALL LINES?
8417: D0 D8          BNE SPRTDRAW  NO, DO NEXT LINE
8419: 60             RTS           DONE, RETURN

841A: A4 05  ERASER  LDY STARTX    START AT UPPER LEFT
841C: 84 0D          STY YSAVE     SAVE HORIZONTAL OFFSET
841E: A2 04          LDX #4        FIVE BYTES PER LINE
8420: A4 09  ERASE1  LDY CNTR2     GET OFFSET INTO BACKGROUND AREA
8422: E6 09          INC CNTR2     NEXT ONE NEXT TIME
8424: B1 02          LDA (PTR2),Y  GET BYTE OF BACKGROUND
8426: A4 0D          LDY YSAVE     GET HORIZONTAL OFFSET AGAIN
8428: E6 0D          INC YSAVE     NEXT ONE NEXT TIME
842A: 91 26          STA (LINADDR),Y PUT BYTE ON SCREEN
842C: CA             DEX           DONE WITH LINE?
842D: 10 F1          BPL ERASE1    NO, DO NEXT BYTE
842F: 20 5E 84       JSR MOV.DOWN  YES, GO TO NEXT LINE
8432: C6 04          DEC CNTR1     DONE WITH ALL LINES?
8434: D0 E4          BNE ERASER    NOPE, DO NEXT LINE
8436: 60             RTS           DONE, RETURN

8437: 20 88 84 BLKDRAW JSR GET.LINE PUT LINE OF SHAPE INTO BUFFERS
843A: A2 00          LDX #0        START AT UPPER LEFT OF SHAPE
843C: A4 05          LDY STARTX    FIRST HORIZONTAL POS
843E: 84 0D  BLK1    STY YSAVE     SAVE HORIZONTAL OFFSET
8440: B1 26          LDA (LINADDR),Y GET BYTE FROM SCREEN
8442: A4 09          LDY CNTR2     GET OFFSET INTO BACGROUND AREA
8444: E6 09          INC CNTR2     NEXT ONE NEXT TIME
8446: 91 02          STA (PTR2),Y  SAVE BYTE FROM SCREEN IN BACKGROUND AREA
8448: B5 9D          LDA LINBUF,X  GET BYTE FROM SHAPE
844A: 15 A5          ORA MSBBUF,X  RESTORE MSB
844C: A4 0D          LDY YSAVE     GET HORIZONTAL POS AGAIN
844E: 91 26          STA (LINADDR),Y PUT BYTE ON SCREEN
8450: C8             INY           NEXT HORIZONTAL POS
8451: E8             INX           NEXT BYTE IN BUFFER
8452: E0 05          CPX #5        DONE WITH LINE?
8454: 90 E8          BCC BLK1      NO, DO NEXT BYTE
8456: 20 5E 84       JSR MOV.DOWN  YES, GO TO NEXT LINE
```

```
8459: C6 04        DEC CNTR1     DONE WITH ALL THE LINES?
845B: D0 DA        BNE BLKDRAW   NO, DO NEXT LINE
845D: 60           RTS           DONE: RETURN


845E: E6 E1  MOV.DOWN INC YCOOR   NEXT Y COORDINATE
8460: A5 27        LDA LINADDR+1 MERELY ADD $400 TO ADDRESS
8462: 29 1F        AND #$1F      STRIP OFF PAGE INFO
8464: 18           CLC
8465: 69 04        ADC #4
8467: C9 20        CMP #$20      IF PAST $2000 THEN Y POS IS A MULTIPLE OF 8
8469: 90 18        BCC GOT.IT    NOT MULTIPLE OF 8 SO EXIT


846B: A5 E1  LINCALC  LDA YCOOR   USE CURRENT Y POS FOR NEW LINE ADDRESS
846D: 4A           LSR           DIVIDE BY EIGHT FOR 24 BYTE TABLE
846E: 4A           LSR
846F: 4A           LSR
8470: A8           TAY           PUT IN Y FOR OFFSET INTO TABLE
8471: B9 D8 82     LDA GBASL,Y   GET LSB
8474: 85 26        STA LINADDR
8476: B9 F0 82     LDA GBASH,Y   AND MSB OF NEAREST MULTIPLE OF 8 ABOVE
8479: 85 27        STA LINADDR+1
847B: A5 E1        LDA YCOOR     RE-GET Y POS
847D: 29 07        AND #7        USE REMAINDER OF Y/8 * 400 FOR EXACT ADDRESS
847F: 0A           ASL           MULTIPLY BY 4 (MSB = $400)
8480: 0A           ASL
8481: 65 27        ADC LINADDR+1 ADD INTO MSB THUS MAKING *4 = *400
8483: 05 E6  GOT.IT  ORA PAGE     PUT ON PAGE INFO
8485: 85 27        STA LINADDR+1
8487: 60           RTS           AND RETURN


8488: A4 08  GET.LINE LDY SHINDEX  GET OFFSET INTO SHAPE
848A: A9 00        LDA #0        ZERO RIGHT EDGE OF SHAPE BUFFER
848C: 85 A1        STA LINBUF+4
848E: A2 03        LDX #3        COPY LINE OF SHAPE INTO BUFFER (4 BYTES)
8490: B1 00  COPY1  LDA (PTR1),Y GET BYTE OF SHAPE
8492: 95 9D        STA LINBUF,X  STORE IN BUFFER
8494: 29 80        AND #$80      GET HIGH BIT
8496: 95 A5        STA MSBBUF,X  SAVE IT FOR LATER USE
8498: 88           DEY           MOVE BACKWARDS THROUGH LINES
8499: CA           DEX           DONE WITH 4 BYTES
849A: 10 F4        BPL COPY1     NO, CONTINUE
849C: A5 A8        LDA MSBBUF+3  PUT BIT7 OF BYTE3 IN BYTE4 (IN CASE OF ROT)
849E: 85 A9        STA MSBBUF+4


84A0: C6 07  ROTATE1  DEC SHFT1   7 BIT ROTATION LEFT
84A2: 30 16        BMI EXIT1     DONE, RETURN
84A4: A5 9D        LDA LINBUF     START WITH LEFTMOST BYTE
84A6: 0A           ASL           ONE EXTRA SHIFT FOR THIS ONE
84A7: A2 00        LDX #0        ZERO INDEX INTO BUFFERS
84A9: F0 02        BEQ ASL2      SKIP RE-LOAD ..ALWAYS
84AB: B5 9D  ROTATE2  LDA LINBUF,X GET NEXT BYTE OF LINE
84AD: 0A     ASL2    ASL           STRIP OFF BIT 6 INTO C
84AE: 36 9E        ROL LINBUF+1,X MOVE INTO BIT 0 OF NEXT BYTE
84B0: 4A           LSR           RESTORE BIT POSITIONS
84B1: 95 9D        STA LINBUF,X  PUT BACK ROTATED BY 1 BYTE
84B3: E8           INX           NEXT BYTE IN BUFFER
84B4: E0 04        CPX #4        DONE?
84B6: 90 F3        BCC ROTATE2   NOPE, DO ANOTHER BYTE
84B8: B0 E6        BCS ROTATE1   ..ALWAYS
84BA: A5 08  EXIT1   LDA SHINDEX  GO TO NEXT LINE OF SHAPE FOR NEXT CALL
84BC: 18           CLC
84BD: 69 04        ADC #4
84BF: 85 08        STA SHINDEX
84C1: A5 06        LDA NUMSHFT   RESTORE NUMBER OF TIMES TO ROTATE
84C3: 85 07        STA SHFT1
84C5: 60           RTS           RETURN
```

