Hardcore COMPUTIST's

Book Of Softkeys

Volume I

SoftKey Publishing
Welcome to the Book Of Softkeys, a publication devoted to the serious user of Apple ][ and Apple ][ compatible computers. The articles published in this volume detail the removal of copy protection schemes from commercial disks or contain information on copy protection and backup methods in general.

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.
General Information:

Applesoft Disks (Softkey To) .............................................. 9
Some general pointers on how to find Applesoft programs loaded by non-standard DOS disks.

Boot Code Tracing ......................................................... 22
There’s no protecting against this method of disk cracking.

Boot Code Tracing Revisited ........................................... 27
Take another look at how to trace a program as it loads into memory.

Demuffins ................................................................. 41
Make this handy disk cracking tool from a program supplied by Apple.

Diskedit ................................................................. 43
Read, Write and Edit sectors on any DOS 3.3 disk.
It’s easy with this program.

Diskview ................................................................. 84
A mini-nibbler that reads raw nibbilized data from any disk, regardless of format.

Getting On The Right Track ........................................... 99
Here’s how to tell what track your read/write head is positioned over. A real help on those difficult copies.

Hidden Locations Revealed ........................................... 106
Take a peek at these favorite locations used by protected software.

(A Fix For) RANA Drive Owners ..................................... 130
A way to boot code trace on a Rana drive.

Tricks & Bombs ......................................................... 139
Contents

Softkeys
Akalabeth .............................................. 1
Ampermagic ............................................. 4
Apple Galaxian (see Boot Code Tracing)
Aztec .................................................. 12
Bag Of Tricks ........................................... 13
Bill Budge’s Trilogy ................................... 19
Buzzard Bait ............................................. 35
Cannonball Blitz ....................................... 37
Casino .................................................. 38
Data Reporter .......................................... 39
Deadline (see Zork I)
Disk Organizer II (see Hidden Locations Revealed)
Egbert II Communications Disk ....................... 89
Hard Hat Mack ........................................... 101
Home Accountant ....................................... 110
Homeword ............................................... 112
Lancaster ............................................... 113
Magic Window II ......................................... 115
Multi-Disk Catalog (see Boot Code Tracing Revisited)
Multiplan ............................................... 119
Pest Patrol ............................................... 120
Prisoner II ............................................... 129
Sammy Lightfoot ....................................... 131
Screenwriter II ......................................... 133
Sneakers .................................................. 135
Spy’s Demise ............................................ 136
Starcross ............................................... 137
Suspended (see Zork I)
Ultima II ................................................. 141
Ultima II (2) ............................................. 147
Visifile ................................................... 149
Visiplot/Visitrend ....................................... 151
Witness (see Zork I)
Wizardry .................................................. 152
Wizardry (2) .............................................. 155
Zork I, Zork II, Zork III ................................. 156
Akalabeth
California Pacific Computers

Requirements:
Apple ][ 48K
One blank disk
MUFFIN (from DOS 3.3 master disk)

By Bobby

Akalabeth is a hi-res adventure/maze/treasure hunt game distributed by California Pacific Computers. It has a few bugs that need correcting. Here is a method of "down-loading" Akalabeth from its protected diskette so that you can make the FIXes described in Softfix and additions.

1) Boot from the DOS 3.3 master disk

PR#6

2) Remove the master disk, insert the blank disk and enter the hello program

10 PRINT CHR$(4) "RUN*I******************************
********I" :REM THERE ARE 26 ASTERISK

3) Initialize the blank disk with the new hello program

INIT HELLO

4) Remove the initialized disk, insert the master disk and load MUFFIN

BLOAD MUFFIN

5) Make the following changes to allow MUFFIN to read the protected disk

POKE 4257,6
POKE 6664,222
POKE 6685,181
POKE 6742,217
POKE 6774,222
POKE 6795,173
POKE 6834,217

6) Now start MUFFIN and follow the prompts to copy Akalabeth. Use the initialized disk for the destination disk.

CALL 2051

6) To play the game, simply run the hello program.
One of the reasons for removing programs and files from copy-protected disks is that you can now fix them, alter them, and even make additions to them. In other words, you can now customize them to fit your particular needs.

Since you have already "freed" Akalabeth from its "protected" disk, you can now fix its annoying scrolling problem and add a new magical command.

**PROBLEM:** The "hit points" of the foes you are fighting often flash (scroll) by so fast that you can't see what they were and therefore cannot make a valid combat decision.

**FIX:** Put in a pause after it prints the hit points. There are other minor fixes that modify screen format, greetings and farewells (Quit has been changed to eXit).

**SPECIAL MAGICAL ADDITION: TELEPORTATION**
To help the brave Magi descend into, and ascend out of, the forbidden depths of this World of Doom, we have added a special teleportation option. (Use only if you have 1 or more magic amulets because this option "burns up" amulets just as if you had used the Magic Ladder-Up or Ladder-Down commands.)

When you choose the 5th option: TELEPORT, you'll get: 1-UP 2-DOWN. Take your choice. If you choose UP you will continue upward until either you surface or you run out of amulets and get an: OUT OF MAGIC message. Choose DOWN and you'll be asked: HOW MANY LEVELS? It is a GET statement, so choose a number from 1 to 9 and don't bother to hit "return". If you run out of magic now, you're really in a World of Doom!

Here's how to add the teleport command to your deprotected akalabeth disk.

1) Load the file "MAIN PROGRAM"

   LOAD MAIN PROGRAM

2) Enter the BASIC program on the next page.
3) Save the changed program back to the disk

   SAVE MAIN PROGRAM

That's all there is to it. If you know of another fix or addition to Akalabeth or for any prepackaged programs on the market, (whether they are on protected disks or normal format disks), drop a note to SoftKey Publishing and we'll let other apple-users know about it.
BASIC program to add teleport option

1664 PRINT M$(MN) "'S' HIT'POINTS=" MZ%(MN , 1) : FOR QS = 1 TO 500 : NEXT
1682 PRINT "1-UP"" 2-DN"" 3-KILL"" 4-??"" 5-TELEPORT" ; GET Q$ :Q = VAL (Q$) :
   PRINT Q : IF Q < 1 OR Q > 5 THEN 1682
1685 ON Q GOTO 1686 ,1690 ,1691 ,1692 ,1800
1800 VTAB 21 : CALL - 958 : PRINT "1-UP"" 2-DOWN"" ; : GET Q$ :QQ = VAL (Q$) :
   PRINT QQ : IF QQ < 1 OR QQ > 2 THEN 1800
1810 ON QQ GOTO 1820 ,1840
1820 IN = IN - 1 : PRINT CHR$ (7 ) : IF IN = 1 THEN 1581
1822 IF PW(5 ) < 1 THEN PRINT "OUT OF""MAGIC" ; GOTO 1090
1824 IF RND (1 ) > .75 THEN PW(5 ) = PW(5 ) - 1
1826 GOTO 1820
1840 PRINT "HOW MANY LEVELS?" ; : GET Q$ :QQ = VAL (Q$) : PRINT QQ : FOR QS
   = 1 TO QQ :IN = IN + 1 : IF RND (1 ) > .8 THEN PW(5 ) = PW(5 ) - 1
1844 IF PW(5 ) < 1 THEN QS = QQ : PRINT "OUT OF""MAGIC"
1846 NEXT : GOTO 1090
60020 DATA "HIT'POINTS..", "STRENGTH....", "DEXTERITY...", "STAMINA....", "WISDOM......", "GOLD......."
60081 VTAB 11 : HTAB 18 : PRINT "X-EXIT"
60210 PRINT "WHICH ITEM SHALT THOU BUY" ; : GET Q$ : IF Q$ = "X" THEN PRINT 
   : PRINT "GOOD LUCK" ; FOR Z = 1 TO 1000 : NEXT : TEXT : HOME : RETURN
60237 VTAB 10 : HTAB 13 : PRINT C(5 ) "***"
Amper-Magic

Anthro-Digital

Requirements:
48K Apple or an Apple //e
Amper-Magic
One blank disk
Apple's COPYA program
Text file editor (AppleWriter II, Apple's EDASM, etc.)
A Disk Edit utility (THE INSPECTOR or WATSON.)
FIXCAT from "Bag of Tricks" and FID are also useful.

By Bob Bragner

Amper-Magic is a disk library of machine-language routines that can be easily attached to Applesoft programs thus providing extensions to BASIC such as PRINT USING, SWAP, DELETE ARRAY, INPUT ANYTHING and many, many more. The routines are connected to Applesoft through the ampersand "&" and it is easy to pass parameters. There is also a second disk of routines available dealing largely with screen formatting.

If you do any serious programming in Applesoft and find the language too slow for some applications but the thought of writing your own machine language routines to speed things up makes your head hurt, then Amper-Magic is for you.

This program is super-friendly and comes with a well thought-out loose-leaf manual. The price is a little steep, however. I paid $67 for my master and another $15 to be a "Registered Commercial Owner." Such registration entitles you to automatic notification in case the manufacturers add any corrections or improvements and also authorizes you to use the Amper-Magic routines in any commercial applications you may write. Having paid this one-time fee, your only other obligation is to mention Anthro-Digital's name in the documentation of any program using their routines. Fair enough. More than fair: generous.

However, I did balk at the $7 price tag on the backup disk. That's about $4 more than I'm willing to pay for such a necessity. But I like lots of backups, so I set about making them.

To my dismay I discovered that not only is Amper-Magic protected, it is protected in a very ingenious, and potentially dangerous, way. Actually, it is mentioned nowhere that the disk is protected and you are nowhere warned not to write anything onto the master disk. This is not friendly.

Amper-Magic has several levels of protection. You can copy the disk with Apple's COPYA without a hitch, but the result won't work. When you attempt to EXEC the main control textfile as directed, the disk drive turns on and stays on.
Hitting reset will lock up your machine. The only ways out are control/open Apple/reset or turn-the-machine-off-then-on.

An examination of the controlling text file called Amper-Magic showed, among other things, the following:

```
FF$ = "A" : FOR I = 1 TO 29 : FF$ = FF$ + CHR$(8) : NEXT I
PRINT CHR$(4); "BRUN";FF$
DEL 1,0
RUN
```

For you newcomers, CHR$(8) is the backspace on your keyboard (⌘H). It’s been so long since anyone used control characters to conceal filenames in catalogs that it never even occurred to me to look for them. This particular trick causes the filename to be written over by the next one in the catalog, thus rendering it invisible. Cute.

Changing the file’s name to something more respectable (on the copy, of course), I proceeded to examine the binary file now simply called “A”.

This is a 288-byte file that lives at $25B. The code is not difficult to disassemble, but it was doing some strange things. It appeared to look for 14 bytes in track 0, sector 0, and to compare them with a table in memory. If they matched, and if a number of other conditions were met, then another program called “AMPER.MAGIC PROGRAM” was loaded. Otherwise, program “A” hung up. Since program “A” did a jump to the standard DOS LOAD command, AMPER.MAGIC PROGRAM had to be somewhere in the catalog track (track $11) and it had to be in the VTOC. But it wasn’t.

Call up WATSON or THE INSPECTOR and do a sector-use map of your Amper-Magic disk. Now look at sector $F of track $11 and backstep through the catalog. In sector $C you will see the “A” file whose name contains 29 inverse “H”s (it’s at track $21, sector $08.

In sectors $B and $A you will see some deleted Applesoft files with odd names like NIL0 and NIL1 which seem to have been used to overwrite other deleted binary files with names like “REPEET” and “RWTS.O.” They contain (meaningless?) data statements. When you get to sector $9, you will see that the link to the next catalog sector (8) is missing! Hmm... To DOS this means that this is the end of the catalog.

Backstep once more and all at once the words “AMPER.MAGIC PROGRAM” are staring at you from sector $8. This program appears to start at track $0F, sector $F. Check it out; it does.

Backstep some more. Notice that the link in sector $2 is missing. Now look at sector 1 and you will find yourself looking at what must be a VTOC! “But VTOC’s are supposed to be in
sector 0," I hear someone say. I don't? Well, anyhow, there is a byte in DOS called READ/WRITE VTOC BUFFER. It is located at $B00D (45069) and it normally contains a 0.

Just for fun, try poking the value 1 there and type "CATALOG" with an ordinary disk in the drive. (For a lot of fun, poke in a number greater than 15!) Unless the disk has had a great many file names on it at one time, this track will usually be empty and your catalog will have disappeared. If any file names have been stored in this sector, CATALOG will display some quaint garbage since DOS will attempt to interpret what it finds there as a VTOC. Place your AMPER.MAGIC master in the drive, do a POKE 45069,1, then CATALOG and you will see the hidden file.

Returning to a more careful examination of the disassembled code of "A", it turns out that this program does indeed store a 1 at $B00D before it attempts to load the AMPER.MAGIC PROGRAM. The latter program, for its part, pokes a 0 back into that location so it can get at the routines stored on the disk. The reason this method of copy-protection is dangerous, aside from the fact that you don't know it is there, is that the normal VTOC at sector 0 may not know anything of AMPER.MAGIC PROGRAM. If it thinks the sectors it occupies are unused, and if you try to save something on the disk, DOS will cheerfully write all over the hidden program. Moral: Never write anything to a master disk, even if the manufacturer doesn't tell you not to.

Additional examination of "A" reveals that the whole routine can be bypassed. The program checks to see whether or not it is running on an original disk and whether or not an EXEC file is in operation. Finally, it doctors the VTOC and loads the hidden program. A quick examination of AMPER.MAGIC PROGRAM shows that it makes no attempt to protect itself once it is running.

Here is a step-by-step procedure to crack AMPER-MAGIC (the commands to type in are given in bold):

1) Copy the master disk using normal COPY A procedures

   RUN COPYA

2) Point DOS to the VTOC on sector 1

   POKE 45069,1

3) Catalog the disk

   CATALOG

4) Load the AMPER.MAGIC PROGRAM file

   LOAD AMPER.MAGIC PROGRAM
5) Unlock the AMPER.MAGIC PROGRAM file
   UNLOCK AMPER.MAGIC PROGRAM

6) Delete the AMPER.MAGIC PROGRAM file
   DELETE AMPER.MAGIC PROGRAM

7) Point DOS to the VTOC on sector 0
   POKE 45069,0

8) Catalog the disk again
   CATALOG

9) Save the AMPER.MAGIC PROGRAM file
   SAVE AMPER.MAGIC PROGRAM

10) Unlock the EXEC text file AMPER.MAGIC
    UNLOCK AMPER-MAGIC

11) Bring up a text file editor such as AppleWriter IIe, Apple EDASM, etc.

12) Load AMPER-MAGIC into the editor.

13) Insert the word "REM" at the beginning of the line that reads PRINTCHR$(4);"BRUN";FF$.

14) Insert the following:
   CHR$(4)"LOAD AMPER.MAGIC PROGRAM"
   before the last line in the file (the one that says "RUN").

15) Save this as a text file under the name AMPER-MAGIC
    SAVE AMPER-MAGIC

16) Exit the editor.

You now have a de-protected copy of Amper-Magic from which you can make all the working backups you want using normal copy procedures.

You're not quite done yet, though, because the disk's catalog is still a bit messed up. To manually restore the missing links you can use WATSON/THE INSPECTOR (and zero out the extra VTOC at track $11, sector 1) or you can run the FIXCAT utility in "Bag of Tricks" and let it do the work for you.

You really should repair the catalog if you intend to use FID to move files off of, or onto, the deprotected disk. If you use FIXCAT you should ignore the temptation to restore the deleted files since most of them have been written over.

There is, however, a strange little Applesoft program not located at track $10, sector $E (its track/sector list is in sector $F). Restore this program and call it "WEIRD HELLO", then run it for an odd message.
By the way, Volume 2 of Amper-Magic is unprotected and can be copied without any fooling around. I like to FID all its binary files over to the deprotected master since both disks are mostly empty space anyhow.

Do you want to add more routines to the library? Move everything to a hard disk or a Ram Pseudisk? Eliminate the annoying beeps in the main program? With your unprotected version of Amper-Magic you are now free to make any modifications you wish.
Softkey To Applesoft Disks

By Bobby

If the program you have is working to your satisfaction, there is really no reason to go through the trouble of “unlocking” the copy-protection. I recommend that everyone obtain a bit-copy program and use it to backup their software. The methods described here are for programs that you feel need FIXing. In order to FIX them, you have to be able to list them.

Another reason for putting programs on normal DOS is to conserve disk space by placing more than one program on a disk. If you just make a bit-for-bit backup of ten disks, you now have 20 disks. However, by “downloading” the protected program to normal DOS, you can probably put all ten on two or three one-sided disks.

The Open-Heart Surgery Method

Requirements:
Apple ][+ 48K
Applesoft in ROM
Integer card
Tape recorder
One blank initialized disk (3.2 or 3.3 as appropriate)

This method is not for beginners. It requires some knowledge of programming and involves a number of monitor commands. An understanding of Zero Page locations is helpful.

No matter what has been altered, this method will work because the computer must know the program location in order to RUN it. The following locations apply to Applesoft in ROM and 48K DOS. They are used by the computer to tell it where the program is located and what to do with it. Multibyte addresses are given in standard format (e.g. hex lo-byte, hi-byte).

$D6 is the RUN flag. Any time this byte has the high bit set, the program in memory will auto-run.
$3F2, $3F3 is the reset vector. (See page 37 of your Apple ][+ Reference Manual (A] RM.)
$3F4 is the power up byte (EOR of $A5 and the value at $3F3. See page 37, A] RM.)
$67, $68 is the start-of-program pointer. See page 140 of the Applesoft BASIC Programming Manual(AB-PRM).
$AF, $B0 is the end-of-program pointer. See page 141, AB-PRM.
$9DBF (CALL-25153) reconnects DOS. See page 144 of your DOS 3.3 Manual (DOSM).