```
83B8: 48 B9 08 83 85 00 B9 58    $6A98
83C0: 83 85 01 BD A8 83 85 02    $C2B1
83C8: BD B0 83 85 03 20 6B 84    $44C9
83D0: A4 E0 B9 4C 82 85 05 A4    $A88D
83D8: E0 B9 C0 81 85 06 85 07    $49B7
83E0: A9 18 85 04 A9 03 85 08    $08C2
83E8: A9 00 85 09 68 F0 2B 30    $56B5
83F0: 46 20 88 84 A2 00 A4 05    $3F79
83F8: 84 0D B1 26 A4 09 E6 09    $2943
8400: 91 02 29 7F 15 A5 15 9D    $EEA3

8408: A4 0D 91 26 C8 E8 E0 05    $A01B
8410: 90 E6 20 5E 84 C6 04 D0    $6E38
8418: D8 60 A4 05 84 0D A2 04    $4FE2
8420: A4 09 E6 09 B1 02 A4 0D    $6D1C
8428: E6 0D 91 26 CA 10 F1 20    $D813
8430: 5E 84 C6 04 D0 E4 60 20    $1887
8438: 88 84 A2 00 A4 05 84 0D    $754D
8440: B1 26 A4 09 E6 09 91 02    $9049
8448: B5 9D 15 A5 A4 0D 91 26    $01B9
8450: C8 E8 E0 05 90 E8 20 5E    $92F4

8458: 84 C6 04 D0 DA 60 E6 E1    $F525
8460: A5 27 29 1F 18 69 04 C9    $C1B5
8468: 20 90 18 A5 E1 4A 4A 4A    $571C
8470: A8 B9 D8 82 85 26 B9 F0    $8864
8478: 82 85 27 A5 E1 29 07 0A    $62C4
8480: 0A 65 27 05 E6 85 27 60    $B102
8488: A4 08 A9 00 85 A1 A2 03    $D4FF
8490: B1 00 95 9D 29 80 95 A5    $84E3
8498: 88 CA 10 F4 A5 A8 85 A9    $8F04
84A0: C6 07 30 16 A5 9D 0A A2    $E788

84A8: 00 F0 02 B5 9D 0A 36 9E    $B8D5
84B0: 4A 95 9D E8 E0 04 90 F3    $83C3
84B8: B0 E6 A5 08 18 69 04 85    $2555
84C0: 08 A5 06 85 07 60          $4DE6
```

Make Tables.anim

```
10 FOR X = 0 TO 139 : POKE 33216 + X , X * 2 - INT
   (X * 2 / 7 ) * 7
20 POKE X + 33356 , X / 3.5 : NEXT
30 FOR X = 0 TO 23 : READ Y : POKE X + 33496 , Y - INT
   (Y / 256 ) * 256 : POKE X + 33520 , Y / 256
   : NEXT
40 FOR X = 0 TO 79 : Y = 24576 + X * 96 : POKE 33544
   + X , Y - INT (Y / 256 ) * 256 : POKE 33624
   + X , Y / 256 : NEXT
50 FOR X = 0 TO 7 : Y = 32256 + X * 120 : POKE 33704
   + X , Y - INT (Y / 256 ) * 256 : POKE 33712
   + X , Y / 256 : NEXT
60    PRINT    CHR$    (4    )    "BSAVE^
   TAB.ANIM,A$81C0,L$1F9"
100 DATA 8192 ,8320 ,8448 ,8576
110 DATA 8704 ,8832 ,8960 ,9088
120 DATA 8232 ,8360 ,8488 ,8616
130 DATA 8744 ,8872 ,9000 ,9128
140 DATA 8272 ,8400 ,8528 ,8656
150 DATA 8784 ,8912 ,9040 ,9168
```

checksums

10	– $83FB	100	– $8DBC
20	– $79E0	110	– $2956
30	– $5CA8	120	– $BB12
40	– $DD96	130	– $C467
50	– $0F01	140	– $DC64
60	– $684C	150	– $1963

Customizing the Monitor by adding...

65C02

by Earl Taylor

Requirements:
Access to an EPROM burner
Apple][Plus (not //e or //c)

The 6502 pin-for-pin compatible 65C02 microprocessor from Western Design Center is available for 6502-based computers like the Apple][and][Plus. This new microprocessor offers eight entirely new instructions, two new addressing modes and extended addressing modes for many of the original instructions. The 65C02 can boost your Apple's computing horsepower.

I obtained one of the chips and installed it in my][+. It powered-up and ran immediately and I have not had any problems with it so far (but I keep my old standard handy just in case).

Having recently upgraded to revision 2.0 of the S-C Macro Assembler (which supports assembly for the 65C02), it looked like I was ready to explore the powers of the new processor. But, I soon realized how easily I had taken the Apple's ROM-resident disassembler for granted. The normal disassembler viewed the new 65C02 opcodes as illegal and simply displayed the ??? mnemonic whenever it found one. Could this old dog be taught the 'C02's tricks?

I studied the disassembler code to see how it might (and if it could) be done. It was possible but there was very little space in the ROM for the required new code. Something had to be given up.

Say Bye-Bye to Lo-res Commands

I extended the disassembler by sacrificing support for lo-res graphics (VLIN, HLIN,

```
* -------------------------------------------------------------- *
*                    MODIFIED DISASSEMBLER TO                    *
*                   ACOMMODATE NEW 65C02                         *
*                         OPCODES                                *
* -------------------------------------------------------------- *

3A-    PCL    .EQ $3A     program counter low byte
3B-    PCH    .EQ $3B     program counter high byte
2E-    FORMAT .EQ $2E     address field print format
2F-    LENGTH .EQ $2F     no. of bytes of address
F879-  SCRN2  .EQ $F879   nibble unpack routine
F948-  PRBLNK .EQ $F948   print 3 blanks (X=0 on return)
F962-  FMT1   .EQ $F962   table of address mode codes for each opcode
F9A6-  FMT2   .EQ $F9A6   table of address formats and instruction lengths
FDED-  COUT   .EQ $FDED   character output
FD96-  PRYX2  .EQ $FD96   print Y and X in hex as "yyxx-   "
FAB7-  XPTNOP .EQ $FAB7

       .OR $F88C
       .TF OBJ.MODIFIED INSDS2


* -------------------------------------------------------------- *
*         determine address field format, instruction length    *
*                and index into mnemonic tables                  *
* -------------------------------------------------------------- *

F88C: A1 3A    INSDS2 LDA (PCL,X) get instruction opcode (X must be 0 on entry)
F88E: A8              TAY         save in Y
F88F: 4A              LSR         divide by 2, test b0
F890: 90 05           BCC IEVEN   its even (x0,x2,x4,x6,x8,xA,xC,xE)
F892: 6A              ROR         its odd, test b1, set b7
F893: B0 0C           BCS ERR     bad code (x3,x7,xB,xF)
F895: 29 87           AND #$87    good odd opcode (yyyxxx01) becomes 10000xxx
F897: 4A       IEVEN  LSR         divide by 2, remainder in carry
F898: AA              TAX         index into address mode table
F899: BD 62 F9        LDA FMT1,X  get address mode code byte
F89C: 20 79 F8        JSR SCRN2   remainder used to select high/low nibble
F89F: D0 03           BNE GETFMT  any nonzero indicates good opcode
F8A1: A9 0F    ERR    LDA #$0F    bad opcode, set implied address mode
F8A3: A8              TAY         and overwrite opcode with "???" code
F8A4: AA       GETFMT TAX         use address mode code
```

Disassembly

```
F8A5: BD A6 F9         LDA FMT2,X    to select print format and instr. length
F8A8: 85 2E            STA FORMAT    save format for caller
F8AA: 29 03            AND #$03      & strip down to length only
F8AC: 85 2F            STA LENGTH    and save for caller
F8AE: 98               TYA           recall opcode
F8AF: 29 1F            AND #$1F      perform mask for CHKIND
F8B1: 20 B7 FA         JSR XPTNOP    handle zp indirect, exception and "???" codes
F8B4: F0 18            BEQ DBLNDX    got one, skip over old MNNDX
F8B6: 29 8F     MNNDX  AND #$8F      form mnemonic index for other opcodes
F8B8: AA               TAX           routine converts opcode into mnemonic index
F8B9: 98               TYA           between $00 and $3F according to
F8BA: A0 03            LDY #$03      opcode's bit pattern as follows:
F8BC: E0 8A            CPX #$8A         (x = significant bit,
F8BE: F0 0B            BEQ PT4          y = insignificant bit)
F8C0: 4A        PT2    LSR           opcode    resulting index    (hex range)
F8C1: 90 08            BCC PT4       xxxxx000 - 000xxxxx        $00-$1F
F8C3: 4A               LSR           xxxyy100 - 00100xxx        $20-$27
F8C4: 4A        PT3    LSR           1xxx1010 - 00101xxx        $28-$2F
F8C5: 09 20            ORA #$20      xxxyyy10 - 00110xxx        $30-$37
F8C7: 88               DEY           xxxyyy01 - 00111xxx        $38-$3F
F8C8: D0 FA            BNE PT3
F8CA: C8               INY
F8CB: 88        PT4    DEY
F8CC: D0 F2            BNE PT2
F8CE: 0A        DBLNDX ASL           double to suit rearranged mnemonics tables
F8CF: 60               RTS
```

```
                .OR $F962
                .TF OBJ.MODIFIED TABLES

  * ------------------------------------------------------------------- *
  *                                                                     *
  *             Address mode code values used to create                *
  *                          FMT1 table                                 *
  *                                                                     *
  * ------------------------------------------------------------------- *

  00-    ERR   .EQ $0    indicates illegal opcode
  01-    IMM   .EQ $1    #$nn
  02-    ZPG   .EQ $2    $nn
  03-    ABS   .EQ $3    $nnnn
  04-    ZXI   .EQ $4    ($nn,X)
```

PLOT etc.). Shortly afterward however, a friend purchased an Apple //c. The //c disassembler supports 65C02 disassembly and I was eager to see how Apple handled the modifications. He allowed me to examine his //c's ROM code.

Both versions were similar (no surprise?), but I had used a "decrement opcode" trick for the new zero page indirect instructions and they had used an "overlapping table" trick.

I might have been content to suffer the loss of lo-res but then I realized that the //c doesn't do a controller card slot search (because it has no slots). Requiring a disk controller in slot 6 seemed to be a much smaller inconvenience than losing lo-res. It yielded just enough space to do the job. The modifications described below are a blend of all these ideas. But first, let's take a look at the original code.

The Monitor's Disassembler

The disassembler resides in the F8 ROM and provides an excellent example of efficient machine language programming with tables to tackle a tough task in a minimum of space. Here is a short description of how it works. You can follow along in Appendix C of the Apple Reference Manual.

The monitor's command processor picks up your typed command (eg. 800L) and places the address into PCL and PCH ($3A and $3B). When it finds the "L" in your command it transfers control to LIST, the disassembler controller ($FE5E). LIST calls INSTDSP ($F8D0) repeatedly until a screenful of disassembled lines is completed after which execution returns to the monitor.

INSTDSP causes a single disassembled line to be printed. In order to accomplish this, it first calls INSDS1 ($F882). INSDS1 prints the

address of the opcode being disassembled (pointed by PCL, PCH) and then continues into INSDS2 ($F88C).

INSDS2 is the code responsible for determining 1) whether the byte is actually a legal opcode, 2) what its address mode and mnemonic are and 3) how many bytes of address are used by the instruction.

After analyzing the opcode byte, INSDS2 returns with the LENGTH ($2F), an address field print FORMAT ($2E) and, in the accumulator, an index into a table of mnemonics.

With this information, INSTDSP completes the one line disassembly by 1) printing the opcode and address bytes (if any), 2) retrieving the mnemonic characters from a table and printing them and 3) printing the formatted address field (if any). Execution then returns to the disassembler controller to repeat the operation if needed.

Revising the Code

As you might expect, it will be necessary to modify the code contained in the F8 ROM to handle the new opcodes. Refer to COMPUTIST No. 6 and 19 if you are new on the modified ROM scene.

The revisions detailed here may be applied to either the NEW F8 ROM described in Issue 19 or, with a minor change, to the AUTOSTART ROM that comes with an Apple][Plus. I used only the original 6502's instruction set, so it is not necessary to have a 65C02 installed to use the extended disassembler. The changes primarily consume the space used to support initialization of the annunciator outputs on reset and the disk controller slot search.

Proceed as follows:

1) Boot DOS and move a copy of the AUTOSTART or the Issue 19 NEW F8 ROM into RAM (or BLOAD an F8 ROM image from disk at $2800).

CALL-151
2800<F800.FFFFM

2) Enter the following hex listings:

Modified   INSDS2

```
288C: A1 3A A8 4A              $8289
2890: 90 05 6A B0 0C 29 87 4A  $727F
2898: AA BD 62 F9 20 79 F8 D0  $A30B
28A0: 03 A9 0F A8 AA BD A6 F9  $ACF5
28A8: 85 2E 29 03 85 2F 98 29  $C563
28B0: 1F 20 B7 FA F0 18 29 8F  $08DA
28B8: AA 98 A0 03 E0 8A F0 0B  $A16F
28C0: 4A 90 08 4A 4A 09 20 88  $F6A4
28C8: D0 FA C8 88 D0 F2 0A 60  $A3B2
```

Modified Tables

```
2962: FD A5 0B A6 0C 20        $122C
2968: 41 F9 20 70 29 4C 8E FD  $B9B0
2970: A9 BC 20 ED FD A9 AD 20  $33F7
2978: ED FD A9 BE 20 ED FD 60  $6DA5
2980: A4 FD A6 FC 20 8E FD 20  $8D9F
2988: 40 F9 A0 00 A9 BA 4C ED  $4BA7
2990: FD A5 FC C5 FE A5 FD E5  $7C87
2998: FF E6 FC D0 02 E6 FD 60  $D46B
```

```
05-   ZIY   .EQ $5   ($nn),Y
06-   ZPX   .EQ $6   $nn,X
07-   ABX   .EQ $7   $nnnn,X
08-   ABY   .EQ $8   $nnnn,Y
09-   ABI   .EQ $9   ($nnnn)
0A-   ZPY   .EQ $A   $nn,Y
0B-   REL   .EQ $B   relative
0C-   ZPI   .EQ $C   ($nn)
0D-   AXI   .EQ $D   ($nnnn,X)
0F-   IMP   .EQ $F   implied
0F-   ACC   .EQ $F   accumulator

* ---------------------------------------------------------------- *
      FMT1
* ---------------------------------------------------------------- *
*                 PACKED TABLE OF OPCODE                           *
*                 ADDRESS MODE CODES                               *
*           Codes are packed two to a byte.                        *
*      The address modes for each of the even opcodes              *
*      are treated first.  The code for the larger-valued          *
*      opcode goes into the high nibble, and the code for          *
*      the lower-valued opcode is placed in the low nibble.         *
*      The odd opcodes are placed at the end of the table.          *
*      Since only eight addressing modes are used by the odd        *
*   opcodes and since they follow a repetitive pattern, only        *
*                 four bytes are needed.                           *
* ---------------------------------------------------------------- *
*            Macro to calculate format bytes                       *
      .MA FMT
      .DA #16*]2+]1
      .EM
* ---------------------------------------------------------------- *