$A851 (CALL-22446) reconnects the input hooks. (Bypasses $9DBF and any time-bombs that may have been planted there.)

$AA60, $AA61 is the length of the last loaded program. See page 144, DOSM.

$AA72, $AA73 is the start of the last BLOADED program. See page 144, DOSM.

The following steps were written for an Apple ][+ 48K with Applesoft in ROM and an Integer card. If you have Integer in ROM and an Applesoft card, be sure to reverse the appropriate steps (i.e. when it says to flip the switch UP on your Integer card, flip the switch DOWN on your Applesoft card). Hitting RETURN after commands is implied and will only be referenced in certain lines for clarity.

1) Boot the backup copy of your program disk (never use the original).
2) Flip the Integer card switch UP and press RESET to enter the monitor.
3) Check the start of program pointer. If the number is not $801, then write it down for later reference.

   67.68

4) Reset the run flag.
   D6:0

5) Change the reset vector to jump into the current language.
   3F2:03 E0 45

6) Flip the Integer card switch DOWN, press RESET and type LIST

If the program does not list, you may be trying to list a binary file. Check $AA72, AA73 to see if this is true.
7) Save your program to tape.
8) Take a good look at your listing. Write down the names of any files that are LOADed or RUN and the start address of any binary file, if given. List all POKEs and CALLs.
9) On some protected disks the command parser in DOS is changed. This tends to wipe out some of the common commands (e.g. LOAD, SAVE, CATALOG). Another popular trick is to change DOS so it does not allow any direct commands. In order to load and save the files that make up your program, select the first line entry that LOADs or BLOADs a file. DELETE all line entries above and below the selected line (e.g. you want line 5,
so DELETE lines 0 through 4, and 6 through 63999).

10) Now

    CALL -25153

If it bombs, restart. Repeat all steps up to 10 and now

    CALL -22446

This reconnects the DOS. Then type

    RUN

This will load your selected file.

11) Save the file to tape and repeat these steps until all files are SAVEed.

12) Boot normal DOS. LOAD the files one at a time from tape and SAVE to your disk.

13) Examine all the Applesoft listings to make certain that all the files have been transferred.

Now that all the files are on normal DOS, examine them for hidden bombs. Bombs are program lines that serve no purpose other than to garbage the program. See "Tricks and Bombs".

For Integer Programs

The zero page pointers for Integer programs are:

<table>
<thead>
<tr>
<th></th>
<th>HEX</th>
<th>DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start of program</td>
<td>$CA, $CB</td>
<td>202, 203</td>
</tr>
<tr>
<td>End of program</td>
<td>$4C, $4D</td>
<td>76, 77</td>
</tr>
</tbody>
</table>

These pointers tell your computer where the program is in memory. The hex address is in normal lo-byte, hi-byte format.

Hope these tips will help you find your program. Good luck!
Aztec
Datamost

Requirements:
Apple II 48K
One blank disk
Copy A from DOS master disk
Disk edit program

By Marco Hunter

Here is a quick softkey.

1) Use the modified COPY A from Zork softkey.
2) Edit track 00, sector 03, change byte 42 from 38 to 18.

That's it.
Bag Of Tricks
Quality Software

Requirements:
Apple ][+, //e or compatible
Blank disk initialized with 48K slave DOS
Bag of Tricks disk

By Neil Taylor, Earl Taylor and Ray Darrah

Have you ever booted the Bag of Tricks disk and received an irritating message to use the original, when it's already in the drive? Have you ever wanted to avoid the menu and skip right to the needed program? Perhaps you are afraid of crashing the original and can't get a good copy. Here is an easy (albeit somewhat long) way to get an unprotected version.

Basic Procedure

To unprotect the programs on the original disk, each one will have to be loaded by its DOS and then saved by a normal DOS. The programs loaded by the Bag of Tricks DOS are located at $800 in memory which normally is overwritten during the boot process. Before they can be saved, they must be moved to a safe area of memory. The programs can then be run by normal DOS.

Loading and Saving

The following procedure for loading and saving TRAX is used in a slightly different form for each of the remaining Bag of Tricks programs: INIT, ZAP, and FIXCAT.

TRAX

1) Boot the 3.3 master, insert the blank disk and type

   FP
   INIT HELLO

2) Boot the Bag of Tricks disk (the menu will be displayed).
3) After the light goes off open the drive door.
4) Press reset once, wait a couple of seconds and press it again.
5) To enter the monitor

   CALL -151

6) Then type

   9489:4C 59 FF
7) Close the drive door.
8) Type
   \textbf{9400G}
9) To load TRAX, type
   \textbf{T}
10) Then type
    \textbf{3800<800.2AFFM}
11) For Trax only, type
    \textbf{6700<8700.93FFM}
12) Place the blank disk into the drive and boot it with
    \textbf{C600G}
13) Now
    \textbf{BSAVE TRAX,A$3800,L$2300}
14) And for TRAX and INIT only
    \textbf{BSAVE TRAX.SUP,A$6700,L$D00}

The same format can be used for INIT, ZAP, and FIXCAT with changes in Steps 9, 10, 13, and 14. Step 11 is not necessary. Complete Steps 2 through 8 for each of the remaining programs, then follow the special steps listed under the program title.

Complete INIT first:

\textbf{INIT}

9) To load INIT type
   \textbf{I}
10) Then type
    \textbf{3800<800.325EM}
11) And
    \textbf{BSAVE INIT,A$3800,L$2B00}
12) Place the blank disk into the drive and boot it with
    \textbf{C600G}
14) Then
    \textbf{BSAVE SUPPLEMENT,A$7600,L$A00}
ZAP

The supplement is the same for INIT, ZAP and FIXCAT, so Step 14 can be eliminated.

9) To load ZAP type
   Z
10) Then type
   5000<800.4CFFM
12) Place the blank disk in the drive and boot it with
   C600G
13) Then
   BSAVE ZAP,A$5000,L$4500

FIXCAT

9) To load FIXCAT type
   F
10) Then type
   4800<800.1FFFM
12) Place the blank disk in the drive and boot it with
   C600G
13) And
   BSAVE FIXCAT,A$4800,L$1C00

To get the picture, complete Steps 2 through 4. Then boot the backup. When the Applesoft cursor is displayed ( [] ), type

   BSAVE PICTURE,A$2000,L$2000

FIXPRG3

The last step is a fix for the SUPPLEMENT program.

15) Once again boot the original Bag of Tricks disk.
16) When the menu appears, open the drive door, press reset, wait a second and press reset again.
17) Enter the monitor and move the patch to a safer location.

   CALL - 151
   5400<9400.96FFM
18) Change the program so that it will work at this new location

\[ 5485:55 \]
\[ 5517:55 \]
\[ 5530:55 \]
\[ 5542:56 \]
\[ 554A:56 \]

19) Insert the blank disk and boot it

\[ C600G \]

20) Save this patch program

\[ BSAVE FIXPRG3,A$5400,L$300 \]

21) Change the HELLO program to the following and you're done!

**Hello Program**

```
10 TEXT: HOME: HIMEM: 25600
20 IF PEEK (104) = 96 THEN 50
30 POKE 103,1: POKE 104,96: POKE 24576,0
40 PRINT CHR$(4) "RUN\"HELLO"
50 A = PEEK (-16384) - 128: IF A = 70 OR A = 73 OR A = 84 OR A = 90 OR A = 69 THEN 100
60 POKE -16368,0
70 HTAB 10: VTAB 10: PRINT "LOADING\"MENU..."
80 PRINT CHR$(4) "LOAD\"PICTURE,A$4000" : POKE -16299,0 : POKE -16297,0 : POKE -16304,0
90 IF PEEK (-16384) < 128 THEN 90
100 A = 0: A$ = CHR$(PEEK (-16384) - 128) : POKE -16368,0
110 IF A$ = "T" THEN A$ = "TRAX" : A = 1 : GOTO 170
120 IF A$ = "Z" THEN A$ = "ZAP" : GOTO 170
130 IF A$ = "I" THEN A$ = "INIT" : GOTO 170
140 IF A$ = "F" THEN A$ = "FIXCAT" : GOTO 170
150 IF A$ = "E" OR A$ = CHR$ (27) THEN TEXT: HOME: END
160 GOTO 90
170 TEXT: HOME: VTAB 11: HTAB (13 + LEN (A$)) / 2: PRINT "LOADING\" A$
180 D$ = CHR$ (4)
190 IF A = 1 THEN PRINT D$ "BLOAD\"TRAX.SUP,A$8700" : GOTO 210
200 PRINT D$ "BLOAD\"SUPPLEMENT,A$7600"
210 PRINT D$ "BLOAD\" A",A$800"
220 PRINT D$ "BLOAD\"FIXPRG3,A$5400"
230 POKE 47016, Peek (43) : POKE 38079,A : CALL 21635
```

**Getting Into The Program**

An alternate method would be to boot code trace the DOS.
The boot process of Bag of Tricks is relatively simple but tedious, especially since it would have to be done five times (once for each program and once for the picture).

That problem can be bypassed by taking advantage of an oversight by the authors. When reset is pressed the Apple tries to boot because the power-up byte is not set correctly. This is the byte that tells the Apple when it has been turned on. (See page 37 of the Apple ][+ Reference Manual). When the power-up byte is set improperly the Apple will try to boot regardless of the address pointed to by the reset vector. When reset is hit from the menu the Apple acts like it has just been turned on and tries to boot. When reset is pressed the second time the Apple is put into Applesoft.

### Loading The Programs

In the sixth step of the save/load procedure the three bytes 4C 59 FF represent the machine language opcodes which tell the computer to jump to the routine that causes it to stop and enter the monitor (acting like a stop from Applesoft). Now after the DOS has loaded any of the programs, control will be given to the user, not to the program.

### Saving To Normal Disk

The program is now in memory and the Apple is under control with the modified Bag of Tricks DOS in the machine. Unfortunately, it is far from normal and has no convenient SAVE or BSAVE. What now? Save it to tape? Perish the thought; a normal DOS can be rebooted.

Since the booting process uses page 8 ($800-$8FF) in memory, which is exactly where the program starts, a special routine in the Apple’s monitor is used for moving memory out of the way. It simply transfers the part of memory which the program resides in byte by byte from one place in memory to another. By moving the programs higher in memory they are put in a safe area not used by the boot. That is what Steps 9 and 10 are for. Once the program is moved, the backup disk can be safely booted.

### Backup Files

There should be nine files on the backup now: TRAX, TRAX.SUP, INIT, SUPPLEMENT, ZAP, FIXCAT, PICTURE, FIXPRG3 and HELLO. The HELLO program is simply a menu that allows the backup to imitate the original disk. The picture is the same as the one on the original disk.
The other seven files make up the four major Bag of Tricks programs (the other files are routines). Each program is in two parts, a main section and a supplement, but the supplements for INIT, ZAP and FIXCAT are the same. To run any of these programs, the accompanying supplement must also be loaded. To use TRAX, TRAX.SUP must be loaded first. For the other three programs, SUPPLEMENT must be loaded first.

How To Run The Programs

Because the programs were moved before they were saved, they will be loaded into the wrong spot if just BRUN or BLOADed. To make sure everything is in the right place, DOS has to be told where to place the program. FOR EXAMPLE: To run TRAX, first load in the supplement with BLOAD TRAX.SUP,A$8700. This loads the supplement into the correct place in memory. Then the TRAX program can be run with BRUN TRAX,A$800.

Similarly, the supplement for INIT, ZAP and FIXCAT would be BLOAD SUPPLEMENT,A$7600. To run the program: BRUN INIT,A$800 (ZAP or FIXCAT can be substituted for the title INIT).

Final Analysis

All four Bag of Tricks programs are extremely useful. ZAP is an excellent disk editor with convenient help pages. It also has definable commands (a nice touch). INIT is the program that you needed to convert all your disks to DOS 3.3. It allows reinitialization without loss of data. FIXCAT is great for doing all of those tedious chores related to recovering crashed disks.

There is only doubt about TRAX. Its sole use seems to be looking at the protection schemes on disks (it gives a great output for users of IOB). However, TRAX will not analyze the Bag of Tricks disk. If the authors couldn’t figure out how to analyze their own protection schemes, TRAX can’t be all that good. On the other hand, maybe it was deliberate. Maybe the authors are trying to say, “Break and copy other disks, but not ours!”
Bill Budge's Trilogy Of Games

California Pacific

Requirements:
48K Apple II Plus
BOOT13 (from DOS master disk)
Trilogy of Games
A blank diskette

By Michael Decker

One of Bill Budge's earliest offerings was his Trilogy of Games: Driver, a rudimentary driving-skills game; Pinball, a rudimentary you-guessed-it; and Space Wars, a (I won't say it) two-player space battle.

This old, DOS 3.2 disk still sells, and Space Wars remains one of the most entertaining arcade-style games in which two players can directly battle each other. Most people often prefer it to newer, much more sophisticated games in which one battles the computer. I was motivated to de-protect the game by a slight bug (one player's ship is more affected by gravity than the other ship) and by my annoyance with the game's DOS 3.2 format.

Inside Budge

I first tried cracking the disk. Protection was simple: changes in the prologues/epilogues. However, it looked like direct disk addressing was used. Ugh.

So, I booted with an Inspector/BASICS disk, then booted Trilogy. At the menu I reset out, then looked memory over: picture at $4000; program at $6000. Hmm.

I then booted DOS and saved picture and program. Next, I restarted the program (6000G) and, in turn, loaded each game. I found the entry points and saved the games. Next, I examined the main program and identified the transition between the menu and the disk access routines. Finally, I wrote an Applesoft program to handle the game switching. Presto! The Transparent Budge!

Doing It

1) Boot the DOS 3.3 System Master.

   PR#6

2) Prepare to boot 13-sector DOS

   BRUN BOOT13
3) Insert the Trilogy disk and press RETURN.
4) Hit ESC to get to menu.
5) Remove the disk and press RESET.
6) Press RESET again.
7) Boot the 48K slave disk.

PR#6

8) Save the picture first

BSAVE PICTURE, A$4000,L$2000

9) Enter the monitor

CALL -151

10) Make this modification

67B7:4C D2 D7

11) Save the first 8 pages of this modified program

BSAVE ATTRACT, A$6000,L$800

12) Restore the original program

67B7:20 00 6D

13) Insert Trilogy disk and re-start the game.

6000G

14) Hit ESC to get to menu.
15) Type 1 to select Driver, the first game in Trilogy.
16) When the title and/or instructions come up, hit RESET.
17) Swap disks and save the just loaded program

BSAVE DRIVER, A$800, L$1801

18) Enter the monitor again

CALL -151

19) Repeat Steps 13-18 for PINBALL and SPACE WARS using the same BSAVE parameters.
20) Coldstart BASIC and DOS

FP

21) Type in the following BASIC menu "HELLO" program.

10 ON PEEK (104) = 112 GOTO 20 : POKE 103, 1 : POKE 104, 112 : POKE 28672, 0
   : PRINT CHR$ (4) "RUNBUDGE.HELLO"
20 PRINT CHR$ (4) "BLOAD"ATTRACT"
30 PRINT CHR$ (4) "BLOAD"PICTURE"
40 POKE 10, 76 : POKE 11, 0 : POKE 12, 96 : PRINT USR (0) : CALL - 10621
50 D$ = CHR$ (13) + CHR$ (4) : VTAB 1 : ON PEEK (67) - 3 GOTO 60, 70, 80
60 PRINT D$ "BLOAD"DRIVER" : CALL 3523 : RUN 40
22) Save it

SAVE BUDGE.HELLO

I would sometimes get an ?OUT OF MEMORY ERROR. If you should encounter this, just type RUN 20 and you’ll be fine.

Now, would someone tell me how to make both spaceships feel the same gravity?
Boot Code Tracing

By Bobby

Requirements:
Apple II
One disk drive with Apple controller
Apple Galaxian
Empty 48K slave disk
Knowledge of machine code/assembly language

Boot code tracing is the most difficult of all the softkeys presented. It is also the most effective method of transferring single load binary programs, those which load into the Apple at Boot and do not access the disk again. Therefore, be warned that this softkey is definitely not for beginners. You must have some knowledge of machine code or assembly language.

The initial Boot or Boot 0 is determined by a program on the disk controller card. This program is stored in ROM (Read Only Memory) and cannot be changed by the program on a disk. A non-standard format may be used on a disk to prevent copying, but that disk will not Boot on your Apple unless track 0, sector 0 is readable by the disk controller card. This means that you can also read it and, by controlling the boot process, you can determine where the program is and save it to disk.

In order to demonstrate this process, the Boot of the binary program Apple Galaxian will be traced. This is a familiar game and almost everyone owns a copy.

The Boot 0 code is at $C600. If your controller card is not in slot 6, then change the number after the $OC to correspond to the appropriate slot (i.e. Slot 5 would be $C500). First the code will be moved down into RAM (Random Access Memory) where a portion of it can be modified. Turn on your Apple, press RESET to halt the boot, then enter the monitor

CALL -151

And type

9600<C600.C6FFM

This will move the Boot code to page $96 ($9600), where the changes will be made.

NOTE: Commands will be on a separate line and printed exactly as you should enter them. Press return after each line entry. If a command has already been listed, it will be referred to but not listed again.

Be sure you move the code to a page boundary that corresponds to the slot that your controller card is in (i.e. for slot 6 - $9600, $8600 etc. or for slot 5 - $9500, $8500 etc.).
The reason for this is that the Boot 0 code contains a routine that finds which slot the controller card is in. It does this by calling a return code in the F8 ROM and extracting the return address to locate the page boundary. The code itself is relocatable (will run anywhere in memory).