F962: 0F    >FMT IMP,ERR    opcodes 00 and 02
F963: 22    >FMT ZPG,ZPG    opcodes 04 and 06
F964: FF    >FMT IMP,ACC    opcodes 08 and 0A
F965: 33    >FMT ABS,ABS    opcodes 0C and 0E
F966: CB    >FMT REL,ZPI    opcodes 10 and 12
F967: 62    >FMT ZPG,ZPX    opcodes 14 and 16
F968: FF    >FMT IMP,ACC    opcodes 18 and 1A
F969: 73    >FMT ABS,ABX    opcodes 1C and 1E
F96A: 03    >FMT ABS,ERR    opcodes 20 and 22
F96B: 22    >FMT ZPG,ZPG    opcodes 24 and 26
F96C: FF    >FMT IMP,ACC    opcodes 28 and 2A
F96D: 33    >FMT ABS,ABS    opcodes 2C and 2E
F96E: CB    >FMT REL,ZPI    opcodes 30 and 32
F96F: 66    >FMT ZPX,ZPX    opcodes 34 and 36
F970: FF    >FMT IMP,ACC    opcodes 38 and 3A
F971: 77    >FMT ABX,ABX    opcodes 3C and 3E
F972: 0F    >FMT IMP,ERR    opcodes 40 and 42
F973: 20    >FMT ERR,ZPG    opcodes 44 and 46
F974: FF    >FMT IMP,ACC    opcodes 48 and 4A
F975: 33    >FMT ABS,ABS    opcodes 4C and 4E
F976: CB    >FMT REL,ZPI    opcodes 50 and 52
F977: 60    >FMT ERR,ZPX    opcodes 54 and 56
F978: FF    >FMT IMP,IMP    opcodes 58 and 5A
F979: 70    >FMT ERR,ABX    opcodes 5C and 5E
F97A: 0F    >FMT IMP,ERR    opcodes 60 and 62
F97B: 22    >FMT ZPG,ZPG    opcodes 64 and 66
F97C: FF    >FMT IMP,ACC    opcodes 68 and 6A
F97D: 39    >FMT ABI,ABS    opcodes 6C and 6E
F97E: CB    >FMT REL,ZPI    opcodes 70 and 72
F97F: 66    >FMT ZPX,ZPX    opcodes 74 and 76
F980: FF    >FMT IMP,IMP    opcodes 78 and 7A
F981: 7D    >FMT AXI,ABX    opcodes 7C and 7E
F982: 0B    >FMT REL,ERR    opcodes 80 and 82
F983: 22    >FMT ZPG,ZPG    opcodes 84 and 86
F984: FF    >FMT IMP,IMP    opcodes 88 and 8A
F985: 33    >FMT ABS,ABS    opcodes 8C and 8E
F986: CB    >FMT REL,ZPI    opcodes 90 and 92
```

```
F987: A6    >FMT ZPX,ZPY    opcodes 94 and 96
F988: FF    >FMT IMP,IMP    opcodes 98 and 9A
F989: 73    >FMT ABS,ABX    opcodes 9C and 9E
F98A: 11    >FMT IMM,IMM    opcodes A0 and A2
F98B: 22    >FMT ZPG,ZPG    opcodes A4 and A6
F98C: FF    >FMT IMP,IMP    opcodes A8 and AA
F98D: 33    >FMT ABS,ABS    opcodes AC and AE
F98E: CB    >FMT REL,ZPI    opcodes B0 and B2
F98F: A6    >FMT ZPX,ZPY    opcodes B4 and B6
F990: FF    >FMT IMP,IMP    opcodes B8 and BA
F991: 87    >FMT ABX,ABY    opcodes BC and BE
F992: 01    >FMT IMM,ERR    opcodes C0 and C2
F993: 22    >FMT ZPG,ZPG    opcodes C4 and C6
F994: FF    >FMT IMP,IMP    opcodes C8 and CA
F995: 33    >FMT ABS,ABS    opcodes CC and CE
F996: CB    >FMT REL,ZPI    opcodes D0 and D2
F997: 60    >FMT ERR,ZPX    opcodes D4 and D6
F998: FF    >FMT IMP,IMP    opcodes D8 and DA
F999: 70    >FMT ERR,ABX    opcodes DC and DE
F99A: 01    >FMT IMM,ERR    opcodes E0 and E2
F99B: 22    >FMT ZPG,ZPG    opcodes E4 and E6
F99C: FF    >FMT IMP,IMP    opcodes E8 and EA
F99D: 33    >FMT ABS,ABS    opcodes EC and EE
F99E: CB    >FMT REL,ZPI    opcodes F0 and F2
F99F: 60    >FMT ERR,ZPX    opcodes F4 and F6
F9A0: FF    >FMT IMP,IMP    opcodes F8 and FA
F9A1: 70    >FMT ERR,ABX    opcodes FC and FE
F9A2: 24    >FMT ZXI,ZPG    opcodes x1 and x5, x even
F9A3: 31    >FMT IMM,ABS    opcodes x9 and xD, x even
F9A4: 65    >FMT ZIY,ZPX    opcodes x1 and x5, x odd
F9A5: 78    >FMT ABY,ABX    opcodes x9 and xD, x odd

* ------------------------------------------------------------------ *
      FMT2
* Address mode print format and instruction length                  *
* packed as follows:                                                 *
* lead-in chars.lead-out chars.length                                *
*                                                                    *
*   $ ($ #$      ,X  ) ,Y     0,1,2                                  *
*   ! ! !        !  !  !       -+-                                    *
*   \ ! /        \  !  /        !                                     *
*    xxx           yyy         zz                                     *
*                                                                    *
*          A "1" indicates the particular character (or pair)        *
*          should be printed, a "0" indicates not to print           *
*          Each bit is tested left-to-right and characters           *
*             printed as required, pausing to print the              *
*          address between the lead-in and lead-out chars.           *
*          For example 01011001 ($59) represents an address field    *
*         format of ($nn,X), zero page indexed indirect addressing.  *
*       Length is coded in zz. This is the number of bytes of address *
*                     for the instruction.                           *
*                                                                    *
*                 xxx yyy zz  address mode     FMT1 code             *
* ------------------------------------------------------------------ *
F9A6: 00         .DA #%000.000.00   ERROR          0
F9A7: 21         .DA #%001.000.01   IMMEDIATE      1
F9A8: 81         .DA #%100.000.01   ZEROPAGE       2
F9A9: 82         .DA #%100.000.10   ABSOLUTE       3
F9AA: 59         .DA #%010.110.01   (ZEROPAGE,X)   4
F9AB: 4D         .DA #%010.011.01   (ZEROPAGE),Y   5
F9AC: 91         .DA #%100.100.01   ZEROPAGE,X     6
F9AD: 92         .DA #%100.100.10   ABSOLUTE,X     7
F9AE: 86         .DA #%100.001.10   ABSOLUTE,Y     8
F9AF: 4A         .DA #%010.010.10   (ABSOLUTE)     9
F9B0: 85         .DA #%100.001.01   ZEROPAGE,Y     A
F9B1: 9D         .DA #%100.111.01   RELATIVE       B
F9B2: 49         .DA #%010.010.01   (ZEROPAGE)     C
F9B3: 5A         .DA #%010.110.10   (ABSOLUTE,X)   D
* ------------------------------------------------------------------ *
F9B4: D9    CHAR2 .AS -"Y"    Lead-out characters for address field
```

```
29A0: FF 70 24 31 65 78 00 21    $5FAF
29A8: 81 82 59 4D 91 92 86 4A    $55CF

29B0: 85 9D 49 5A D9 00 D8 A4    $2659
29B8: A4 00 AC A9 AC A3 A8 A4    $7CF4
29C0: 1C D8 8A 62 1C 5A 23 48    $6439
29C8: 5D 26 8B 62 1B 94 A1 88    $F46B
29D0: 9D 54 8A 44 1D C8 23 54    $3668
29D8: 9D 68 8B 44 1D E8 A1 94    $F342
29E0: 1C C4 29 B4 19 08 AE 84    $B412
29E8: 69 74 A8 B4 19 28 23 6E    $8D76
29F0: 24 74 53 F4 1B CC 23 4A    $0CCC
29F8: 24 72 53 F2 19 A4 A1 8A    $0B14

2A00: AD 06 1A AA 5B A2 5B A2    $AAEF
2A08: A5 74 69 74 24 74 24 72    $9844
2A10: AE 44 AE 68 A8 B2 AD 32    $42C5
2A18: 29 B2 8A 72 7C 22 8B 72    $BE7D
2A20: 15 1A 9C 1A 6D 26 9C 26    $AEF1
2A28: A5 72 69 72 29 88 53 C8    $D36F
2A30: 84 C4 13 CA 34 26 11 48    $F821
2A38: A5 44 69 44 23 A2 A0 C8    $C589
```

Table Extension

```
2A6F: AD                         $59B9
2A70: FF CF 90 0A 00 00 AC C6    $BDD3
2A78: A5 76 8A 74 8B 74          $5B28
```

XPTNOP

```
2AB4: 4C 00 C6 20                $65C4
2AB8: 02 FB 98 A2 0C CA 30 0A    $DFA2
2AC0: DD CB FA D0 F8 BD`EA FE    $6BC7
2AC8: A0 00 60 14 1A 1C 3A 5A    $055C
2AD0: 64 74 7A 89 9C 9E 0F       $4C58
```

CHKIND

```
2B02: C9 12 D0 01 88 60          $59E9
2B08: FF                         $A4A6

28F1: C1 F9

2919: B9

291F: B3
```

3a) If you are modifying the AUTOSTART
ROM enter the following:

```
2AC6: B3 FB

2BB3: 5B 37 5B 36 5D             $5688
2BB8: 5C 5C 5E 21 5C 5C 5A       $9C75
```

3b) If you are modifying the Issue 19 NEW F8
ROM enter the following:

```
2EEA: 5B 37 5B 36 5D 5C          $5B2B
2EF0: 5C 5E 21 5C 5C 5A 20       $CE19

2FE6: C4

2FDC: B2
```

```
2FDE:  B2

2FF3:  C4

2FF5:  C4
```

4) Save the new version of the ROM to disk.

**BSAVE
F8.W/C02.DISASM,A$2800,L$800**

5) Use this file to program an EPROM to replace your current F8 ROM.

How it Works

The accompanying source listings describe much of the detail of the new code including the packed data table structures used by the disassembler. To summarize, here is an overview of the modifications:

The INSDS2 code had to be changed to accept the new opcodes and provide the correct length, format and mnemonic for INSTDSP. Some programs call INSDS2 directly to find three-byte instructions which require adjustment during relocation, so the standard entry point ($F88C) was left unchanged.

INSDS2 first screens out all opcodes ending in 3, 7, B, or F since they are not legal opcodes. In the process of being tested, the opcode is modified to become an index into table FMT1. This table contains codes which identify the address mode for the opcode. These codes are packed two to a byte for efficiency and are unpacked by the code at SCRN2.

A code of zero indicates that, although the opcode passed the first test, it is nonetheless an illegal opcode. Normally, all the new 65C02 opcodes end up pointing to FMT1 entries holding zeroes. However, we have revised these entries to indicate the appropriate address mode codes for the new instructions.

If a non-zero address mode code is obtained from table FMT1, execution branches to GETFMT. Bad codes drop through to ERR where they are given a default implied address mode (length of 1, no address field characters), and are changed to an $0F opcode to force a ''???'' mnemonic.

Non-zero address mode codes are used to index into table FMT2. This table contains address field print format and instruction length for the address mode. The retrieved byte is stored in FORMAT ($2E) and the 2 least significant bits (the number of bytes of address for the instruction) are stripped off and placed in LENGTH ($2F).

The last step is to develop the index into the mnemonic tables. The three letter mnemonics are coded into two bytes now arranged as a table of consecutive pairs instead of the two separate tables used by the original disassembler. Subroutines XPTNOP and CHKIND handle the new opcodes. CHKIND intercepts the new zero

```
F9B5: 00                .HS 00        00 is skipflag for address field and
*                                      doubles as implied, accumulator and      *
*                                      error FMT2 entry (address mode code $0F)  *
F9B6: D8 A4 A4          .AS -"X$$"
F9B9: 00                .HS 00
F9BA: AC A9 AC
F9BD: A3 A8 A4  CHAR1   .AS -",),#($"  Lead-in characters for address field
* ------------------------------------------------------------------------------ *
* Packed mnemonic letters, 5 bits per letter                                     *
* ,total 15 bits + 1 unused (0) arranged                                         *
* as     xxxxxyyy yyzzzzz0                                                        *
*        MNEML     MNEMR                                                          *
*             Add $BF to obtain character's ASCII code (msb set)                 *
*
*             Example: mnemonic TRB has MNEML byte $AC and                       *
*             MNEMR byte $C6.  In binary form $ACC6 is                           *
*                     1010110011000110                                           *
*             The first five bits are 10101 or $16.  Adding $BF                  *
*             produces $D5, the code for "T".  The next five bits                *
*             are 10011 or $13.  $13 + $BF = $D2, code for "R", etc.             *
* ------------------------------------------------------------------------------ *
*                  Macro to generate bytes from characters                       *
* ------------------------------------------------------------------------------ *

00-    X      .SE 0
01-    Y      .SE 1
02-    Z      .SE 2
              .MA MNEM
       X      .SE "]1"-"?"*2048
       Y      .SE "]2"-"?"*64
       Z      .SE "]3"-"?"*2
              .DA /X+Y+Z
              .DA #X+Y+Z
              .EM
* ------------------------------------------------------------------------------ *
F9C0- MNEML  >MNEM B,R,K
F9C0: 1C      >          .DA /X+Y+Z
F9C1: D8      >          .DA #X+Y+Z
F9C1- MNEMR  .EQ *-1
F9C2-        >MNEM P,H,P
F9C2: 8A      >          .DA /X+Y+Z
F9C3: 62      >          .DA #X+Y+Z
F9C4-        >MNEM B,P,L
F9C4: 1C      >          .DA /X+Y+Z
F9C5: 5A      >          .DA #X+Y+Z
F9C6-        >MNEM C,L,C
F9C6: 23      >          .DA /X+Y+Z
F9C7: 48      >          .DA #X+Y+Z
F9C8-        >MNEM J,S,R
F9C8: 5D      >          .DA /X+Y+Z
F9C9: 26      >          .DA #X+Y+Z
F9CA-        >MNEM P,L,P
F9CA: 8B      >          .DA /X+Y+Z
F9CB: 62      >          .DA #X+Y+Z
F9CC-        >MNEM B,M,I
F9CC: 1B      >          .DA /X+Y+Z
F9CD: 94      >          .DA #X+Y+Z
F9CE-        >MNEM S,E,C
F9CE: A1      >          .DA /X+Y+Z
F9CF: 88      >          .DA #X+Y+Z
F9D0-        >MNEM R,T,I
F9D0: 9D      >          .DA /X+Y+Z
F9D1: 54      >          .DA #X+Y+Z
F9D2-        >MNEM P,H,A
F9D2: 8A      >          .DA /X+Y+Z
F9D3: 44      >          .DA #X+Y+Z
F9D4-        >MNEM B,V,C
F9D4: 1D      >          .DA /X+Y+Z
F9D5: C8      >          .DA #X+Y+Z
F9D6-        >MNEM C,L,I
F9D6: 23      >          .DA /X+Y+Z
```