The purpose of Boot 0 is to transfer the code stored on disk at track 0, sector 0 into memory and to execute it. Then the disk code (Boot 1) will take over the Boot process. Here is where the first change will be made.

Examine the code at the end of page $96. At $96F8 you will find a JMP to $801. This is the next Boot stage (Boot 1). Boot 0 needs to load and not run the Boot 1 code at page $08, so the code at $96F8 will be changed to JMP to the monitor after turning off the drive motor. In order to save having to reenter the same code, put the routine at $9500.

96FA:95 N 9501:AD E8 C0 4C 59 FF

The base address $C088 is used to turn the drive motor off. You must add the slot number of your controller card to this address in the form $s0 where "s" is the slot (i.e. slot 5 would be $C088 + $50 = $C0D8). If you’re not used to hexadecimal numbers, there is a table on page 82 of the Apple ][+ reference manual with the base addresses and conversions for each slot.

After making the changes and checking that they are correct, you are ready to begin. Run the code at $9600 (Boot 0).

$9600G

When the drive stops, Boot 1 will be in memory at page $08. If you examine the code at $800 you will see that it is written to run at page $02. The first part of the code is a move routine that transfers the code from page $08 to page $02, then JMPs to page $02.

Boot 1 should be moved to page $98 so you can make changes. The code must be moved or else the next time you run Boot 0, it will be overwritten.

9800<800.8FFM

Now the modified Boot 0 can be linked at $9600 to the Boot 1 code at $9800. The move routine should then be changed so it will work at its new address, and the JMP to the next Boot stage should be changed to the routine at $9500.

96FA:98 N 9805:98 N 9843:95

The "N" is used as a null command to separate changes. It is the monitor command to set normal mode and does nothing, since you are already in normal mode.

After you have made the changes and checked that they are correct, run the code at $9600 again. When the drive stops, Boot
2 will be in memory at page $03. Move the code at $300 to $9300.

9300<300.3FFM

The exit from Boot 2 is via an indirect JMP at $9343. This JMP normally points into itself. Rather than write any additional code to check when this JMP is changed, allow the code at $300 to be called as a subroutine and change the indirect JMP at $9343 to point to $9501. This works because the JMP at $9343 is not seen until Boot 3 is completely loaded.

Link Boot 1 to Boot 2 and run Boot 0 again.

9343:4C 01 95 N 9843:93 N 9600G

Boot 3 is now in memory. Location $93CC holds the page minus 1 of the start of Boot 3. You should find a $B6 there. That means that Boot 3 starts at page $B7 or $B700 in memory. This is the final Boot. This stage will load the main program.

Boot 3 is large and would be cumbersome to modify if moved. It can remain right where it is and the Boot 2 code can be changed so that next time it will load in a different location. The page number that Boot 2 uses when it loads Boot 3 from page $B6 to page $D6 will be changed.

9313:A9 D6 EA

This will cause Boot 2 to try to load the Boot 3 code into the space occupied by the monitor. Since the monitor is in ROM (Read Only Memory), nothing will change and the Boot 3 code at $B700 is protected.

At $B749 is a routine that garbages the Boot 3 code after it loads in the program. At $B759 is a JMP to $600. This is the actual start of the program. Since $600 is in the text page, the boot with the code can't just be stopped at $9501. If this were done, the code on the text page would scroll or be over-written. The code needs to be moved up to some safe place in memory. Also, the end of the program is at $A000 (my guess), which is the same place that DOS resides on a normal disk.

The Galaxian Logo is on HIRES page 1 ($2000-3FFF). It is reasonable to assume that this is a safe place and that no code exists there. The logo will be lost, but the memory is needed.

The following code is a routine that compresses memory by moving page $00 thru page $08 up to page $20, and moving page $40 thru page $9F down to page $29.

B000:A2 00 BD 00 40 9D 00 29
B008:E8 D0 F7 EE 04 B0 EE 07
B010:B0 AD 04 B0 C9 A0 90 EA
B018:A2 00 BD 00 00 9D 00 20
B020:E8 D0 F7 EE 1C B0 EE 1F

Book of Softkeys Vol. I 24
After entering the code, link Boot 3 to it and link Boot 2 to Boot 3.

Check to see that all the changes are correct, then run Boot 0. When the drive stops you should see a lot of inverse and flashing characters on the screen. This is the portion of the program that is loaded into the text screen area. An image of this code is safely stored at $2000.

Remove the backup copy of Apple Galaxian and insert the 48K slave disk in the drive. Boot the disk and enter the monitor. There are a few more changes to make before the file can be BSAVEd.

The binary program that you are going to save is 132 sectors long. DOS will not normally allow you to save a file this long, so change the range limitation in DOS from 32K to 64K.

When the slave DOS was Booted, it over-wrote the code on page $08. An image of the page $08 code was saved at $2800. Move this code down to page $08.

Now all that is left is to enter the routine that will move the program back to where it will run. This routine also disconnects DOS and selects the hi-res screen before it JMPs to $600.

Add a JMP at $7FD to the move routine, and you are ready to BSAVE the file.

BSAVE GALAXIAN, A$7FD, L$8103
Now that it is safely stored on the disk, you can run it and discover how much of the memory that was saved is really used. More often than not, all of memory is not used. A large part of what was saved is not required by the program. BLOADing the file and erasing pages before you run the program is the simplest way to determine what sectors are necessary. You could determine if there is sufficient memory that is not used and retrace the Boot code in order to save the HIRES picture. This is a fine point that is not required and is left up to the reader.
Boot Code Tracing: Revisited

By Mycroft

Requirements:
Knowledge of machine language
48K Apple ][ with Integer Card
Multi-Disk Catalog by Sensible Software
One blank initialized disk

If you have a little knowledge of machine language programming, and a good measure of perseverance, this article will help you to defeat the locking scheme used in a large group of programs (those that boot up and run without subsequent disk access) and let you capture them on standard DOS. Most games and many utility and business programs fall into this category.

The Theory

The basis for this approach is: No matter what protection scheme is used, the program must boot on a standard Apple in order to run. If we could somehow step through the boot process, get everything loaded, and then stop just prior to going to the start of the program, we would be able to save the whole thing and run it under standard DOS.

The APPLE Boot process starts with Boot 0 in the Disk Controller ROM. This short machine language program BLOADS track 0, sector 0 (containing Boot 1 of the disk being booted) at locations $800 thru $8FF, and then jumps to $801 to execute Boot 1. Boot 1 reads Boot 2, and the process continues through successive Boot stages until finally the main program is loaded and run.

Assuming your disk controller card is in slot 6, the code for Boot 0 starts at $C600 and extends through $C6FA. If you power up your APPLE, enter the monitor and do a “C600L”, the beginning of the disassembly listing of Boot 0 can be seen. If you type a few more L’s you will be able to see the jump to Boot 1 at $C6F8.

What we would like to do is execute Boot 0, but stop before jumping to Boot 1. But, how is this done?

Reset

The RESET routine is at $FF59 in the monitor ROM. It performs a function similar to pushing the RESET button. If called, a RESET cycle is performed and any executing program will be stopped. If we could get Boot 0 to jump to $FF59 instead of $801, we will have accomplished our first objective.
Modifying Boot 0

Since Boot 0 is in ROM, it cannot be directly modified. The solution is to move it to RAM using the monitor’s memory move routine so we can change it to suit our needs. The new location should be somewhere in RAM where it will not be overwritten in successive Boot stages. Memory that is just below DOS is usually safe to use, since many locked programs use only slightly modified versions of normal DOS.

Because of the routine in Boot 0 that finds the slot the Disk Controller is in, the second digit of the address we move Boot 0 to must be the same as the slot number. So let’s use $9600-$96FF.

The First Step

From the monitor, type

9600<C600.C6FFM

to move Boot 0 and, instead of jumping to $801 at the end, have it jump to RESET by entering

96F8:4C 59 FF

Now put a program disk in drive 1 and execute the modified Boot 0 by typing

9600G

The drive will start, and in a second or two you should hear a beep and see the monitor prompt on the screen. Turn off the drive by typing

C0E8

Now you can have a look at Boot 1 by listing from $801 to see what it does

801L

This process can be repeated for each successive stage in the Boot process. Whenever there is a jump out to a new code section, put in good old 4C 59 FF (JMP to RESET). You can examine this code at your leisure. Don’t worry about the code at this point. Look mainly for JMP’s out.

A Practical Example

To illustrate the procedure, I will use the program “Multi-Disk Catalog III”, by Sensible Software. This program is an excellent and very useful utility (which I would recommend you purchase and add to your library.) Because I have examined dozens of
other programs which use a virtually identical Boot sequence, I am sure they can be unlocked using this same technique with only a few changes (try one of your own once-load programs if you don't have Multi-Disk Catalog). You'll need an initialized slave diskette and make sure there is a write-protect tab on the disk you are trying to unlock.

Unlocking Multi-Disk Catalog III

**BOOT 0**

Turn on your APPLE with the disk drive empty and push RESET to stop the drive. This will keep the APPLE's memory clear. Insert the locked disk and type

CALL -151

Type

9600<C600.C6FFM

to move Boot 0. Then fix the jump out by typing

96FA:98

This changes the jump out to $9801 where, in the next step, we will be moving Boot 1. At $9801 type

9801:4C 59 FF

so we can stop the Boot. Now type

9600G

After the beep, type

C0E8

to turn off the drive.

**BOOT 1**

Look at Boot 1 by typing

801L

This code relocates itself to memory page two, loads Boot 2, and jumps out to $301 at $841. Move this code so it can be modified, by typing

9800<800.8FFM

Fix the jump out to go to $9301 by typing

9843:93
and stop it at that point with

9301:4C 59 FF

One other byte in Boot 1 must be changed so that our modified code is executed properly. Type

9805:98

Now do a

9600G

once again, and type

C0E8

to stop the drive.

BOOT 2

The next stage of the Boot normally starts at $301. Move this code by typing

9300<300.3FFM

Take a look at the disassembled listing of the code beginning at $9301. The JMP out of this stage can be seen at $9343, but it is disguised by being an in direct JMP through page 0 location $3E.

If you examine the code in this Boot stage beginning at $931F, you will find that this indirect jump is used repeatedly to go to $25D, but ultimately, the indirect jump address is changed to go to the next Boot stage. This change occurs at $933A-$9341.

The final jump is determined by the byte stored in memory location $3CC, which the program increments by 1 before executing the final indirect jump. If you look at byte $3CC

$3CC

you will find that it contains value $36 (other programs commonly use $B6 here). This is the high byte of the jump-to address (the low byte has value $00). The program increments this value by 1, so the final JMP address is to $3700.

What we want to do is let the Boot do all the jumps when it is going to $25D, but stop it before it makes the final jump out to $3700.

Handling The Indirect Jump

Because zero page location $3E contains the value $5D for all the indirect jumps except the final one, we can put in a subroutine which checks to see if this value changes and, if so, stops the boot.
Enter
9000:A9 5D C5 3E D0 03 4C 5D
9008:02 4C 59 4C 59 FF

The source code for this routine would look like

```
9000-A9 5D
9002-C5 3E
9004-D0 03
9006-4C 5D 02
9009-4C 59 FF
```

- `LOA #$50` Load value.
- `CMP $3E` Same?
- `BNE $9009` No, go RESET
- `JMP $0250` Yes, go on.
- `JMP $FF59` Jump RESET.

Change the Boot code to jump to this subroutine by typing
9343:4C 00 90

and do another
9600G

**BOOT 3**

At the beep, you can stop the drive (C0E8) and examine the code beginning at $3700 looking for the next jump out. It is at $3747, and is a jump to $1B03. So, as before, change this jump by typing
374 7:4C 59 FF

**Writing To The ROM**

The code we moved to memory page $93 ($9300) was responsible for reading in this portion of the Boot. But since we just made a change at $3747, we don't want it to be over-written when we start the Boot over again. What we do is change byte $93CC so that a dummy write is done by letting it "write" to the ROM!

93CC:D0

and also changing bytes $9315 and $933E to reference this location instead of $3CC. Type

9315:93
933E:93

The "write" of the next Boot stage; therefore, begins at $D000, and is ineffectual except to keep the drive running and in the proper read mode. Change the subroutine we put in at $9000 to go to the modified next stage by typing
9009:4C 00 37
Now type

9600G

one last time. This time when you hear the beep, the drive will stop by itself.

Start listing the program at $1B03, looking for the next major jump out. You should find it at $1C25, and it is a jump to $1E54. Type

1C25:4C 59 FF
1B03G

Now list beginning at $1E54. There is an immediate jump to $9D84. List from $9D84.

Language Card?

At $9DE4 and $9DE7 are two indirect jumps, through $9D5E and $9D5C, respectively. Examine the code carefully, beginning at $9D84, and you will find that the first indirect jump is taken by systems equipped with language cards (RAM cards), and the second for those without. No matter; the second indirect address will ultimately be jumped to whichever system you have. To find out what it is, type

9DE7:4C 59 FF
9D84G

When you hear the beep, type

9D5C.9D5D

and the screen will display (low byte first) the address indirectly jumped to as $33D5. Begin listing from $33D5 and you should find the next JMP way down at $34BC; it goes to $00FD. Once again, type

34BC:4C 59 FF
33D5G

The disk drive will start and the last segment of the program will be loaded in. If everything worked correctly you should hear a beep, the drive will stop, and the screen will display garbage.

Where The Program Starts

Normally, the program would next jump via the page zero location we just changed at $34BC to the start of the multi-disk catalog main program. Type

00FDL

to see where the start is.
The software protectors have put one last obstacle in our path. $00FD takes an indirect jump through page zero locations $4E and $4F to the start of the program. We can’t examine these locations to find out where the jump goes because they get changed when a RESET cycle is executed. Not to worry, though, because if you type

348FL

you can see that locations $4E and $4F are set from $33C0 and $33C1, respectively. Typing

33C0.33C1

will thus divulge (low byte first) the starting address of the main program as $1294.

The program occupies memory from $800-$18FF, $5000-$5CFF, and $9D00-$BFFF. You find this out by scrolling through memory to try and identify program statements and data, often a trial and error process. If you get too much, no real harm is done, but too little and the program will not run.

All that remains to be done is to capture the program under normal DOS.

Moving The Memory

Warm Booting a slave diskette will over-write memory locations $800-$8FF and $9600-$BFFF, but everything from $900-$95FF will be unaffected. To move the “lower” part of the program ($800-$1800) up and out of the way of the Boot, and adjacent to the “middle” part, type

3F00<800.18FFM

Similarly, move the “top” part of the program down by typing

5D00<800.BFFF

Type the following relocation routine, when the program is BRUN, everything will return to its proper place:

3ED0:00 00 A9 5D 85 3D A9 7F
3ED8:85 3F A9 9D 85 43 20 F3
3EE0:3E A9 3F 85 3D A9 4F 85
3EE8:3F A9 08 85 43 20 F3 3E
3EF0:4C 94 12 A0 FF 84 3E C8
3EF8:84 3C 84 42 20 2C FE 60
The source code for this routine looks like:

3ED2- A9 5D  LDA #$5D  SET UP
3ED4- 85 3D  STA $3D  ADDRESS DATA
3ED6- A9 7F  LDA #$7F  FOR MEMORY
3ED8- 85 3F  STA $3F  MOVE ROUTINE
3EDA- A9 9D  LDA #$9D  FOR TOP OF
3EDC- 85 43  STA $43  PROGRAM.
3EDE- 20 F3 3E  JSR $3EF3  'MOVE' SUBR
3EE1- A9 3F  LDA #$3F  DO IT AGAIN
3EE3- 85 3D  STA $3D  FOR 'BOTTOM'
3EE5- A9 4F  LDA #$4F  PART OF PROG
3EE7- 85 3F  STA $3F  
3EE9- A9 08  LDA #$08  
3EEB- 85 43  STA $43  
3ED-  20 F3 3E  JSR $3EF3  
3EF0- 4C 94 12  JMP $1294  PROG START
3EF3- A0 FF  LDY #$FF  
3EF5- 84 3E  STY $3E  
3EF7- C8  INY  
3EF8- 84 3C  STY $3C  
3EFA- 84 42  STY $42  
3EFC- 20 2C FE  JSR $FE2C  MONITOR MEM  
3EFF- 60  RTS  MOVE

The Final Test

Now remove the protected disk from the drive, replace it with a normal DOS (slave) disk and type 6@P

(Hold the @ key down and press "P", then release the @ key and press RETURN). When the Boot is complete, type

BSAVE MDC,A$3ED2,L$412E

Your unlocked program will now BRUN normally and can be customized as you see fit.

Try this procedure with your other "one-shot" load programs. You will probably be surprised at how often it works.

Page zero locations changed by RESET:

$20-$2B
$31
$33-$3F
$40-$49
$4E-$4F
Buzzard Bait

Sirius Software Inc.

Requirements:
48K Apple, with old F8 monitor ROM
One disk drive with DOS 3.3
Initialized 48K Slave DOS 3.3 disk
Buzzard Bait

By Clay Harrell

Sirius Software always provides us with games that are challenging both in play and protection. Buzzard Bait is no exception. If you try copying the disk with your favorite nibble copier, you will find that the people at Sirius have done their homework in discovering ways to defeat you (but we have come to expect that from this fun-loving bunch).

Not being one who enjoys watching the bytes go by for hours with a copier, I tend to think there is a better way. Although the Sirius people have gone to great lengths to protect their disk from the bit bunch, they failed to protect the memory to any great extent. (A Note for Replay-Wildcard owners: Sirius hasn’t forgotten you either! Just enough disk access has been put in to discourage any easy copies).

Once the game is done with its load and the little red light goes out, RESET should be the next key pressed and the monitor prompt should appear.

Snooping through memory and checking all the “standard” starting locations reveals that an 8000G will start the game up just as if nothing happened! Further examination of memory reveals that Buzzard Bait lives from $800 to $9800.