```
F9D7: 54        >         .DA #X+Y+Z
F9D8-        >MNEM R,T,S
F9D8: 9D        >         .DA /X+Y+Z
F9D9: 68        >         .DA #X+Y+Z
F9DA-        >MNEM P,L,A
F9DA: 8B        >         .DA /X+Y+Z
F9DB: 44        >         .DA #X+Y+Z
F9DC-        >MNEM B,V,S
F9DC: 1D        >         .DA /X+Y+Z
F9DD: E8        >         .DA #X+Y+Z
F9DE-        >MNEM S,E,I
F9DE: A1        >         .DA /X+Y+Z
F9DF: 94        >         .DA #X+Y+Z
F9E0-        >MNEM B,R,A
F9E0: 1C        >         .DA /X+Y+Z
F9E1: C4        >         .DA #X+Y+Z
F9E2-        >MNEM D,E,Y
F9E2: 29        >         .DA /X+Y+Z
F9E3: B4        >         .DA #X+Y+Z
F9E4-        >MNEM B,C,C
F9E4: 19        >         .DA /X+Y+Z
F9E5: 08        >         .DA #X+Y+Z
F9E6-        >MNEM T,Y,A
F9E6: AE        >         .DA /X+Y+Z
F9E7: 84        >         .DA #X+Y+Z
F9E8-        >MNEM L,D,Y
F9E8: 69        >         .DA /X+Y+Z
F9E9: 74        >         .DA #X+Y+Z
F9EA-        >MNEM T,A,Y
F9EA: A8        >         .DA /X+Y+Z
F9EB: B4        >         .DA #X+Y+Z
F9EC-        >MNEM B,C,S
F9EC: 19        >         .DA /X+Y+Z
F9ED: 28        >         .DA #X+Y+Z
F9EE-        >MNEM C,L,V
F9EE: 23        >         .DA /X+Y+Z
F9EF: 6E        >         .DA #X+Y+Z
F9F0-        >MNEM C,P,Y
F9F0: 24        >         .DA /X+Y+Z
F9F1: 74        >         .DA #X+Y+Z
F9F2-        >MNEM I,N,Y
F9F2: 53        >         .DA /X+Y+Z
F9F3: F4        >         .DA #X+Y+Z
F9F4-        >MNEM B,N,E
F9F4: 1B        >         .DA /X+Y+Z
F9F5: CC        >         .DA #X+Y+Z
F9F6-        >MNEM C,L,D
F9F6: 23        >         .DA /X+Y+Z
F9F7: 4A        >         .DA #X+Y+Z
F9F8-        >MNEM C,P,X
F9F8: 24        >         .DA /X+Y+Z
F9F9: 72        >         .DA #X+Y+Z
F9FA-        >MNEM I,N,X
F9FA: 53        >         .DA /X+Y+Z
F9FB: F2        >         .DA #X+Y+Z
F9FC-        >MNEM B,E,Q
F9FC: 19        >         .DA /X+Y+Z
F9FD: A4        >         .DA #X+Y+Z
F9FE-        >MNEM S,E,D
F9FE: A1        >         .DA /X+Y+Z
F9FF: 8A        >         .DA #X+Y+Z
FA00-        >MNEM T,S,B
FA00: AD        >         .DA /X+Y+Z
FA01: 06        >         .DA #X+Y+Z
FA02-        >MNEM B,I,T
FA02: 1A        >         .DA /X+Y+Z
FA03: AA        >         .DA #X+Y+Z
FA04-        >MNEM J,M,P
FA04: 5B        >         .DA /X+Y+Z
FA05: A2        >         .DA #X+Y+Z
FA06-        >MNEM J,M,P
```

```
FA06: 5B        >         .DA /X+Y+Z
FA07: A2        >         .DA #X+Y+Z
FA08-        >MNEM S,T,Y
FA08: A5        >         .DA /X+Y+Z
FA09: 74        >         .DA #X+Y+Z
FA0A-        >MNEM L,D,Y
FA0A: 69        >         .DA /X+Y+Z
FA0B: 74        >         .DA #X+Y+Z
FA0C-        >MNEM C,P,Y
FA0C: 24        >         .DA /X+Y+Z
FA0D: 74        >         .DA #X+Y+Z
FA0E-        >MNEM C,P,X
FA0E: 24        >         .DA /X+Y+Z
FA0F: 72        >         .DA #X+Y+Z
FA10-        >MNEM T,X,A
FA10: AE        >         .DA /X+Y+Z
FA11: 44        >         .DA #X+Y+Z
FA12-        >MNEM T,X,S
FA12: AE        >         .DA /X+Y+Z
FA13: 68        >         .DA #X+Y+Z
FA14-        >MNEM T,A,X
FA14: A8        >         .DA /X+Y+Z
FA15: B2        >         .DA #X+Y+Z
FA16-        >MNEM T,S,X
FA16: AD        >         .DA /X+Y+Z
FA17: 32        >         .DA #X+Y+Z
FA18-        >MNEM D,E,X
FA18: 29        >         .DA /X+Y+Z
FA19: B2        >         .DA #X+Y+Z
FA1A-        >MNEM P,H,X
FA1A: 8A        >         .DA /X+Y+Z
FA1B: 72        >         .DA #X+Y+Z
FA1C-        >MNEM N,O,P
FA1C: 7C        >         .DA /X+Y+Z
FA1D: 22        >         .DA #X+Y+Z
FA1E-        >MNEM P,L,X
FA1E: 8B        >         .DA /X+Y+Z
FA1F: 72        >         .DA #X+Y+Z
FA20-        >MNEM A,S,L
FA20: 15        >         .DA /X+Y+Z
FA21: 1A        >         .DA #X+Y+Z
FA22-        >MNEM R,O,L
FA22: 9C        >         .DA /X+Y+Z
FA23: 1A        >         .DA #X+Y+Z
FA24-        >MNEM L,S,R
FA24: 6D        >         .DA /X+Y+Z
FA25: 26        >         .DA #X+Y+Z
FA26-        >MNEM R,O,R
FA26: 9C        >         .DA /X+Y+Z
FA27: 26        >         .DA #X+Y+Z
FA28-        >MNEM S,T,X
FA28: A5        >         .DA /X+Y+Z
FA29: 72        >         .DA #X+Y+Z
FA2A-        >MNEM L,D,X
FA2A: 69        >         .DA /X+Y+Z
FA2B: 72        >         .DA #X+Y+Z
FA2C-        >MNEM D,E,C
FA2C: 29        >         .DA /X+Y+Z
FA2D: 88        >         .DA #X+Y+Z
FA2E-        >MNEM I,N,C
FA2E: 53        >         .DA /X+Y+Z
FA2F: C8        >         .DA #X+Y+Z
FA30-        >MNEM O,R,A
FA30: 84        >         .DA /X+Y+Z
FA31: C4        >         .DA #X+Y+Z
FA32-        >MNEM A,N,D
FA32: 13        >         .DA /X+Y+Z
FA33: CA        >         .DA #X+Y+Z
FA34-        >MNEM E,O,R
FA34: 34        >         .DA /X+Y+Z
FA35: 26        >         .DA #X+Y+Z
```

page indirect instructions (non-indexed) and decrements them to form the indirect indexed opcodes with the same mnemonics.

Some opcodes defy manipulation to form an index, so XPTNOP checks the opcode against a table of these codes and retrieves a corresponding index if a match is found.

If the opcode is not in the exception table, execution falls through to the original MNNDX routine which processes the opcode bits to form an index from $00 to $3F. In either case, the resulting index is doubled to suit the paired mnemonic table before returning.

The Test Code Generator

For those who would like to exercise the new disassembler capabilities, the following program will store 256 groups of the pattern "xx FF EF" where xx increments from $00 to $FF. Enter the code as follows:

Test Code Generator

```
0800: A9 00 85 2E A9 30 85 2F    $717C
0808: A0 00 A2 00 8A 91 2E E6    $99A1
0810: 2E D0 02 E6 2F A9 FF 91    $3FB8
0818: 2E E6 2E D0 02 E6 2F A9    $32F1
0820: EF 91 2E E6 2E D0 02 E6    $BDDA
0828: 2F E8 D0 E0 60             $6742
```

When ready, save the file to disk with

**BSAVE B.TEST CODE
GENERATOR,A$800,L$2D**

When you BRUN B.TEST CODE GENERATOR it will write the test pattern starting at address $3000. Disassembling $3000 to $32FF will let you see how the modified disassembler handles each possible opcode.

Conclusion

I have found the Apple's built-in disassembler an invaluable learning tool, a useful debugging aid and an essential code-tracing partner. It seems only fitting that it keep pace with the very microprocessor that brings it to life.

Editors Note: The source code presented with this article cannot be assembled by any version of the S-C Assembler before version 2.0.

Due to the extremely large size of the six source code listings accompanying this article, they are one-after-another all through this article. The source code is the boxed portion of this article.

To save space, the source code on this page is printed as two one-column listings.

Also, the next page contains the final piece of this source code listing.

```
FA36-        >MNEM A,D,C
FA36: 11     >        .DA /X+Y+Z
FA37: 48     >        .DA #X+Y+Z
FA38-        >MNEM S,T,A
FA38: A5     >        .DA /X+Y+Z
FA39: 44     >        .DA #X+Y+Z
FA3A-        >MNEM L,D,A
FA3A: 69     >        .DA /X+Y+Z
FA3B: 44     >        .DA #X+Y+Z
FA3C-        >MNEM C,M,P
FA3C: 23     >        .DA /X+Y+Z
FA3D: A2     >        .DA #X+Y+Z
FA3E-        >MNEM S,B,C
FA3E: A0     >        .DA /X+Y+Z
FA3F: C8     >        .DA #X+Y+Z
```

```
*           Extension of mnemonic table           *

          .OR $FA6F On top of INITAN
          .TF OBJ.TABLE EXTENSION

* --------------------------------------------- *
00-  X    .SE 0
01-  Y    .SE 1
02-  Z    .SE 2
CFFF- CLRROM .EQ $CFFF
* --------------------------------------------- *

*       Macro to generate bytes from characters       *
          .MA MNEM
X         .SE "]1"-"?"*2048
Y         .SE "]2"-"?"*64
Z         .SE "]3"-"?"*2
          .DA /X+Y+Z
          .DA #X+Y+Z
          .EM

* --------------------------------------------- *
FA6F: AD FF CF        LDA CLRROM Turn off extension ROM for RESET
FA72: 90 0A           BCC END skip over table
FA74-        >MNEM ?,?,?
FA74: 00     >        .DA /X+Y+Z
FA75: 00     >        .DA #X+Y+Z
FA76-        >MNEM T,R,B
FA76: AC     >        .DA /X+Y+Z
FA77: C6     >        .DA #X+Y+Z
FA78-        >MNEM S,T,Z
FA78: A5     >        .DA /X+Y+Z
FA79: 76     >        .DA #X+Y+Z
FA7A-        >MNEM P,H,Y
FA7A: 8A     >        .DA /X+Y+Z
FA7B: 74     >        .DA #X+Y+Z
FA7C-        >MNEM P,L,Y
FA7C: 8B     >        .DA /X+Y+Z
FA7D: 74     >        .DA #X+Y+Z
FA7E: END    .EQ *
C600- BOOT .EQ $C600     standard drive boot address
FB02- CHKIND .EQ $FB02   code to test for zp indirect codes
FEEA- XPTNDX .EQ $FEEA   indexes for exceptional codes

* --------------------------------------------- *
          .OR $FAB4          after SETPLP in reset handler
          .TF OBJ.XPTNOP
* --------------------------------------------- *

FAB4: 4C 00 C6       JMP BOOT    boot on cold start
FAB7: 20 02 FB  XPTNOP JSR CHKIND handle zp indirect codes first
FABA: 98             TYA         recall opcode
FABB: A2 0C          LDX #$0C    twelve opcodes to check for
FABD: CA     .1      DEX
```

```
FABE: 30 0A       BMI .2       exit if no matchups
FAC0: DD CB FA    CMP OPTBL,X  check opcode against table
FAC3: D0 F8       BNE .1       no match, try next entry
FAC5: BD EA FE    LDA XPTNDX,X matches, pickup
                               corresponding mnemonic
                               index
FAC8: A0 00       LDY #$00     must exit with Y = 0
FACA: 60     .2   RTS

* --------------------------------------------- *
FACB: 14     OPTBL .HS 14    TRB zpg
FACC: 1A           .HS 1A    INC acc
FACD: 1C           .HS 1C    TRB abs
FACE: 3A           .HS 3A    DEC acc
FACF: 5A           .HS 5A    PHY imp
FAD0: 64           .HS 64    STZ zpg
FAD1: 74           .HS 74    STZ zpg,X
FAD2: 7A           .HS 7A    PLY imp
FAD3: 89           .HS 89    BIT imm
FAD4: 9C           .HS 9C    STZ abs
FAD5: 9E           .HS 9E    STZ abs,X
FAD6: 0F           .HS 0F    ??? code
```

```
*            CHECK ZP INDIRECT ROUTINE            *

          .OR $FB02          on top of old DISKID
          .TF OBJ.CHKIND

* --------------------------------------------- *
FB02: C9 12    CHKIND CMP #$12   is it one of the new
                                 indirect opcodes?
FB04: D0 01           BNE PT1    no, exit
FB06: 88              DEY        yes, convert to
                                 indirect,Y code with same
                                 mnemonic
FB07: 60       PT1    RTS
FB08: FF              .HS FF     filler byte
```

```
* Table of mnemonic indexes for exceptional 65C02 opcodes *

          .OR $FEEA     use RDORWR space
          .TF OBJ.XPTNDX

* --------------------------------------------- *
21-  BIT  .EQ $21    Mnemonic index for BIT
36-  DEC  .EQ $36    Mnemonic index for DEC
37-  INC  .EQ $37    Mnemonic index for INC
5A-  ERR  .EQ $5A    Mnemonic index for ???
5B-  TRB  .EQ $5B    Mnemonic index for TRB
5C-  STZ  .EQ $5C    Mnemonic index for STZ
5D-  PHY  .EQ $5D    Mnemonic index for PHY
5E-  PLY  .EQ $5E    Mnemonic index for PLY
* --------------------------------------------- *

FEEA: 5B           .DA #TRB
FEEB: 37           .DA #INC
FEEC: 5B           .DA #TRB
FEED: 36           .DA #DEC
FEEE: 5D           .DA #PHY
FEEF: 5C           .DA #STZ
FEF0: 5C           .DA #STZ
FEF1: 5E           .DA #PLY
FEF2: 21           .DA #BIT
FEF3: 5C           .DA #STZ
FEF4: 5C           .DA #STZ
FEF5: 5A           .DA #ERR
```

softkey for...

Uptown Trivia

by Phil Pattengale

> *Uptown Software*

Requirements:
4 blank disks (or 2 double-sided)
FID and COPYA

In first looking at this disk, it seemed to be unprotected. I could CATALOG it. I was able to make a COPYA version by modifying DOS so it would ignore errors. I tried to boot the COPYAed version; however, the protection reared its ugly head, and the program crashed. I decided to start my search through the code with the file UPTOWN TRIVIA since it made logical sense that it would be the hello file. (Also, if you boot a sector editor and look at track $01, sector $07 you'll see the hello filename there.)

After looking through the file's code, starting at $0801, I passed a few JSR's and some slick code having nothing to do with the protection (more on that below). Finally, at $0872, I encountered a JMP to $093D. Looking here I found a JSR to $0878. When I looked at $0878 I did a real quick double take. The code was incredibly self-modifying. Each line changes the start of the following line, until it gets to some disk accessing code. I decided that this was my routine that did the disk check, and so I simply put NOP commands in place of the JSR opcode. (NOP-No OPeration opcodes). I then set the screen back to graphics mode with the appropriate pokes ($C050, $C052, $C054) and started the program back up with a $801G (which I couldn't see because I was in graphics

mode) command and guess what? It started right up and ran beautifully!

Later experimenting showed that it could be copied with FID, but the DOS is what turns on the hi-res screen for the game. An interim Hello program is needed to do that without their DOS.

On with the softkey

So, in order to softkey Uptown Trivia, you must:

1) Type in the following program.