Once the game is done with its load and the little red light goes out, RESET should be the next key pressed and the monitor prompt should appear.

Snooping through memory and checking all the “standard” starting locations reveals that an 8000G will start the game up just as if nothing happened! Further examination of memory reveals that Buzzard Bait lives from $800 to $9800.

Now all we must do is move the portions of memory that get destroyed in a Slave disk, boot to a safe location and save the game as a BLOADABLE file. Hi-res page one is a perfect candidate for this since it gets re-drawn upon starting the game and, therefore, does not need to be saved.

We must also defeat the disk access that occurs between all levels. This access does not load any data, but just checks to see if the disk is present.

In cookbook fashion, here is what we must do:

1) Boot Buzzard Bait.
2) Reset into monitor after the drive stops.
3) Move the code from $800 through $1000 up to $2000

2000<800.1000M
4) Move the code from $9600 through $9800 down to $3000
   \[3000<9600.9800M\]
5) Boot a 48K normal DOS 3.3 Slave disk
   \[60P\]
6) Reduce the number of DOS buffers to one
   \[MAXFILES1\]
7) Enter the monitor
   \[CALL -151\]
8) Move the code at $2000 through $2800 back down to $800
   \[800<2000.2800M\]
9) Move the code at $3000 through $3200 back up to $9600
   \[9600<3000.3200M\]
10) Save the first chunk of code
    \[BSAVE BAIT2,A$4000,L$5800\]
11) Make two patches that bypass the disk access between levels
    \[7FD:4C 00 20\]
    \[2000:A9 18 8D B5 B7 A9 60 8D\]
    \[2008:B6 B7 4C 00 80\]
12) Save the second chunk of code
    \[BSAVE BAIT1,A$7FD,L$1811\]
13) Enter the following Applesoft program:
    \[1 HIMEM:16284\]
    \[2 D$ = CHR$(4)\]
    \[3 PRINT D$ “MAXFILES1”\]
    \[4 PRINT D$ “BLOAD BAIT2”\]
    \[5 PRINT D$ “BRUN BAIT1”\]
14) Save the Applesoft program
    \[SAVE BUZZARD BAIT\]

This softkeyed version of Buzzard Bait you have created is not an exact copy of the original because the backup does not have any sound effects. To me this is an acceptable tradeoff for the security of not having to wear out my original game disk. Perhaps with the information I have provided, someone out there can produce a softkey procedure that retains those annoying sound effects we have all come to love.
Cannonball Blitz
Sierra On-Line, Inc.

Requirements:
Apple ][+ or equivalent
COPYA
A sector editor
Cannonball Blitz
One blank disk

By Staff

Here is a short APT for Cannonball Blitz which will reduce the hazards encountered on the second level of play. After finishing Level 1, just press the space bar and the repeat key simultaneously (or press the space bar continuously if you have automatic repeat) until the screen changes to the next level. When play begins at the second level, the number of cannons will have been reduced to only two.

And a Quick Softkey...

To copy the entire disk, use COPYA. Then, using a sector edit program such as DiskEdit, read Track 17, Sector 0E, and change address CD from 49 to 60. Finally, write the sector back to the disk. This backup copy can be copied using any of the numerous copy programs on the market. To run the program, simply boot the disk.
Casino

By Leonard Nadel, DDS

To unlock the disk Casino so it can be accessed and backed-up with COPYA, use the same method as for Zork. (See table of contents.) No changes in the sector mode are necessary!
Data Reporter
Synergistic Software

Requirements:
48K Apple ][+
COPYA
Data Reporter
One blank disk

By Don Halley

The Data Reporter, from Synergistic Software is advertised as Version 2 of the popular modifiable database system. It is basically a data storage and retrieval system with graph plotting capabilities, a text editor, and many data management features. Documentation for Data Reporter is fairly complete, although it suffers from the same unimaginative approach to organization as does the documentation for its competitors. There is no index and the table of contents offers no help beyond one or two-word references to program features. A tutorial section would be appreciated by the uninitiated user, and a reference table containing pointers to key sections would help.

Normal Copy

The documentation suggests that a copy of the original disk be made for general use in order to prolong the original’s life. The COPYA program on the DOS System Master may be used, but it will encounter a read error on the last track. This track has been written with a modified DOS and its contents are read into memory via a short machine language program appended to the last line of the HELLO program. This means that you must always boot from the original, then swap to your application disk for processing.

Unprotected Copy

A little PEEKing around will reveal that the protected sectors are from $00 to $06 on track $22, and that the information contained there is loaded into memory from $9400 to $9AFF. Examination of the HELLO program shows that it does not touch this area upon exit. This means that both a way of reading the protected portion of the disk into the proper memory locations (HELLO) and a clean exit have been provided.

Here’s the step-by-step procedure:

1) Make a copy using COPYA. (Ignore the read error.)
2) RUN the HELLO program on the original disk.
3) Choose the QUIT option from the primary menu.
4) Replace the original disk with your copy.
5) Type these commands:

    BSAVE HELLO.OBJ, A$9400, L$06FF
    LOCK HELLO.OBJ
    UNLOCK HELLO
    63999 PRINT D$ "BLOAD HELLO.OBJ" : RETURN
    SAVE HELLO
    LOCK HELLO
    PR#6

You now have a fully operational backup for only the cost of the disk itself. Of course, you may make as many additional backups as you like from this disk.
Demuffin

By Bobby

Requirements
Apple II, II+ //e with 48K
One disk drive
Blank initialized disk
DOS 3.3 Master disk with MUFFIN
Programmers Aid ROM

Muffin is a program on the 3.3 DOS Master disk. It is used to transfer 13-sector files to 16-sector disks. To accomplish this, it contains an image of 13-sector RWTS (Read or Write a Track and Sector). It uses this internal RWTS to read the file and then writes to your diskette using the external or resident DOS.

Demuffin is created by changing the jumps in the program so that it uses the resident DOS to read and the internal RWTS image to write. Also, the internal RWTS is changed from a 13 sector image to a 16 sector image.

1) Boot the DOS 3.3 Master disk
   PR#6

2) Switch to Integer and load MUFFIN
   INT
   BLOAD MUFFIN

3) Enter the monitor and initialize the Programers Aid Code-Relocation feature
   CALL -151
   D4D5G

4) Tell the relocate routine what we’re moving and where it goes
   1900<B800.BFFF8Y*

5) Move the first Code segment down to $1900
   1900<B800.BA108Y

6) Move the data segment
   .BC57M

7) Move the last code segment
   .BFFF8Y

8) Make the following patches
   1155:00 1E
9) Save the new DEMUFFIN program.

**BSAVE DEMUFFIN,A$803,L$1900**

**Directions & Explanations**

Demuffin directions are identical to Muffin’s. You want to convert the files, so you have to select the disk slot and drive and the file name you want to convert. It is helpful to type = for the file name so that all the available file names will be displayed when you answer yes to the question: “Do you want prompting?”. Now you can choose which files to convert (transfer).

If a file doesn’t transfer and you get an I/O ERROR, you’ll be returned to the menu. Just repeat all the steps but don’t bother converting the files you have already converted, and **bypass the problem file** (it probably isn’t needed by the program anyway). Continue to convert as many files as you can.
DiskEdit
By Charles Haight

Certain tools are required to understand DOS and to manipulate disk files. The first is a nibbler or bit editor. The second and most important of these is a sector editor. DiskEdit is one such utility.

DiskEdit is a user oriented direct disk access program. Simply stated, DiskEdit allows the user to read or write any sector on a disk. This means that the user can:

**Directly edit files on disk.**
- Change text in binary files.
- Insert illegal characters in REMs.
- Directly alter data base files.

**Move sectors** (even between disks).
- Repair crashed disks.

**Format catalog names.**
- Remove illegal codes in file names.
- Write flashing and inverse titles.
- Repair the VTOC.
- UnDELETE deleted files.
- Hide file names.

DiskEdit will display an entire sector as hexadecimal and ASCII.

The keyword in DiskEdit is simplicity. The commands are single key entry (you don't have to keep hitting return). With DiskEdit you can directly enter control, inverse, flashing and lower case characters. Input and display information can be in hex or decimal. The shimmering cursor is easy to identify even with a screen full of inverse and flashing characters. You can jump the cursor to any absolute position within a sector. The NEXT and LAST commands allow you to single-step through track/sectors. And DiskEdit has a simple escape. If you change your mind, pressing the escape key will set the defaults and return you to the command mode.

**Disk Overview**

Before we begin entering DiskEdit, let's take a closer look at DOS and a normal disk.

The flexible (or floppy) diskette can be thought of as a disc-shaped piece of recording tape, and essentially that's all it is. A flat disk shape is used, instead of a flat strip (as in a tape), in order to maximize the rate of data transfer. For instance, to transfer data to and from a tape, the computer would have to READ all of the tape preceding the area where the data was stored before it could transfer the required data. This method of
information retrieval is known as "sequential access" and is about the same as scanning a cassette tape for a favorite song.

The disk, on the other hand, is set up in such a way that the computer can go directly to a piece of data or program by scanning the disk laterally. This method of information retrieval is known as "random access" and is similar to selecting a particular song on a record.

Before a disk can be used, it must be formatted. The INIT command is used for this purpose.

When a disk is initialized, the Disk Operating System (DOS) writes 35 concentric tracks. Each track is divided into 16 blocks called "sectors". (DOS version 3.2 writes only 13 "sectors".) Each sector contains an address mark and a data mark. These marks start and end with a unique pattern of bytes.

The address mark tells the DOS what track/sector it is currently reading. It contains the volume, track, sector and checksum information. The data mark contains the actual data. It tells the DOS where the data begins and ends and includes a checksum that is used to verify the accuracy of the data.

If you have ever tried to load a program and the disk drive started making a slight chatter, chances are that the DOS could not read one of these markers. It then recalibrates the read/write head by moving it back to track zero and stepping (counting each track that it passes over) back out to where it was supposed to be.

The tracks are numbered from $00 (0) to $22 (34) and the sectors from $00 (0) to $0F (15). Tracks $00 through track $02 (a total of three tracks; zero, one and two) contain the DOS program.

The DOS gives the Apple the ability to manipulate data on a diskette. In this program are all of the commands related to controlling the disk drive (i.e. CATALOG, INIT, LOAD...) and a set of ERROR messages which, unless you either are a magician or don't use the Disk II, you have probably seen before.

The disk controller card that connects the Disk II to the Apple also has a small program on it. When you boot a disk, this program tells the Disk II to read track $00 (0), sector $00 (0) (remember, we start counting at zero instead of one) into memory.

The program on track $00, sector $00 contains the information required to read in sectors $00 through $09 on track $00. The program on sectors $00-$09 reads in the remaining information on track $00-$02. When this process is completed, the entire operating system (DOS) will be in memory.

At this point, DOS takes over and runs the "HELLO" program. The program that was used to initialize a disk is usually referred to as the hello or greeting program.
In order to find your “HELLO” program, DOS goes to the Volume Table of Contents (VTOC) and Directory located on track $11 (17). The VTOC and Directory are used by DOS whenever you read or write to the disk. The VTOC or “bit map” shows which sectors are in use and which are free. The second and third byte of the VTOC point to where the directory starts.

The Directory begins on sector $0F (15) and continues down to sector $01 (1). The second and third byte of each directory sector point to the next available sector. If these two bytes are zero, then there are no more sectors. The Directory contains a list of all the files on the disk. Each entry contains a pointer to the track/sector list, a file status (locked/unlocked) code, a file type code (1 letter), the file name (30 characters) and the file size. The track/sector list is a list of track/sector pairs that are used to store that program. This is why saving a blank file always takes two sectors. One for the blank file and one for the track/sector list.

DOS will read the VTOC which will point to the directory. DOS then finds the program name in the directory and finds where the track/sector list is. DOS then loads all of the track/sector pairs into the proper memory locations. Finally, DOS transfers control to the resident BASIC (Applesoft?) which will run the program.

**Entering the Program**

Enter the machine code portion of DiskEdit first. Save it to disk as ED.OBJ.,

```
BSAVE ED.OBJ, A$800, L$A21
```

Enter the BASIC listing and save it to disk as ED.BAS.

```
SAVE ED.BAS
```

Bload the binary file.

```
BLOAD ED.OBJ
```

Type “RUN” and press return. After the “?UNDEF’D STATEMENT ERROR” message, run ED.BAS.

```
RUN ED.BAS
```

This will combine the two programs to form DiskEdit. Type ‘X’ to exit to BASIC. Now, insert a blank disk in the drive and type ‘INIT DISK EDIT’. Use this back up copy for the following examples and ALL other uses.

**Getting Familiar**

This exercise will aid you in understanding how to use the commands by taking you on a tour of a normal DOS diskette.
Please read each paragraph before pressing any keys and follow the directions carefully.

Insert the DiskEdit back-up disk in Drive 1. Turn on your computer. DiskEdit will prompt you when it is ready.

Press any key to start.

What is your status?

On the bottom of the screen are the status indicators and prompts. They tell you the slot (SL), drive (DR), track (T), sector (S), volume (V), byte position (B), filter (F) and data entry mode currently selected.

Reading

Press the ‘R’ key. This tells DiskEdit that you want to READ a sector from the disk. A flashing prompt will appear next to the track (T) indicator. DiskEdit is asking you what track to read.

Type '01'. This tells DiskEdit that you wish to read track $01 (1). The flashing prompt will move over to the sector (S) indicator. Respond to this prompt by typing ‘8’.

The disk drive should whirr for about two seconds, and then stop. The screen should be full of numbers and letters. You are now looking at the contents of track $01 (1), sector $08 (8) in what is known as hex or hexadecimal format on the left side of your screen and ASCII on the right side.

Hex a what?

Hexadecimal is a base sixteen numbering system. It gets its name from the fact that it contains all of the numbers found in normal base 10 (decimal 0-9) plus six alphabetic characters (A thru F).

Say ‘AS-KEY’

ASCII stands for ‘American Standard Code for Information Interchange.’ This is the alphanumeric equivalent of all of those hex symbols on the right.

Error messages

The sector you are now viewing ($08) contains the DOS error messages (they are continued on sector $09).

Press the ‘N’ key. This will increment the sector count and cause Diskedit to read the next sector. If the sector count had been at $0F (15), the track count would have been incremented by one and the sector count reset to $00 (0).

The “Boot” Program

You are now viewing the sector where the “Boot” program name is stored. In the center of the screen is the file name ‘DISK EDIT’. This is the name of the program that the DOS
will automatically ‘RUN’ when this disk is booted. (If you decide later to change the boot program name on this disk, this is where you should come.)

Let’s follow how DOS located the file “DISK EDIT” when you booted this disk.

Press ‘R’ to read. Type ‘11’ for the track and ‘0’ for the sector.

You are looking at the VTOC or bit map. The second and third byte point to the first directory (catalog) sector. These bytes should be ‘11 0F’.

Press ‘R’ and type ‘11’ for the track and ‘F’ for the sector.

The sector you are viewing is the first part of the directory, which extends downward to sector $01 (1). Press the zero key. This is a special function key designed to make viewing catalog sectors more meaningful. The screen will return to normal when you press any other key.

Moving the cursor

The I, J, K and M keys are the cursor movement keys. The cursor has a wrap around feature. If you go off the screen on one side, you will come back on the opposite side.

Press the ‘O’ key. The flashing prompt will appear next to the byte position (B) indicator.

This command allows us to move the cursor to a specific location on the screen. Move the cursor to the beginning of the file name by typing ‘0E’. The cursor should now be in front of the ‘D’ of “DISK EDIT”.

Move the cursor back one character by pressing ‘J’. Look at the hex portion of your screen. The ‘02’ is used by DOS to tell what type of program DiskEdit is and whether it is locked or unlocked. The ‘0’ means that the file is unlocked. The ‘2’ means the file is Applesoft.

Editing

Press the ‘E’ key. This tells DiskEdit that you wish to edit the sector.

Type ‘82’. Press ‘ESC’ to exit the EDIT mode. Press the ‘O’ key. Type ‘2C’. The byte you are looking at and the ‘00’ following it are the hex equivalent of the sector use count for the file. Press the ‘E’ key. Type ‘00’. Press ‘ESC’ to exit the EDIT mode.

Press the zero key.

The program HELLO is shown with an asterisk. Changing the ‘02’ into a ‘82’ locked the file. Entering the ‘00’ will change the sector count for the file to zero.
Writing

WARNING: Read the following paragraph completely before you press any keys.

Up to this point, you have only been editing the disk information that is in the computer’s memory. In order to make the changes permanent you need to WRITE this information back to the disk.

The command to do this is ‘W’ for WRITE. Press the ‘W’ key. Press ‘RETURN’ for the track (T) and sector (S).

When the RETURN key is pressed in response to a prompt the program will act as if the default values were entered. The default values for the track and sector are the last track/sector that was read or written.

The program will beep and a warning will be printed. This is your last chance to change your mind. You must press RETURN to have DiskEdit write to your disk. Any other key will abort this operation.

Press RETURN. The buffer contents are now written to the disk. Press the ‘C’ key to see the catalog. The first file will be locked (indicated by the asterisk ‘*’ next to the file type) and the sector count will be ‘000’. Press any key to continue.

This completes the exercise. Experiment with DiskEdit using this same scratch diskette.

Summary of Commands

ESC This is the “I changed my mind” key. Press this key to reset defaults and exit back to the command mode.

RTN The RETURN key, when used to answer an input prompt, will accept the current default and continue. (Example: When prompted for the track and sector during a read command, pressing RETURN twice will cause the current track and sector to be read.)

> Track skip command. Increments the track number and performs a READ. Does not increment the sector.

< Track skip command. Decrements the track number and performs a READ. Does not decrement the sector number.