```
10   HGR : POKE -16302,0
20   PRINT CHR$(4);"BRUN UPTOWN TRIVIA"
```

2) Initialize a blank disk with the above as the HELLO program.

INIT HELLO

3) Copy all the files from Uptown Trivia Side #1 to your new disk using FID or a similar file copy program.

4) Copy sides 2-4 with plain old COPYA.
And that's all there is to it!

Oh, by the way...

This file loads at $801 and proceeds to do something I had never thought of doing. The file does its DOS commands by doing a JSR to a routine that reads the stack to find out where it was called from, and proceeds to "print" the string of characters immediately following the calling JSR. DOS will intercept the characters as if they were printed from BASIC or typed, and perform the DOS command if it starts with a Ctrl-D ($84) and ends with a Ctrl-M or Return ($8D). The subroutine continues to print the characters until it finds a $00, at which point it will set the JSR's return address in the stack to the location of the $00 and return as if nothing happened!

Let me illustrate that with a simplified example...

```
$800-   JSR $0900    ;Call print routine.
$803-   .HS 84C3C1D4C1CCCFC78D00
        "ctrlD CATALOG ctrlM"+$00
$80D-   JMP $03D3    ;coldstart DOS
-------------------------------------------
$900-   PLA          ;Get lobyte of return
$901-   TAX          ; addr and save it in X.
$902-   PLA          ;Get hibyte and put
$903-   TAY          ; it in Y.
$904-   INX          ;Point X to next byte.
$905-   LDA $0800,X  ;Load Accumulator with
                     ; what is at $0800+X.
$908-   BEQ $0910    ;If A = 0 then branch
                     ; to the exit.
$90A-   JSR $FDED    ;Outputs character in
                     ; A to the screen.
$90D-   JMP $0904    ;Continue the loop.
$910-   TYA          ;Move the Y index
$911-   PHA          ; (hibyte) back
                     ; onto the stack.
$912-   TXA          ;Move the latest X
$913-   PHA          ; value (new lobyte)
                     ; back onto the stack.
$914-   RTS          ;return to caller
```

Now, when this program enters the subroutine, the stack has the values $08 and $02 in it (the return address). The "print" routine uses this in order to find the text string. After the printing is done, its counter is at the location of the $00 ($080C) so at the end of the "print" routine, a $0C and a $08 are pushed back onto the stack. When the 6502 gets the RTS at $0914, it adds one to the address it finds on the stack ($80C) and continues the program at $080D. In this example we just exit to BASIC instead.

Well, I found this a nifty little way to use DOS from machine language so I thought I'd pass it on to you.

Softkey for ...

by Randy Ramirez

CBS Software
One Fawcett Pl.
Greenwich, CT 06386

Requirements:
48K Apple][
DOS 3.3
FID
A blank disk

Nothing much was happening at the office. I wanted to call it a day. Suddenly, the phone rang. Someone just reported a murder. It seems as though a homicide detective never gets a break. Pity. The heat's on. I've got to search for clues, gather evidence, and hopefully close the case. That's the object of Murder by the Dozen.

Upon booting up the disk, one can see the familiar ''] '' prompt. However, when I tried booting up a normal DOS 3.3 disk, modified DOS to ignore read errors, and tried CATALOGing Murder by the Dozen, I got the catalog. From this, I concluded that it is a relatively normal DOS.

Further investigation of the disk reveals altered data and address end marks. When coming across a problem that involves files and just altered epilogues, the most logical step is to tell DOS not to read the epilogues and copy

the files to a blank disk with FID.

I found that it is best to tell DOS not to read the epilogues rather than tell it to ignore all kinds of errors. This has the advantage of still checking the integrity of the data while copying.

If you follow the ''read address field'' and ''read data field'' routines, they each end with a disk read, followed by a CMP (CoMPare) with $DE, then another read and CMP with $AA. If any bytes do not match, they go directly to $B942 and indicate an error. The trick is to tell DOS to stop reading at the beginning of the epilog and simply CLC (CLear Carry) and RTS (ReTurn from Subroutine).

The changes to do this are:

```
B92F:18 60
B98B:18 60
```

We can incorporate this into our softkey.

1) Initialize a blank disk and delete the HELLO program.

```
INIT HELLO
DELETE HELLO
```

2) Enter the monitor and alter DOS as described above.

```
CALL-151
B92F:18 60
B98B:18 60
```

3) BRUN FID and copy all the files from Murder by the Dozen to your freshly formatted disk.

That's all there is to it. You now own a deprotected version of Murder by the Dozen.

ENCHANTER

● Looking under lily pads can be helpful, as can going in shacks!
● Hungry? Try opening the oven.
● Spring water is very thirst-quenching.
● Looking under trees can mend torn things.
● Try ''rezrov'' on the egg.
● Pursuing strange noises in the library can be fatal!
● Talking to frogs can be very helpful.
● Try reading the old dusty book.
● Dreams can come true.
● Be careful when entering temples!

By Jesse Weissman

PLANETFALL

● To cross the ravine, DROP LADDER, EXTEND LADDER and PUT LADDER.

By Paul Wilson

WIZARD & PRINCESS

● Visit the tower in the Wizard's castle twice and be sure to have the ring.
● Copy the two notes on paper and put the glyphs one above the other.
● One of the snakes is good, help him.
● Can't cross the bridge with any of your load? The locket will help.
● Only one item from the fly by night peddler will be of use to you.
● It serves as a medieval doorbell.
As for the giant, think of the fairy tale about Jack.

By Paul Wilson

CRITICAL MASS

● Please don't forget the flowers, they will be needed at the airport.
● You must have the soup to stay alive.

By Debbie Holloway

TRANSYLVANIA

● You must eat garlic.
● Show cross.

By Ed Higgins

The end is near!

Were running out of

The Best of Hardcore Computing

This could be your LAST CHANCE to get your own copy of this rare collector's edition.

Just send in $19.95 check or money order (US funds only) to:
COMPUTIST
PO Box 110846-T
Tacoma, WA 98411

Washington state orders add 7.8% sales tax. Foreign orders add 20% shipping and handling. *VISA* *MasterCard* orders enclose signature and expiration date. OFFER GOOD ONLY WHILE SUPPLY LASTS!

Description of Available Back Issues

28 *Softkeys* | Ultima IV | Robot Odyssey | Rendezvous | Word Attack & Classmate | Three from Mindscape | Alpahabetic Keyboarding | Hacker | Disk Director | Lode Runner | MIDI/4 | *Feature* | Capturing the Hidden Anchon Editor | *Core* | Fingerprints Plus: A Review | Beneath Beyond Castle Wolfenstein (part 2) | ....

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

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

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

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

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

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

21 *Softkeys* | DB Master version 4+ | Dazzle Draw | Archon | Twerps | *Readers' Softkeys* | Advanced Blackjack | Megaworks | Summer Games | College Entrance Exam Prep | Applewriter revisited | *Features* | Demystifying The Quarter Track | *Core* | Proshadow: A ProDOS Disk Monitor...........................

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

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

18 *Softkeys* | Scholastic Version of Bank Street Writer | Applewriter //e | SSI's Non-RDOS Disks | *Readers' Softkeys* | BPI Accounting Programs and DesignWare Programs | *Features* | Installing a Free Sector Patch Into Applewriter //e | Simple Copy Protection | *Core* | The Games of 1984: In Review | 65C02 Chips Now Available | Checksoft v2 | ....

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

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

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

Locksmith 5.0 and Locksmith Programming Language..

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

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

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

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

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

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

Back issues not listed are no longer available.

But disks are still available for ALL sold-out issues !

Use the order form on the other side of this page

Writer's Guide

COMPUTIST

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

MANUSCRIPT MECHANICS

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

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

TITLE OF WORK
FULL NAME OF AUTHOR
ADDRESS
PHONE NUMBER

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

The article and any accompanying program **should be submitted as a standard text file on a DOS 3.3 disk.** Label the disk with the title of the work and the author's full name and address. On disk, text must be single-spaced only. Please identify your editing program.

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

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

SUBJECTS

We prefer material on these topics:

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

WRITING YOUR ARTICLE

Observe the following points of style:

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

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

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

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

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

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

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

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

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

AUTHOR'S RIGHTS

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

PAYMENTS

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

Please mail your submissions to:

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