A Sets character entry mode to ASCII

B Disassemble buffer command. Calls the monitor to disassemble buffer contents starting at the cursor location. Use the space bar to continue disassembly one line at a time or press RETURN to disassemble 20 additional lines. Press ‘P’ to print the screen display. (Press ESC to exit.)

C Displays the disk catalog using the current slot and drive. Prints the number of free sectors on the disk.

D Flips the active drive from 1 to 2 or from 2 to 1 on each keypress.
A continuous-edit mode, this mode allows you to type changes just like on a typewriter. Pure cursor movement is supported using control keys. If you are in hexadecimal format, only valid hex digits are accepted as input. In ASCII format all keys are valid except the control keys listed below. (Press ESC to exit.)

<table>
<thead>
<tr>
<th>Ctrl Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>set FLASH mode</td>
</tr>
<tr>
<td>I</td>
<td>set INVERSE mode</td>
</tr>
<tr>
<td>N</td>
<td>set NORMAL mode</td>
</tr>
<tr>
<td>Q</td>
<td>move cursor up</td>
</tr>
<tr>
<td>Z</td>
<td>move cursor down</td>
</tr>
<tr>
<td>→</td>
<td>move cursor right</td>
</tr>
<tr>
<td>←</td>
<td>move cursor left</td>
</tr>
</tbody>
</table>

+ This edit submode is entered using the plus (+) key. The ‘>>EDIT<<’ prompt is changed to ‘++EDIT++.’ It is identical to the normal edit mode except that it does not support control functions. All keys are valid except ESC. Control characters may be directly entered. The plus (+) key or the semi-colon (;) may be used to enter this submode.

F This is the filter format command it allows you to change the filter values so that you can configure your own filters.

G Turns the sound on or off each time you press the ‘G’ key. (Default at BOOT is on.)

H Sets character entry mode to Hexadecimal

I Moves cursor up.

J Moves cursor left.

K Moves cursor right.

M Moves cursor down.

L Reads last sector.

N Reads next sector.

O Allows cursor to be jumped to any absolute position in the displayed sector.

P Sends the buffer contents to your printer. A header is printed first which shows the track, sector, and volume. When first used, the program will ask which slot your printer is using and whether you wish to use 40 or 80 columns.

R Prompts you for the track and sector to read. Use the RETURN key to accept default values.

S Prompts you for a new slot. Valid entries are from 1 to 7.

U Toggles the status indicators between hex and decimal and updates the display information. Only the track, sector, and cursor are affected by this key. (Default at BOOT is hex.)

W Prompts you for the track and sector to write to. Use the RETURN key to accept default values. After entering the track
and sector, DiskEdit will beep and pause. This is your last chance to change your mind. Press RETURN to WRITE, or any other key to escape.

X Clears the screen and exits to BASIC.

ASCII Filters

The number following the filter (F) indicator is the filter currently selected.

There are 9 filters. Each affects the format of the displayed screen contents. They do not change the actual buffer contents in any way. They may be selected by pressing the corresponding number (1-9) key.

Rolling your own

The filters can be modified from the keyboard. Select a filter (1-9) by pressing the appropriate number key. Press the ‘F’ key.

The 256 screen characters are divided into 8 blocks. The prompt under ‘BLOCK’ indicates the original group of characters while the prompt under ‘CHG:’ indicates what characters will be displayed on the screen.

The first prompt is ‘INV1’ for inverse letters. Press ‘7’. This causes all inverse characters in block 1 to display as normal.

Block 7 is normal letters. The ‘INV1’ prompt under ‘CHG:’ will change to ‘NOR2’. By pressing a number from 1 to 8, each of the original blocks can be changed to display as any other block. Pressing ‘RETURN’ will skip a block.

Next to ‘CHG:’ is ‘FN#’. The ‘FN#’ is short for function number. There are 3 functions.

1. Print block, delete one character
2. Delete block, print one character
3. Delete entire block

Customizing the Program

DiskEdit is an Applesoft program with packed machine code. This means that the machine code portion of the program is hidden in such a way that DOS thinks it is part of the Applesoft program.

The machine code is hidden behind the REM in line 0 rather than at the end of the BASIC program. This was done in order to allow program modification while keeping the program size as small as possible.

If you load the program and list it, you will see a single BASIC line:

0 CALL 2167 : GOTO 10 : REM
In order to make changes you will need to follow these steps:

1. RUN the program.
2. When the copyright notice is on the screen, press RESET to exit the program.
3. LIST the program and make changes.
4. After making any changes, RUN the program and exit using the "X" key. This will change the zero page pointers so that DOS can save the machine code along with the modified program.
5. SAVE the modified program to disk.

**DiskEdit BASIC program**

```basic
10 TEXT: HOME: GOSUB 2150: GOTO 750
20 REM CLEAR TEXT WINDOW
30 POKE 35,21: HOME: RETURN
40 REM GET CHARACTER WITH PROMPT
50 POKE -16368,0
60 GET N$: KY = ASC (N$) + 128: IF KY <> 155 THEN RETURN
70 REM RESET ALL DEFAULTS
80 POKE TS, TK: POKE SS, SE: CALL TT: CALL MV
90 REM CLEAR STACK, GOTO CMD PARSER
100 CALL -10621: GOTO 750
110 REM MAKE NOISE AND RETURN
120 PRINT G$: RETURN
130 REM FIND BINARY START
140 IF PEEK (1024) = 164 THEN 190
150 REM FOR DECIMAL NUMBER
160 A1 = PEEK (1024) - 176: A2 = PEEK (1025) -176: IF A2 > -1 THEN GOSUB 400: A1 = KY: RETURN
170 KY = A1: RETURN
180 REM FOR HEX NUMBER
200 REM GET KEY WITHOUT PROMPT
210 KY = PEEK (-16384): IF KY < 128 THEN 210
220 POKE -16368,0: RETURN
230 REM HANDLE AN ERROR
250 PRINT "DISK'DRIVE'ERROR"
260 PRINT G$: RETURN
270 REM PROCESS HEX/DEC INPUT
280 KY = KY - 176: IF KY < 0 OR KY > 22 THEN KY = 128: RETURN
290 IF KY > 9 THEN KY = KY - 7: IF KY < 10 OR KY > 15 THEN KY = 128
300 RETURN
310 REM GET HEX OR DEC ONLY
```

51 Book of Softkeys Vol. I
320 GOSUB 50
330 IF KY = 141 THEN RETURN
340 GOSUB 280
350 IF KY = 128 THEN GOSUB 120: GOTO 320
360 IF PEEK (HF) AND KY > 9 THEN GOSUB 120: GOTO 320
370 RETURN
380 REM CALCULATE HEX/DEC NO.
390 IF NOT PEEK (HF) THEN KY = A1 * 16 + A2: RETURN
400 KY = A1 * 10 + A2: RETURN
410 REM GET TRACK VALUE
430 IF NOT PEEK (HF) AND KY > 2 THEN 480
440 IF KY > 3 THEN 480
460 A2 = KY: GOSUB 390
470 REM CHECK FOR VALID TRACK#
480 IF KY < 0 OR KY > 34 THEN PRINT G$: GOTO 420
490 REM SAVE OLD TRK#, POKE NEW
500 TS = TK: TK = KY: POKE TR, TK: CALLTT
510 REM GET SECTOR VALUE
530 REM CHECK FOR HEX I/O
540 IF NOT PEEK (HF) THEN 620
550 REM SAVE KEY
560 IF KY > 1 THEN 620
570 REM GET ANOTHER KEY
590 REM CHECK FOR VALID SECTOR#
600 A2 = KY: GOSUB 390: IF KY < 0 OR KY > 15 THEN PRINT G$: GOTO 520
610 REM SAVE OLD SCT#, POKE NEW
620 SS = SE: SE = KY: POKE SC, SE: CALLTT
630 REM IF WRITE THEN LAST CHANCE
640 IF PEEK (CM) = WR THEN VTAB24: HTAB 2: PRINT "PRESS RETURN"TO->"; FL4SH: PRINT "WRITE"; NORMAL: PRINT "<-", "ESC" TO "EXIT" G$: NORMAL: POKE – 16368, 0: GOSUB 210: IF KY <> 141 THEN 80
650 GOTO 710
660 REM PRINT 40 "=" "S
670 FOR X = 1 TO 40: PRINT "="": NEXT: RETURN
680 REM PRINT SCREEN PROMPTS
690 CALL TT
700 REM READ OR WRITE A SECTOR
710 CALL 10
720 REM PRINT BUFFER TO SCREEN
730 CALL MV: RETURN
740 REM COMMAND PARSER
750 POKE 216, 0: CALL TT: VTAB 23: HTAB 1: CALL – 958: IF PEEK(EF) > 0 THEN
GOSUB 240

760 REM SAVE CURRENT TRACK/SECTOR
770 TS = PEEK (TR): SS = PEEK (SC): TK = TS: SE = SS
780 CALL XC: KY = PEEK (225) - 192
790 IF KY = -5 OR KY = -21 THEN 1380
800 IF KY < 0 OR KY > 26 THEN 750
810 ON KY GOSUB 100, 1870, 1830, 100, 1400, 840, 1450, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 1590, 1480, 100, 420, 1680, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100:
GOTO 750
820 PRINT G$;: GOTO 750
830 REM *** DEFINE FILTER ***
840 TEXT: HOME: VTAB 22: HTAB 7: PRINT "CONFIGURATION" FOR "FILTER#" PEEK (FL)
850 VTAB 2: PRINT G$ "# BLOCK **** CHG: **** FN# CHR$ STATUS"
860 PRINT
870 DL = PEEK(231) + PEEK(232) * 256 - 1: CG = PEEK(233) + PEEK(234) * 256 -1
880 FI = PEEK (FL)
890 REM PRINT CURRENT VALUES
900 FOR X = 1 TO 8: PRINT X" !F$(X)" "->";
910 F = PEEK (CG + X)
920 F1 = INT (F / 32) + X: IF F1 > 8 THEN F1 = F1 - 8
930 F2 = F - (INT (F / 32) * 32)
940 F3 = PEEK (DL + X)
950 F4 = PEEK (NO + FI)
960 F1(X) = F1: F2(X) = F2: F3(X) = F3 + (F(F1) * (F2 <> 0)) + (F(X) * (F2 = 0))
970 PRINT F1" !F$(F1) ;: HTAB 23: PRINT F2; ;: HTAB 27: POKE 2091, F3: CALL HP: CALL AP: IF X <> 1 THEN 1000
980 HTAB 36: IF F4 = 1 THEN PRINT "ON" ;
990 IF F4 = 0 THEN PRINT "OFF" ;
1000 PRINT : PRINT : NEXT
1010 REM EDIT CURRENT VALUES
1020 FOR X = 1 TO 8: VTAB X * 2 + 2: HTAB 12
1030 REM GET BLOCK #
1040 GOSUB 50: A = KY - 176: IF N$ = CHR$ (13) THEN A = F1(X): N$ = ""
1050 IF A < 1 OR A > 8 THEN PRINT G$;: GOTO 1040
1060 PRINT N$; ;: HTAB 15: PRINT F$(A); ;: HTAB 23
1070 C = F2(X)
1080 REM CALCULATE OFFSET
1090 IF A > = X THEN F = A - X
1100 IF A < X THEN F = (8 - X) + A
1110 POKE CG + X, F * 32 + C
1120 REM GET FUNCTION #
1130 GOSUB 50: C = KY - 176: IF N$ = CHR$ (13) THEN C = F2(X): N$ = ""
1140 IF C < 0 OR C > 3 THEN PRINT G$: GOTO 1130
1150 PRINT N$;
1160 REM CHANGE FILTER VALUE
1170 POKE CG + X, F * 32 + C

Gosub 240

Book of Softkeys Vol. I
1180 KY = F3(X): IF C = 0 THEN KY = 0
1190 IF C < 1 OR C = 3 THEN 1270
1200 VTAB 20: HTAB 1: PRINT "ENTER CHARACTER:"; : GOSUB 50: IF KY = 141 THEN
   KY = F3(X)
1210 IF KY < 160 OR KY > 223 THEN PRINT G$; : GOTO 1200
1220 IF KY < 192 THEN KY = KY + (2 + A) * 32: GOTO 1240
1230 IF KY > 191 THEN KY = KY + (1 + A) * 32
1240 KY = KY - 256: HTAB 1: CALL - 868: VTAB X * 2 + 2
1250 POKE DL + X, KY
1260 HTAB 27: POKE 2091, KY: CALL HP: CALL AP
1270 NEXT

1280 REM GET FILTER STATUS
1290 PRINT: PRINT: PRINT "LEAVE FILTER ON DURING EDIT? (Y/"; INVERSE:
   PRINT "N"); : NORMAL: PRINT ");: GOSUB 50
   "ON"; : GOTO 1320
1310 PRINT "OFF"; : A = 0
1320 POKE NC, O + 1, A
1330 REM RESTORE SCREEN, EXIT
1340 FOR X = 1 TO 500: NEXT
1350 GOTO 730
1360 REM ++EDIT++ MODE ENTRY POINT
1370 IF FI = 0 THEN RETURN
1380 VTAB 24: HTAB 2: INVERSE: PRINT "++EDIT++"; : POKE NC, 0: GOTO 1410
1390 REM EDIT MODE ENTRY POINT
1400 VTAB 24: HTAB 2: INVERSE: PRINT ">>EDIT<<"; : POKE NC, 1
1410 NORMAL: HTAB 12: PRINT "MODE"; ;
1420 PRINT "***PRESS<ESC>"TO"EXIT";
1430 CALL ED: VTAB 23: HTAB 1: CALL - 958: GOTO 80
1440 REM TURN SOUND ON/OFF
1450 PRINT G$;: IF G$ = CHR$ (7) THEN G$ = "": RETURN
1460 G$ = CHR$ (7): RETURN
1470 REM *** PRINT HARDCOPY ***
1480 IF NOT PR THEN GOSUB 1760
1490 GOSUB 30
1500 A1 = PEEK (BF) * 256 - 1
1510 PR#: PR: PRINT
1520 PRINT "TRACK:*"; : POKE NM, TK: CALL HX: PRINT "SECTOR:"; : POKE NM, SE:
   CALL HX: PRINT "VOLUME:"; : PEEK (VO)
1530 FOR X = 0 TO 255 STEP 16 /L/: POKE NM, X: CALL HX: HTAB 5: PRINT "-";
1540 FOR A = 1 TO 16 /L/: POKE2091, PEEK (A1 + X + A): CALL HP: NEXT
1550 FOR A = 1 TO 16 /L/: POKE2091, PEEK (A1 + X + A): CALL AP: NEXT
1560 PRINT: NEXT
1570 PR#: 0: GOTO 80
1580 REM *** JUMP CURSOR ***
1590 VTAB 22: HTAB 32 - PEEK (HF): GOSUB 320: IF KY > 15 THEN CALL TT:
   RETURN

Book of Softkeys Vol. I 54
1610 A2 = KY: PRINT NS;: GOSUB 390: IF NOT PEEK (HF) THEN 1660
1620 IF KY > 25 THEN 1660
1640 A2 = KY: PRINT NS;: GOSUB 390: IF KY < 0 OR KY > 255 THEN CALL TT: GOTO 1590
1650 REM CALCULATE NEW CURSOR POSN
1660 POKE CS, KY: CALL MV: CALL TT: RETURN
1670 REM CHANGE SLOT NO.
1680 VTAB 22: HTAB 4: GOSUB 320: IF KY > 15 THEN CALL TT: RETURN
1690 IF KY < 1 OR KY > 7 THEN 1680
1700 POKE SL, KY * 16: CALL TT: RETURN
1710 REM WRITE A TRACK/SECTOR
1720 GOSUB 30: VTAB 12: PRINT "WHICH SLOT IS YOUR PRINTER USING? 1-7":
1730 REM CLEAR SCREEN, RECONNECT DOS AND EXIT TO BASIC
1740 TEXT: HOME: POKE 103, 1: POKE 104, 8: CALL 1002: END
1750 REM FIND PRINTER SLOT
1760 GOSUB 30: VTAB 12: PRINT "WHICH SLOT IS YOUR PRINTER USING?":
1770 IF KY > 7 THEN GOSUB 120: GOTO 1760
1780 IF NOT KY THEN RETURN
1790 PR = KY: LI = 2
1800 PRINT: PRINT: PRINT TAB(6) "PRINT USING"80"COLUMNS"(Y/";: INVERSE:
1810 REM CALL FOR CATALOG
1820 CALL 1002: ONERR GOTO 1850
1830 GOSUB 30: PRINT: PRINT CHR$(4) "CATALOG,D" PEEK (DR),", S" PEEK (SL) /
1840 GOSUB 30: PRINT: PRINT TAB(6) "PRINT USING"80"COLUMNS"(Y/";: INVERSE:
1850 POKE 216, 0: GOTO 240
1860 REM DISASSEMBLE THE BUFFER
1870 GOSUB 30: VTAB 21: PRINT: PRINT: KY = PEEK (CS)
1880 REM START AT CURSOR
1890 POKE 58, KY: POKE 59, PEEK (BF)
1900 A1 = 0: A2 = 21
1910 REM START AT LAST BYTE
1920 FOR X = 1 TO A2: IF PEEK (59) > PEEK (BF) THEN : A1 = 1: IF PEEK (1152) <
1930 IF A1 THEN PRINT: NEXT: GOTO 2090
1940 CALL BI
1950 NEXT
1960 REM <ESC> KEY? = EXIT
1970 GOSUB 210: IF KY = 155 THEN 2130
1980 REM <RTN> KEY? = 20 LINES
1990 IF KY = 141 THEN 1900
2000 REM <SPACE> KEY? = 1 LINE
2010 IF KY = 160 THEN A2 = 1: GOTO 1920
2020 IF KY = 213 THEN GOSUB 140: GOSUB 1550: VTAB 1: GOTO 1890

55 Book of Softkeys Vol. I
2030 IF KY <> 208 THEN 1970
2040 REM PRINT SCREEN
2050 GOSUB 140: L = KY
2060 IF NOT PR THEN GOSUB 1760
2070 HOME: KY = L: PR# PR: GOTO 1890
2080 REM LAST CHANCE TO PRINT
2090 PR INT "ENO'OF'BUFFER'PRESS'RETURN'TO'CONTINUE" ; GOSUB 210
2100 REM DEFINE VARIABLES
2110 RD = 1: WR = 2: LI = 2
2120 SL = 2071: OR = 2072: VO = 2084: TR = 2074: SC = 2075: CM = 2082
2130 NM = 2091: FL = 2101: EF = 2094: HF = 2095: CS = 2100: BF = 2103
2140 NC = 2099
2150 F1 = PEEK (FL)
2160 NO = PEEK (2106) + PEEK (2107) * 256
2180 HP = 2141: AP = 2144
2190 F$(1) = "INV1": F$(2) = "INV2": F$(3) = "FLS1": F$(4) = "FLS2": F$(5) = "CTRL": F$(6) = "NOR1": F$(7) = "NOR2": F$(8) = "L/C"
2200 F(1) = 192: F(2) = 128: F(3) = 128: F(4) = 64: F(5) = 64: F(6) = 0: F(7) = 0: F(8) = -64
2210 G$ = CHR$ (7)
2220 VTAB 8: PRINT "O·'·S·K·E·O·I·T .... V·E·R·S·'·0·N ... 4· "0": PRINT "'COPYR IGHT'1981' (C) 'HARDCORE'COMPUT 1ST": PR INT
2230 HTAB 5: FOR X = 1 TO 32: PRINT "-" ; NEXT: PRINT : HTAB 6: PRINT "A'O ISK'EO ITI NG'UT ILLI TY'PROGRAM"
2240 HTAB 5: FOR X = 1 TO 32: PRINT "-" ; NEXT: PRINT : PRINT
2250 Diskedit source code

0010
0015 * DISKEDIT II - VERSION 4.1
0020 * COPYRIGHT 1981 SOFTKEY
0025 * LAST UPDATED MAR 24 84
0030
0035 .OR $800
0040 .TF EDO
0045
0050 WNDTOP .EQ $22
0055 WNBRTM .EQ $23
0060 CH .EQ $24
0065 CV .EQ $25
0070 BASE2 .EQ $26,27
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0028-</td>
<td>0075 BASE1</td>
<td>$28,29</td>
</tr>
<tr>
<td>003A-</td>
<td>0080 PCL</td>
<td>$3A,3B</td>
</tr>
<tr>
<td>0048-</td>
<td>0085 10BPL</td>
<td>$48</td>
</tr>
<tr>
<td>0067-</td>
<td>0090 PRGSTR</td>
<td>$67</td>
</tr>
<tr>
<td>00E0-</td>
<td>0095 LOC</td>
<td>$E0</td>
</tr>
<tr>
<td>00E1-</td>
<td>0100 NUM</td>
<td>$E1</td>
</tr>
<tr>
<td>00E4-</td>
<td>0105 BUFFER.PINTER</td>
<td>$E4</td>
</tr>
<tr>
<td>00E7-</td>
<td>0110 DCHR</td>
<td>$E7,E8</td>
</tr>
<tr>
<td>00E9-</td>
<td>0115 CFLT</td>
<td>$E9,EA</td>
</tr>
<tr>
<td>03D9-</td>
<td>0120 RWTS</td>
<td>$3D9</td>
</tr>
<tr>
<td>03E3-</td>
<td>0125 GET1OB</td>
<td>$3E3</td>
</tr>
<tr>
<td>CO00-</td>
<td>0130 KEY</td>
<td>$C000</td>
</tr>
<tr>
<td>C010-</td>
<td>0135 STROBE</td>
<td>$C010</td>
</tr>
<tr>
<td>B3F2-</td>
<td>0140 VTOC</td>
<td>$B3F2</td>
</tr>
<tr>
<td>ED24-</td>
<td>0145 LNPRT</td>
<td>$ED24</td>
</tr>
<tr>
<td>F88C-</td>
<td>0150 INSDS</td>
<td>$F88C</td>
</tr>
<tr>
<td>F8D3-</td>
<td>0155 INSTDS</td>
<td>$F8D3</td>
</tr>
<tr>
<td>F94A-</td>
<td>0160 PRBLANK</td>
<td>$F94A</td>
</tr>
<tr>
<td>F953-</td>
<td>0165 PCADJ</td>
<td>$F953</td>
</tr>
<tr>
<td>FC58-</td>
<td>0170 HOME</td>
<td>$FC58</td>
</tr>
<tr>
<td>FC62-</td>
<td>0175 CR.LF</td>
<td>$FC62</td>
</tr>
<tr>
<td>FDDA-</td>
<td>0180 PRHEX</td>
<td>$FDDA</td>
</tr>
<tr>
<td>FDED-</td>
<td>0185 COUT</td>
<td>$FDED</td>
</tr>
<tr>
<td>0190-</td>
<td>0195 CTRL.AT</td>
<td>$80</td>
</tr>
<tr>
<td>0081-</td>
<td>0200 CTRL.A</td>
<td>$81</td>
</tr>
<tr>
<td>0082-</td>
<td>0205 CTRL.B</td>
<td>$82</td>
</tr>
<tr>
<td>0084-</td>
<td>0210 CTRL.D</td>
<td>$84</td>
</tr>
<tr>
<td>0086-</td>
<td>0215 CTRL.F</td>
<td>$86</td>
</tr>
<tr>
<td>0088-</td>
<td>0220 CTRL.H</td>
<td>$88</td>
</tr>
<tr>
<td>0089-</td>
<td>0225 CTRL.I</td>
<td>$89</td>
</tr>
<tr>
<td>008C-</td>
<td>0230 CTRL.L</td>
<td>$8C</td>
</tr>
<tr>
<td>008D-</td>
<td>0235 RETURN</td>
<td>$8D</td>
</tr>
<tr>
<td>008E-</td>
<td>0240 CTRL.N</td>
<td>$8E</td>
</tr>
<tr>
<td>0091-</td>
<td>0245 CTRL.Q</td>
<td>$91</td>
</tr>
<tr>
<td>0095-</td>
<td>0250 CTRL.U</td>
<td>$95</td>
</tr>
<tr>
<td>009A-</td>
<td>0255 CTRL.Z</td>
<td>$9A</td>
</tr>
<tr>
<td>009B-</td>
<td>0260 ESCAPE</td>
<td>$9B</td>
</tr>
<tr>
<td>00A0-</td>
<td>0265 SPACE</td>
<td>$A0</td>
</tr>
<tr>
<td>00AA-</td>
<td>0270 STAR</td>
<td>$AA</td>
</tr>
<tr>
<td>00AE-</td>
<td>0275 PERIOD</td>
<td>$AE</td>
</tr>
<tr>
<td>00B5-</td>
<td>0280 FIVE</td>
<td>$B5</td>
</tr>
<tr>
<td>00C9-</td>
<td>0285 LTR.I</td>
<td>$C9</td>
</tr>
<tr>
<td>00CA-</td>
<td>0290 LTR.J</td>
<td>$CA</td>
</tr>
<tr>
<td>00CB-</td>
<td>0295 LTR.K</td>
<td>$CB</td>
</tr>
<tr>
<td>00CD-</td>
<td>0300 LTR.M</td>
<td>$CD</td>
</tr>
<tr>
<td>0305-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Book of Softkeys Vol. I**

57
0310 * ------------------------ 1ST LINE OF BASIC PROGRAM
0315
0800- 00 11 08
0803- 00 00 BC
0806- 32 30 0320 START .HS 0010800008C3230
0808- 36 37 3A .HS 36373AAB31303AB2
080B- AB 31 30 .HS 000000
0810- 00 00 00 0330 .JMP INITDOS 0335
0813- 4C 73 08 0335
0816- 01 0355 IOBIND .HS 01
0817- 60 0360 SLOT .HS 60 SLOT * 16
0818- 01 0365 DRIVE .HS 01 DRIVE #
0819- 00 0370 EXPVOL .HS 00 REQ. VOLUME
081A- 00 0375 TRACK .HS 00 TRACK #
081B- 00 0380 SECTOR .HS 00 SECTOR #
081C- 27 08 0385 .DA DCT
081E- 00 09 0390 .DA BUFFER
0820- 00 00 0395 .HS 0000
0822- 01 0400 CMND .HS 01 COMMAND
0823- 00 0405 ERCODE .HS 00 ERROR CODE
0824- 00 0410 VOLUME .HS 00 VOLUME #
0825- 60 0415 OLDSLOT .HS 60 PREV. SLOT
0826- 01 0420 OLDRIYE .HS 01 PREV. DRIVE
0425
0827- 00 0430 DCT .HS 00 TYPE CODE
0828- 01 0435 PHASES .HS 01 PHASES/TRK
0829- EF D8 0440 .HS EFDD TIME COUNT
0445
0450 * ------------------------ BASIC variables
0455
082B- 00 0460 BYTE .HS 00 NM
082C- 00 0465 OLDTRK .HS 00 OT
082D- 00 0470 OLDSCST .HS 00 OS
082E- 00 0475 ERRFLG .HS 00 EF
082F- 00 0480 HEX.OR.DEC.FLG .HS 00 HF
0830- 01 0485 ON.OFF .HS 01 ST
0831- 01 0490 CFLG .HS 01 PF
0832- 00 0495 .HS 00
0833- 00 0500 USE.CTRL.CHARS .HS 00 TH
0834- 00 0505 CRSVAL .HS 00 CS
0835- 01 0510 FLTNM .HS 01 FL
0836- 00 0515 .HS 00
0837- 09 0520 .DA /BUFFER BF
0838- 00 00 0525 .HS 0000
083A-8B 0C 0530 .DA FSTAT NO
083C-00 00 00 0535 .HS 000000
0540
0545 * ---------------BASIC Call table
0550
083F-4C 90 08 0555 JMP CALLIO 10
0842-4C 29 0A 0560 JMP PRINT.SCREEN.DATA
0845-4C 0E 12 0565 JMP HXBYTE
0848-4C 7E 0F 0570 JMP EDIT ED
084B-4C 00 0A 0575 JMP BINARY BI
084E-4C 6B 0B 0580 JMP CALC.FREE.SECTORS FR
0851-4C C9 0C 0585 JMP PROMPT TT
0854-4C BD 0C 0590 JMP PROMPTO T1
0857-4C 5D 0E 0595 JMP PARSE XC
085A-4C AE 0B 0600 JMP FILTERO HC
085D-4C 03 12 0605 JMP HEXPRINT HP
0860-4C DA 11 0610 JMP ASCPRINT AP
0863-4C 0A 0E 0615 JMP RIGHT UNUSED
0866-60 60 60 0620 .HS 606060 UNUSED
0625
0630 * ----------------------------- INTERNAL VARIABLES
0635
0869-00 0640 OFFSET .HS 00
086A-FF 0645 FIRST .HS FF
086B-01 0650 EDFLG .HS 01
086C-00 0655 HCOUNT .HS 00
086D-00 0660 SPACES .HS 00
086E-00 0665 EDIT.MODE.FLAG .HS 00
086F-01 0670 KEYFLG .HS 01
0870-10 0675 MAXSCT .HS 10
0871-23 0680 MAXTRK .HS 23
0872-00 0685 SPECIAL.FUNCTION .HS 00
0690
0695 * ----------------------------- Get DOS pointers
0700
0873-20 E3 03 0705 INITDOS JSR GETIOB
0876-84 48 0710 STY IOBPL
0878-85 49 0715 STA IOBPL+1
087A-A0 01 0720 LDY #1
087C-B1 48 0725 LDA (IOBPL),Y
087E-8D 17 08 0730 STA SLOT
0881-C8 0735 INY
0882-B1 48 0740 LDA (IOBPL),Y
0884-8D 18 08 0745 STA DRIVE
0750 * ------------------------------ Reset program pointer
0887-A9 1F 0755 LDA #STOP
0889-85 67 0760 STA PRGSTR
088B-A9 12 0765 LDA /STOP

59 Book of Softkeys Vol. I
088D- 85 68 0770  STA PRGSTR+1
088F- 60 0775  RTS
0780
0785 * ---------------------------Call Read/Write Track Sector
0790
0890- A9 08 0795 CALL10  LDA /IOBIND
0892- A0 16 0800  LDY #IOBIND
0894- 20 D9 03 0805  JSR RWTS
0897- 90 06 0810  BCC .1
0899- AD 23 08 0815  LDA ERCODE  GET ERROR #
089C- 8D 2E 08 0820  STA ERRFLG
089F- 60 0825 .1  RTS
0830
0835 * ----------------------------------Put buffer here
0840
08A0- 0845  .BS $900- *
0900- 0850 BUFFER  .BS $100 256 bytes
0855
0860 * ---------------------------Disassemble an instruction
0865
0A00- A9 8D 0870 BINARY  LDA #RETURN
0A02- 20 ED FD 0875  JSR COUT  PRINT <CR>
0A05- A5 3A 0880  LDA PCL
0A07- 8D 2B 08 0885  STA BYTE
0A0A- 20 OE 12 0890  JSR HXBYTE
0A0D- A9 04 0895 STEP  LDA #4
0A0F- 85 24 0900  STA CH
0A11- A9 AD 0905  LDA #$AD
0A13- 20 ED FD 0910  JSR COUT  Print dash
0A16- A2 01 0915  LDX #1 and a
0A18- 20 4A F9 0920  JSR PRBLANK  space.
0A1B- 20 8C F8 0925  JSR INSDS  Disassem
0A1E- 20 D3 F8 0930  JSR INSTD$  current
0A21- 20 53 F9 0935  JSR PCADJ  instr.
0A24- 85 3A 0940  STA PCL  & update
0A26- 84 3B 0945  STY PCL+1  prg cntr.
0A28- 60 0950  RTS
0955
0960 * -------------------------------Select a filter
0965
0970 PRINT.SCREEN.DATA
0975
0A29- AD 35 08 0980  LDA FLTNUM
0A2C- 0A 0985  ASL
0A2D- 0A 0990  ASL
0A2E- AA 0995  TAX
0A2F- A0 00 1000  LDY #0
0A31- BD 95 0C 1005 .1  LDA FLT.LOC,X
STA DCHR,Y
INX
INY
CPY #4
BCC .1

LDA #0
STA BUFFER.POINTER
INC CV
LDA CV
LDA CV
JSR PRINT.OLD.LINE
INC CV
LDA CV
LDY #39
LDA #SPACE
STA (BASE1),Y
DEY
BPL .3
JMP PROMPT

LDA #0
STA CV
JSR PRINT.OLD.LINE
INC CV
CMP #20 BNE .2 Last line?
BNE .2 No!
LDA CV
JSR FIND.BASE.ADDR
LDY #39
LDA #SPACE
STA (BASE1),Y
DEY
BPL .3
JMP PROMPT

* -------------------Print buffer data to screen

LDA #0
STA BUFFER.POINTER
INC CV
LDA CV
LDA CV
JSR PRINT.OLD.LINE
INC CV
LDA CV
LDY #39
LDA #SPACE
STA (BASE1),Y
DEY
BPL .3
JMP PROMPT

* -------------------Memory locations for text scrn

DA $400 Line 1
DA $480
DA $500
DA $580
DA $600
DA $680
DA $700
DA $780
DA $428
DA $4A8
DA $528
DA $5A8
DA $628
DA $6A8
DA $728
DA $7A8
DA $450
DA $4D0
DA $550
LOX #20
LOA FIRST.CHAR.POSN,X
CMP CRSVAL
BCC .2
BEQ .2
JMP .i
STX CV
STA BUFFER POINTER
RTS

ASL
TAX
LOA TEXT.SCREEN.BYTE,X
STA BASE1
CLC
ADC #27
STA BASE2
LOA TEXT.SCREEN.BYTE+1,X
STA BASE1+1
STA BASE2+1
RTS

.* ------------ Enter with line# in ACC.
.
FIND.BASE.ADDR

ASL
TAX
LDA TEXT.SCREEN.BYTE,X
STA BASE1
CLC
ADC #27
STA BASE2
LDA TEXT.SCREEN.BYTE+1,X
STA BASE1+1
STA BASE2+1
RTS

.* Convert CRSVAL to line#

FIND.CURRENT.LINE

LDX #20
DEX
LDA FIRST.CHAR.POSN,X
CMP CRSVAL
BCC .2
BEQ .2
JMP .1
STX CV
STA BUFFER POINTER
RTS

FIRST.CHAR.POSN

0AB8- 00 0D 1A
0ABB- 27 34 41
OABE- 4E
0ABF- 5B 68 75
OAC2- 82 8F 9C
OAC5- A9
OAC6- B6 C3 D0
OAC9- DD EA F7 1460

Book of Softkeys Vol. I 62
OACC- 20 A3 OA 1480 JSR FIND.CURRENT.LINE
1485
1490 PRINT.OLD.LINE
1495
OACF- A5 25 1500 LDA CV
OAD1- 20 8F 0A 1505 JSR FIND.BASE.ADDR
OAD4- A9 0D 1510 LDA #13
OAD6- 8D 6C 08 1515 STA HCOUNT
OAD9- A6 E4 1520 LDX BUFFER.POINTER
OADB- BD 00 09 1525 .2 LDA BUFFER,X
OADE- 48 1530 PHA
OADF- EC 34 08 1535 CPX CRSVAL
OAE2- D0 05 1540 BNE .3
OAE4- AE 6F 08 1545 LDX KEYFLG
OAE7- FO 03 1550 BZR .4
OAE9- 20 B1 0B 1555 .3 JSR FILTER
OAE C- A0 00 1560 .4 LDY #0
OAEE- 91 26 1565 STA (BASE2),Y
OAF0- E6 26 1570 INC BASE2
OAF2- 68 1575 PLA
OAF3- AE 6F 08 1580 LDX KEYFLG
OAF6- F0 44 1585 BZR .9
OAF8- 48 1590 PHA
OAF9- 4A 1595 LSR
OAF A- 4A 1600 LSR
OAFB- 4A 1605 LSR
OAF C- 4A 1610 LSR
OAF D- A6 E4 1615 LDX BUFFER.POINTER
OAF E- EC 34 08 1620 CPX CRSVAL
OBO2- D0 OB 1625 BNE .5
0B04- 09 30 1630 ORA #$30
0B06- C9 3A 1635 CMP #$3A
0B08- 90 0D 1640 BCC .6
0B0A- E9 39 1645 SBC #$39
0B0C- 4C 17 0B 1650 JMP .6
0B0F- 09 B0 1655 .5 ORA #$B0
0B11- C9 BA 1660 CMP #$BA
0B13- 90 02 1665 BCC .6
0B15- 69 06 1670 ADC #$06
0B17- 91 28 1675 .6 STA (BASE1),Y
0B19- E6 28 1680 INC BASE1
0B1B- 68 1685 PLA
0B1C- 29 0F 1690 AND #$0F
0B1E- A6 E4 1695 LDX BUFFER.POINTER
0B20- EC 34 08 1700 CPX CRSVAL
OB23- D0 0B 1705 BNE .7
OB25- 09 30 1710 ORA #$30
OB27- C9 3A 1715 CMP #$3A
OB29- 90 0D 1720 BCC .8
OB2B- E9 39 1725 SBC #$39
OB2D- 4C 3B 0B 1730 JMP .8
OB30- 09 B0 1735 .7 ORA #$B0
OB32- C9 BA 1740 CMP #$BA
OB34- 90 02 1745 BCC .8
OB36- 69 06 1750 ADC #$06
OB38- 91 28 1755 .8 STA (BASE1),Y
OB3A- E6 28 1760 INC BASE1
OB3C- E6 E4 1765 .9 INC BUFFER.POINTER
OB3E- A6 E4 1770 LDX BUFFER.POINTER
OB40- F0 08 1775 BZR .10
OB42- CE 6C 0B 1780 DEC HCOUNT
OB45- AD 6C 0B 1785 LDA HCOUNT
OB48- D0 91 1790 BNE .2
OB4A- AE 6F 0B 1795 .10 LDX KEYFLG
OB4D- F0 1B 1800 BZR .12
OB4F- A9 A0 1805 LDA #$A0
OB51- 91 28 1810 STA (BASE1),Y
OB53- A6 25 1815 LDX CV
OB55- E0 13 1820 CPX #19
OB57- D0 11 1825 BNE .12
OB59- A9 A0 1830 LDA #$SPACE
OB5B- 91 26 1835 .11 STA (BASE2),Y
OB5D- 91 28 1840 STA (BASE1),Y
OB5F- E6 28 1845 INC BASE1
OB61- 91 28 1850 STA (BASE1),Y
OB63- C8 1855 INY
OB64- C0 04 1860 CPY #4
OB66- D0 F3 1865 BNE .11
OB68- 91 28 1870 STA (BASE1),Y
OB6A- 60 1875 .12 RTS
1880
1885 * -----------------------------------
1890 1895 CALC.FREE.SECTORS
1900
OB6B- A9 00 1905 LDA #$00
OB6D- 85 E1 1910 STA NUM
OB6F- 85 E2 1915 STA NUM+1
OB71- A0 C8 1920 LDY #$C8
OB73- B9 F2 B3 1925 NXTBYTE LDA VTOC,Y
OB76- F0 0B 1930 NXTBIT BEQ .2
OB78- 0A 1935 .1 ASL
OB79- 90 FB 1940 BCC NXTBIT
LOA BYTE
STA LOC
LSR
LSR
LSR
LSR
LSR
LOA (CFLT),Y
TAX
AND #$FO
CLC
ADC LOC
STA LOC
TXA
AND #$OF
BNE .2
LOA LOC
RTS
INC NUM
BNE .1
INC NUM+1
BNE .1
DEY
BNE NXTBYTE
LOX #15
LOA FSTEXT-1,X
JSR COUT
OEX
BNE .3
LOX NUM
LOA NUM+1
JSR L1NPRT
LOA #RETURN
JSR COOT
RTS

OB7B- E6 E1 1945
OB7D- D0 F9 1950
BNE .1
OB7F- E6 E2 1955
INC NUM+1
OB81- D0 F5 1960
BNE .1
OB83- 88 1965 .2
DEY
OB84- D0 ED 1970
BNE NXTBYTE
OB86- A2 0F 1975
LDX #15
OB88- BD 9D OB 1980 .3
LDA FSTEXT-1,X
OB8B- 20 ED FD 1985
JSR COUT
OB8E- CA 1990
DEX
OB8F- D0 F7 1995
BNE .3
OB91- A6 E1 2000
LDX NUM
OB93- A5 E2 2005
LDA NUM+1
OB95- 20 24 ED 2010
JSR L1NPRT
OB98- A9 8D 2015
LDA #RETURN
OB9A- 20 ED FD 2020
JSR COUT
OB9D- 60 2025
RTS
OB9E- A0 BD AO
OBA1- C5 C5 D2
OBA4- C6 A0 D3
OBA7- D2 CF D4
OBA9- C3 C5 D3
OBAD- A0 2030 FSTEXT .AS -" = EERF SROTCES "
2035
2040 * -------------------------------Screen character filter
2045
OBAE- AD 2B 08 2050 FILTER0 LDA BYTE
OBB1- 85 E0 2055 FILTER STA LOC
OBB3- 4A 2060 LSR
OBB4- 4A 2065 LSR
OBB5- 4A 2070 LSR
OBB6- 4A 2075 LSR
OBB7- 4A 2080 LSR
OBB8- A8 2085 TAY
OBB9- B1 E9 2090 LDA (CFLT),Y
OBBB- AA 2095 TAX
OBBC- 29 F0 2100 AND #$FO
OBBE- 18 2105 CLC
OBBF- 65 E0 2110 ADC LOC
OBC1- 85 E0 2115 STA LOC
OBC3- 8A 2120 TXA
OBC4- 29 0F 2125 AND #$OF
OBC6- D0 03 2130 BNE .2
OBC8- A5 E0 2135 .1 LDA LOC
OBCA- 60 2140 RTS
2145
2150 * ----------------------------------Select function
2155

65 Book of Softkeys Vol. I
OBCB- C9 01  2160.2  CMP #1  Function 1?
OBCD- D0 09  2165   BNE .4
OBCF- B1 E7  2170   LDA (DCHR),Y
OBDF- C5 E0  2175   CMP LOC
OBDF- D0 F3  2180   BNE .1
OBDF- A9 A0  2185 .3  LDA #SPACE
OBDF- 60     2190   RTS
OBDF- C9 02  2195 .4  CMP #2  Function 2?
OBDF- D0 08  2200   BNE .5
OBDF- B1 E7  2205   LDA (DCHR),Y
OBDF- C5 E0  2210   CMP LOC
OBDF- F0 E6  2215   BEQ .1
OBDF- D0 F1  2220   BNE .3
OBDF- C9 03  2225 .5  CMP #3  Function 3?
OBDF- F0 ED  2230   BEQ .3
OBDF- 4C C8 0B  2235   JMP .1
        2240
        2245
        2250 * -------------------------------Filter parameter data
        2255
OBDF- 2260   CHG0   .BS 8
OBDF- 2265   CHG1   .BS 8
OBDF- C0 80 80
OBDF- 40 80 00
OC01- 00 E0  2270   CHG2   .HS C0808040800000E0
OC03- C1 81 81
OC06- 41 81 01
OC09- 01 E1  2275   CHG3   .HS C1818141810101E1
OC0B- C0 80 80
OC0E- 40 01 00
OC11- 00 E0  2280   CHG4   .HS C0808040010000E0
OC13- 02 02 02
OC16- 02 C0 00
OC19- 40 E0  2285   CHG5   .HS 02020202C00040E0
OC1B- 00 00 00
OC1F- 00 00 00
OC21- 00 00 2290   CHG6   .HS 0000000000000000
OC23- C0 80 80
OC26- 40 80 00
OC29- 00 E0  2295   CHG7   .HS C0808040800000E0
OC2B-    2300   CHG8   .BS 8
OC33-    2305   CHG9   .BS 8
        2310
        2315 * ----------------------------------------------
        2320
OC3B- 2325   DEL0   .BS 8
OC3A- 2330   DEL1   .BS 8
OC3B- 2335   DEL2   .BS 8

Book of Softkeys Vol. I  66
FILTER #0

LOA OLDTRK
STA TRACK
LOA OLDSCT
STA SECTOR
LOA #21
JSR FINO.BASE.ADDR

2375

2380 *----------------------------------FILTER STATUS I=ON
2385

OC8B- 00 2390 FSTAT .HS 00 FILTER #0
OC8C- 00 00 00
OC8F- 00 01 00

OC9- 00 00 00 2395 .HS 000000000100000000

OC95- 3B OC 2400
OC97- EB OB 2420 .DA CHG0
OC99- 43 OC 2425 .DA DEL1
OC9B- F3 OB 2430 .DA CHG1
OC9D- 4B OC 2435 .DA DEL2
OC9F- FB OB 2440 .DA CHG2
OCA1- 53 OC 2445 .DA DEL3
OCA3- 03 OC 2450 .DA CHG3
OCA5- 5B OC 2455 .DA DEL4
OCA7- 0B OC 2460 .DA CHG4
OCA9- 63 OC 2465 .DA DEL5
OCAB- 13 OC 2470 .DA CHG5
OCAD- 6B OC 2475 .DA DEL6
OCAF- 1B OC 2480 .DA CHG6
OCB1- 73 OC 2485 .DA DEL7
OCB3- 23 OC 2490 .DA CHG7
OCB5- 7B OC 2495 .DA DEL8
OCB7- 2B OC 2500 .DA CHG8
OCB9- 83 OC 2505 .DA DEL9
OCBB- 33 OC 2510 .DA CHG9

2515

2520 *----------------------------------Print screen prompts
2525

OCBD- AD 2C 08 2530 PROMPTO LDA ODLTRK
OCO- 8D 1A 08 2535 STA TRACK
OC3- AD 2D 08 2540 LDA OLDSCT
OC6- 8D 1B 08 2545 STA SECTOR
OC9- A9 15 2550 PROMPT LDA #21
OCB- 20 8F 0A 2555 JSR FIND.BASE.ADDR

67 Book of Softkeys Vol. I
OCE- A0 00 2560 LDY #0
OCD- B9 72 0D 2565 .1 LDA PROMPT1,Y
OCD3- 91 28 2570 STA (BASE1),Y
OCD5- C8 2575 INY
OCD6- C0 03 2580 CPY #3
OCD8- 90 F6 2585 BCC .1
OCDA- AD 17 08 2590 LDA SLOT
OCDD- 4A 2595 LSR
OCDE- 4A 2600 LSR
OCDF- 4A 2605 LSR
OCE0- 4A 2610 LSR
OCE1- 09 B0 2615 ORA #$80
OCE3- 91 28 2620 STA (BASE1),Y
OCE5- C8 2625 INY
OCE6- B9 72 0D 2630 .2 LDA PROMPT1,Y
OCE9- 91 28 2635 STA (BASE1),Y
OCEB- C8 2640 INY
OCEC- C0 08 2645 CPY #8
OCEE- 90 F6 2650 BCC .2
OCF0- AD 18 08 2655 LDA DRIVE
OCF3- 09 B0 2660 ORA #$80
OCF5- 91 28 2665 STA (BASE1),Y
OCF7- C8 2670 INY
OCF8- B9 72 0D 2675 .3 LDA PROMPT1,Y
OCFB- 91 28 2680 STA (BASE1),Y
OCFD- C8 2685 INY
OCFE- C0 0C 2690 CPY #12
OD00- 90 F6 2695 BCC .3
OD02- AD 1A 08 2700 LDA TRACK
OD05- 20 D1 10 2705 JSR PRINT.HEX.OR.DECIMAL
OD08- B9 72 0D 2710 .4 LDA PROMPT1,Y
OD0B- 91 28 2715 STA (BASE1),Y
OD0D- C8 2720 INY
OD0E- C0 12 2725 CPY #18
OD10- 90 F6 2730 BCC .4
OD12- AD 1B 08 2735 LDA SECTOR
OD15- 20 D1 10 2740 JSR PRINT.HEX.OR.DECIMAL
OD18- B9 72 0D 2745 .5 LDA PROMPT1,Y
OD1B- 91 28 2750 STA (BASE1),Y
OD1D- C8 2755 INY
OD1E- C0 18 2760 CPY #24
OD20- 90 F6 2765 BCC .5
OD22- AD 24 08 2770 LDA VOLUME
OD25- 20 D1 10 2775 JSR PRINT.HEX.OR.DECIMAL
OD28- B9 72 0D 2780 .6 LDA PROMPT1,Y
OD2B- 91 28 2785 STA (BASE1),Y
OD2D- C8 2790 INY
OD2E- C0 1E 2795 CPY #30

Book of Softkeys Vol. I 68
OD30- 90 F6 2800 BCC .6
OD32- AD 34 08 2805 LDA CRSVAL
OD35- 20 D1 10 2810 JSR PRINT.HEX.OR.DECIMAL
OD38- B9 72 0D 2815 .7 LDA PROMPT1,Y
OD3B- 91 28 2820 STA (BASE1),Y
OD3D- C8 2825 INY
OD3E- C0 23 2830 CPY #35
OD40- 90 F6 2835 BCC .7
OD42- AD 35 08 2840 LDA FLTNUM
OD45- 09 B0 2845 ORA #$B0
OD47- 91 28 2850 STA (BASE1),Y
OD49- C8 2855 INY
OD4A- B9 72 0D 2860 .8 LDA PROMPT1,Y
OD4D- 91 28 2865 STA (BASE1),Y
OD4F- C8 2870 INY
OD50- C0 25 2875 CPY #37
OD52- 90 F6 2880 BCC .8
OD54- AE 6E 08 2885 LDX EDIT.MODE.FLAG
OD57- BD 63 0D 2890 .9 LDA EDIT.MODE.TEXT,X
OD5A- 91 28 2895 STA (BASE1),Y
OD5C- C8 2900 INY
OD5D- E8 2905 INX
OD5E- C0 28 2910 CPY #40
OD60- 90 F5 2915 BCC .9
OD62- 60 2920 RTS

2925 * ----------------------------------------------
2930 EDIT.MODE.TEXT
2935

OD63- 08 05 18
OD66- 01 13 03
OD69- 09 0E 2940 .HS 080518011303090E
OD6B- 16 06 OC
OD6E- 13 0C 2F
OD71- 03 2945 .HS 16060C130C2F03
OD75- A0 A0 2965 .HS 130CBA0A0A0 ....SL
OD77- 04 12 BA
OD7A- A0 A0 2970 .HS 0412BA0A0A0 ....DR
OD7C- 14 BA A0
OD7F- A0 A0 A0 2975 .HS 14BA0A0A0A0 ..T
OD82- 13 BA A0
OD85- A0 A0 A0 2980 .HS 13BA0A0A0A0 ..S
OD88- 16 BA A0
OD8B- A0 A0 A0 2985 .HS 16BA0A0A0A0 ..V
OD8E- 02 BA A0
OD91- A0 A0 A0 2990 .HS 02BA0A0A0A0 ..B
OD94- 06 A0 A0
OD97- A0 A0 A0 2995 .HS 06A0A0A0A0A0 ..F
3000
3005 * -----------------------------------------------
3010
3015 SET.HEX.OR.DEC
3020
OD9A- A2 01 3025 LDX #1
OD9C- EC 2F 08 3030 CPX HEX.OR.DEC.FLG
OD9F- DO 01 3035 BNE .1
ODA1- CA 3040 DEX
ODA2- 8E 2F 08 3045 .1 STX HEX.OR.DEC.FLG
ODA5- 4C C9 0C 3050 JMP PROMPT
3055 * -----------------------------------------------
ODA8- A2 01 3060 SWT.DRV LDX #1
ODAA- EC 18 08 3065 CPX DRIVE
ODAD- DO 01 3070 BNE .1
ODAF- E8 3075 INX
ODB0- 8E 18 08 3080 .1 STX DRIVE
ODB3- 4C C9 0C 3085 JMP PROMPT
3090 * -----------------------------------------------
ODB6- A5 E1 3095 FSET LDA LOC+1
ODB8- 38 3100 SEC
ODB9- EC 80 3105 SBC #$BO
ODBB- BD 35 08 3110 STA FLTNUM
ODBE- 4C 29 0A 3115 JMP PRINT.SCREEN.DATA
3120 * -----------------------------------------------
ODC1- CE 1B 08 3125 DEC.SCT DEC SECTOR
ODC4- 10 13 3130 BPL IOJMP
ODC6- AE 70 08 3135 LDX MAXSCT
ODC9- CA 3140 DEX
ODCA- 8E 1B 08 3145 STX SECTOR
3150 * -----------------------------------------------
ODCD- CE 1A 08 3155 DEC.TRK DEC TRACK
ODDO- 10 07 3160 BPL IOJMP
ODD2- AE 71 08 3165 LDX MAXTRK
ODD5- CA 3170 DEX
ODD6- 8E 1A 08 3175 STX TRACK
ODD9- 20 90 08 3180 IOJMP JSR CALLIO
ODDC- 4C 29 0A 3185 JMP PRINT.SCREEN.DATA
3190 * -----------------------------------------------
ODDF- EE 1B 08 3195 INC.SCT INC SECTOR
ODE2- AE 1B 08 3200 LDX SECTOR
ODE5- EC 70 08 3205 CPX MAXSCT
ODE8- 90 EF 3210 BCC IOJMP
ODEA- A2 00 3215 LDX #0
ODEC- 8E 1B 08 3220 STX SECTOR

Book of Softkeys Vol. I 70
3225 * ----------------------------------------------
3230 INC.TRK INC TRACK
3235 LDX TRACK
3240 CPX MAXTRK
3245 BCC 10JMP
3250 LDX #0
3255 STX TRACK
3260 BEQ 10JMP ... ALWAYS
3265
3270 * ---------------------------------------------- CURSOR MOVEMENT ROUTINE 3275
3280 JSR FIND.CURRENT.LINE
3285 DEC CRSVAL
3290 JMP CRS1
3300 JSR FIND.CURRENT.LINE
3305 INC CRSVAL
3310 JMP CRS1
3320 JSR FIND.CURRENT.LINE
3325 LDA CRSVAL
3330 SEC
3335 SBC #13
3340 BCS .2
3345 CMP #$FC
3350 BCC .1
3355 SBC #14
3360 ADC #4
3365 STA CRSVAL
3370 JSR PRINT.OLD.LINE
3375 JSR PRINT.NEW.LINE
3380 JSR PRTCRS
3385 LDX EDFLAG
3390 BNE PARSE2
3395 RTS
3400
3405 * ----------------------------------------------
3410 JSR FIND.CURRENT.LINE
3415 LDA CRSVAL
3420 CLC
3425 ADC #13
3430 BCC .2
3435 CMP #4
3440 BCS .1
3445 ADC #14
3450 ADC #$FB
3455 STA CRSVAL
3460 JMP CRS1

71 Book of Softkeys Vol. I
3465
3470 SET.HEX.EDIT
3475
OE51- A9 00 3480 LDA #0
OE53- 8D 6E 08 3485 SETMODE STA EDIT.MODE.FLAG
OE56- 4C 9C 0C 3490 JMP PROMPT
3495
3500 SET.ASCII.EDIT
3505
OE59- A9 03 3510 LDA #3
OE5B- D0 F6 3515 BNE SETMODE ...Always
3520
3525
3530 *
3535 PARSE LDA TRACK
OE60- 8D 2C 08 3540 STA OLDTRK
OE63- AD 1B 08 3545 LDA SECTOR
OE66- 8D 2D 08 3550 STA OLDSCT
OE69- AE 72 08 3555 PARSE2 LDX SPECIAL.FUNCTION
OE6C- F0 0C 3560 BEQ .2
OE6E- CA 3565 OEX
OE6F- 8E 72 08 3570 STX SPECIAL.FUNCTION
OE72- AD 00 C0 3575 .1 LDA KEY
OE75- 10 FB 3580 BPL .1
OE77- 20 29 OA 3585 JSR PRINT.SCREEN.DATA
OE7A- 20 83 10 3590 .2 JSR INKEY
OE7D- A2 FD 3595 LDX #$FD
OE7F- E8 3599 .3 INX
OE80- E8 359D INX
OE81- E8 3600 INX
OE82- BD 95 0E 3615 LDA VALID.CMND.TABLE,X
OE85- F0 0D 3620 BEQ .4
OE87- C5 E0 3625 CMP LOC
OE89- D0 F4 3630 BNE .3
OE8B- E8 3634 INX
OE8C- BD 96 0E 3640 LDA VALID.CMND.TABLE+1,X
OE8F- 4B 3645 PHA
OE90- BD 95 0E 3650 LDA VALID.CMND.TABLE,X
OE93- 48 3655 PHA
OE94- 60 3660 .4 RTS Bad Command
3665
3670 *
3680 VALID.CMND.TABLE
OE95- C9 3685 .HS C9 I
OE96- 12 0E 3690 .DA UP-1
OE98- CA 3695 .HS CA J
OE99- 00 0E 3700 .DA LEFT-1
<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OE9B</td>
<td>CB 3705</td>
<td>.HS CB K</td>
</tr>
<tr>
<td>OE9C</td>
<td>09 3710</td>
<td>.DA RIGHT-1</td>
</tr>
<tr>
<td>OE9D</td>
<td>CD 3715</td>
<td>.HS CD M</td>
</tr>
<tr>
<td>OE9F</td>
<td>37 3720</td>
<td>.DA DOWN-1</td>
</tr>
<tr>
<td>OEA1</td>
<td>88 3725</td>
<td>.HS 88 &lt;-</td>
</tr>
<tr>
<td>OEA2</td>
<td>00 3730</td>
<td>.DA LEFT-1</td>
</tr>
<tr>
<td>OEA4</td>
<td>95 3735</td>
<td>.HS 95 -&gt;</td>
</tr>
<tr>
<td>OEA5</td>
<td>09 3740</td>
<td>.DA RIGHT-1</td>
</tr>
<tr>
<td>OEA7</td>
<td>AC 3745</td>
<td>.HS AC</td>
</tr>
<tr>
<td>OEA8</td>
<td>CC 0D 3750</td>
<td>.DA DEC.TRK-1</td>
</tr>
<tr>
<td>OEA9</td>
<td>AE 3755</td>
<td>.HS AE</td>
</tr>
<tr>
<td>OEB1</td>
<td>EE 0D 3760</td>
<td>.DA INC.TRK-1</td>
</tr>
<tr>
<td>OED1</td>
<td>B1 3765</td>
<td>.HS B1 1</td>
</tr>
<tr>
<td>OEB3</td>
<td>B2 3770</td>
<td>.DA FSET-1</td>
</tr>
<tr>
<td>OEB6</td>
<td>B3 3775</td>
<td>.HS B2 2</td>
</tr>
<tr>
<td>OEB7</td>
<td>B4 3780</td>
<td>.DA FSET-1</td>
</tr>
<tr>
<td>OEB8</td>
<td>B5 3785</td>
<td>.HS B3 3</td>
</tr>
<tr>
<td>OEB9</td>
<td>B5 3790</td>
<td>.DA FSET-1</td>
</tr>
<tr>
<td>OEBB</td>
<td>B5 3795</td>
<td>.HS B4 4</td>
</tr>
<tr>
<td>OEB4</td>
<td>B6 3800</td>
<td>.DA FSET-1</td>
</tr>
<tr>
<td>OEB5</td>
<td>B6 3805</td>
<td>.HS B5 5</td>
</tr>
<tr>
<td>OEB7</td>
<td>B5 3810</td>
<td>.DA FSET-1</td>
</tr>
<tr>
<td>OEB8</td>
<td>B6 3815</td>
<td>.HS B6 6</td>
</tr>
<tr>
<td>OEB9</td>
<td>B5 3820</td>
<td>.DA FSET-1</td>
</tr>
<tr>
<td>OEBF</td>
<td>B7 3825</td>
<td>.HS B7 7</td>
</tr>
<tr>
<td>OEC0</td>
<td>B5 3830</td>
<td>.DA FSET-1</td>
</tr>
<tr>
<td>OEC1</td>
<td>B8 3835</td>
<td>.HS B8 8</td>
</tr>
<tr>
<td>OEC3</td>
<td>B5 3840</td>
<td>.DA FSET-1</td>
</tr>
<tr>
<td>OEC5</td>
<td>B9 3845</td>
<td>.HS B9 9</td>
</tr>
<tr>
<td>OEC6</td>
<td>B5 3850</td>
<td>.DA FSET-1</td>
</tr>
<tr>
<td>OEC8</td>
<td>BC 3855</td>
<td>.HS BC &lt;</td>
</tr>
<tr>
<td>OEC9</td>
<td>CC 0D 3860</td>
<td>.DA DEC.TRK-1</td>
</tr>
<tr>
<td>OECB</td>
<td>BE 3865</td>
<td>.HS BE &gt;</td>
</tr>
<tr>
<td>OECQ</td>
<td>EE 0D 3870</td>
<td>.DA INC.TRK-1</td>
</tr>
<tr>
<td>OECF</td>
<td>C1 3875</td>
<td>.HS C1 A</td>
</tr>
<tr>
<td>OED1</td>
<td>0C 3880</td>
<td>.DA SET.ASCII.EDIT-1</td>
</tr>
<tr>
<td>OED2</td>
<td>C4 3885</td>
<td>.HS C4 D</td>
</tr>
<tr>
<td>OED3</td>
<td>A7 0D 3890</td>
<td>.DA SM1.DRV-1</td>
</tr>
<tr>
<td>OED4</td>
<td>C8 3895</td>
<td>.HS C8 H</td>
</tr>
<tr>
<td>OED5</td>
<td>05 0E 3900</td>
<td>.DA SET.HEX.EDIT-1</td>
</tr>
<tr>
<td>OED7</td>
<td>CC 3905</td>
<td>.HS CC L</td>
</tr>
<tr>
<td>OED8</td>
<td>CE 0D 3910</td>
<td>.DA DEC.SCT-1</td>
</tr>
<tr>
<td>OEDA</td>
<td>C0 3915</td>
<td>.HS CE N</td>
</tr>
<tr>
<td>OEB9</td>
<td>DE 0D 3920</td>
<td>.DA INC.SCT-1</td>
</tr>
<tr>
<td>OEDD</td>
<td>D5 3925</td>
<td>.HS D5 U</td>
</tr>
<tr>
<td>OEDN</td>
<td>DE 99 3930</td>
<td>.DA SET.HEX.OR.DEC-1</td>
</tr>
<tr>
<td>OEE0</td>
<td>B0 3935</td>
<td>.HS B0 0</td>
</tr>
<tr>
<td>OEE1</td>
<td>3F 11 3940</td>
<td>.DA FILES-1</td>
</tr>
</tbody>
</table>
**Book of Softkeys Vol. I** 74
LOX ON.OFF
STX FLTNUM
LDX #1
STX EDFLG EDIT OFF
LDA EDIT. MODE. FLAG
BZR .4
LDA #3
STA EDIT. MADE. FLAG
RTS

LOX #$FF
STX FIRST
INX
STX EDFLG 0 = EDIT ON
LDX FLTNUM
STX ON.OFF
LDA FSTAT,X
BNE .1
STA FLTNUM
JSR PRINT. SCREEN. DATA
JSR PRINT. NEW. LINE
JSR PRTCRS
JSR INKEY
CMP #ESCAPE
BNE .5
LOX CRSVAL
LDA BUFFER1,X
STA BUFFER,X
INX
BNE .11
JMP PRINT. SCREEN. DATA
CMP #CTRL.A
BNE .14
LOX #$FE
DEC CRSVAL
LOA BUFFER,X
STA BUFFER1,X
DEX
CPX CRSVAL
BNE .13
INC CRSVAL
JMP PRINT. SCREEN. DATA
LOX #1
STX EDFLG
RTS

LDA BUFFER1,X
STA BUFFER,X
INX
BNE .11
JMP PRINT. SCREEN. DATA
CMP #CTRL.A
BNE .14
LOX #$FE
DEC CRSVAL
LOA BUFFER,X
STA BUFFER1,X
DEX
CPX CRSVAL
BNE .13
INC CRSVAL
JMP PRINT. SCREEN. DATA
LOX #1
STX EDFLG
RTS

LDA BUFFER1,X
STA BUFFER,X
INX
BNE .11
JMP PRINT. SCREEN. DATA
CMP #CTRL.A
BNE .14
LOX #$FE
DEC CRSVAL
LOA BUFFER,X
STA BUFFER1,X
DEX
CPX CRSVAL
BNE .13
INC CRSVAL
JMP PRINT. SCREEN. DATA
LOX #1
STX EDFLG
RTS

LDA BUFFER1,X
STA BUFFER,X
INX
BNE .11
JMP PRINT. SCREEN. DATA
CMP #CTRL.A
BNE .14
LOX #$FE
DEC CRSVAL
LOA BUFFER,X
STA BUFFER1,X
DEX
CPX CRSVAL
BNE .13
INC CRSVAL
JMP PRINT. SCREEN. DATA
LOX #1
STX EDFLG
RTS
4425
4430  * CHECK FOR HEX OR ASCII EDIT
4435
4440 .5
4445  BNE .8
4450
4455  * HEX EDIT ROUTINE
4460
4465  CMP #$SPACE
4470  BCS .6
4475  JSR CTRLMV
4480  LDX #$FF
4485  STX FIRST
4490  INX
4495  STX EDFLG
4500  JMP .2
4505 .6
4510  CMP #$16
4515  BCS .3
4520  INC FIRST
4525  BNE .7
4530  JSR CKHEX
4535  STA BUFFER,X
4540  JMP .2
4545 .7
4550  LDX CRSVAL
4555  STA BUFFER,X
4560  ASL
4565  ASL
4570  ASL
4575  ASL
4580  ORA LOC
4585  STA BUFFER,X
4590  JSR FIND.CURRENT.LINE
4595  INC CRSVAL
4600  JSR PRINT.OLD.LINE
4605  LDA #$FF
4610  STA FIRST
4615  JMP .2
4620  4625  * ASCII EDIT ROUTINE
4630
4635 .8
4640  BCS .10
4645  LDX USE.CTRL.CHARS
4650  BZR .14
4655  JSR CTRLMV
4660  LDX #$0
1018- EC 6B 08 4665  CPX EDFLG
101B- D0 03 4670  BNE .9
101D- 4C 98 0F 4675  JMP .2
1020- 8E 6B 08 4680 .9  STX EDFLG
1023- 4C 42 10 4685  JMP .14
1026- AE 69 08 4690 .10  LDX OFFSET
1029- E0 20 4695  CPX #$20  LOWER CASE?
102B- D0 0A 4700  BNE .11
102D- C9 C1 4705  CMP #$C1
102F- 90 11 4710  BCC .14  < "A"
1031- C9 DB 4715  CMP #$DB
1033- B0 0D 4720  BCS .14  > or = "["
1035- 90 08 4725  BCC .13  ...always
1037- C9 C0 4730 .11  CMP #$C0
1039- B0 03 4735  BCS .12  > = "@"
103B- 6D 69 08 4740  ADC OFFSET
103E- 18 4745 .12  CLC
103F- 6D 69 08 4750 .13  ADC OFFSET
1042- AE 34 08 4755 .14  LDX CRSVAL
1045- 9D 00 09 4760  STA BUFFER,X
1048- 20 A3 OA 4765  JSR FND.CURRENT.LINE
104B- EE 34 08 4770  INC CRSVAL
104E- 20 CF OA 4775  JSR PRINT.OLD.LINE
1051- 4C 98 0F 4780  JMP .2
1054- C9 B0 4800 CKHEX  CMP #$B0  "0"
1056- 90 10 4805  BCC .3  < "0"
1058- C9 C7 4810  CMP #$C7
105A- B0 OC 4815  BCS .3  > "F"
105C- C9 BA 4820  CMP #$BA
105E- 90 06 4825  BCC .2  < "10"
1060- C9 C1 4830  CMP #$C1
1062- 90 04 4835  BCC .3  >= "A"
1064- E9 07 4840  SBC #7
1066- 29 0F 4845 .2  AND #$0F
1068- 60 4850 .3  RTS
1069- AE 34 08 4870 NOPRESS  LDX CRSVAL
106C- BD 00 09 4875  LDA BUFFER,X
106F- 48 4880  PHA
1070- A9 20 4885  LDA #$20
1072- 20 9A 10 4890  JSR WAIT.FOR.KEY
1075- A9 A0 4895  LDA #SPACE
1077- 20 9A 10 4900  JSR WAIT.FOR.KEY

4785
4790 * ---------------------------------------------------------------
4795
4800 CKHEX  CMP #$B0  "0"
4805  BCC .3  < "0"
4810  CMP #$C7
4815  BCS .3  > "F"
4820  CMP #$BA
4825  BCC .2  < "10"
4830  CMP #$C1
4835  BCC .3  >= "A"
4840  SBC #7
4845 .2  AND #$0F
4850 .3  RTS
4855
4860 * ---------------------------Flashing cursor routine
4865
1069- AE 34 08 4870 NOPRESS  LDX CRSVAL
106C- BD 00 09 4875  LDA BUFFER,X
106F- 48 4880  PHA
1070- A9 20 4885  LDA #$20
1072- 20 9A 10 4890  JSR WAIT.FOR.KEY
1075- A9 A0 4895  LDA #SPACE
1077- 20 9A 10 4900  JSR WAIT.FOR.KEY
107A-  A2 01  4905  LDX #1
107C-  8E 6F 08  4910  STX KEYFLG
107F-  68  4915  PLA
1080-  20  9A 10  4920  JSR WAIT.FOR.KEY
1083-  A2 00  4925  INKEY  LDX #0
1085-  8E 6F 08  4930  STX KEYFLG
1088-  AD 00  CO  4935  OUTKEY  LDA KEY
108B-  10 DC  4940  BPL NOPRESS
108D-  8D 10 CO  4945  STA STROBE
1090-  85 E0  4950  STA LOC
1092-  85 E1  4955  STA LOC+1
1094-  A2 01  4960  LDX #1
1096-  8E 6F 08  4965  STX KEYFLG
1099-  60  4970  RTS

4975
4980  * ----------------------------------------------4985
4985 4990  WAIT.FOR.KEY  4995

109A-  AE 34 08  5000  LDX CRSVAL
109D-  9D 00 09  5005  STA BUFFER,X
10A0-  20 CC 0A  5010  JSR PRINT.NEW.LINE
10A3-  A9 3C  5015  LDA #60
10A5-  AA  5020 .1  TAX
10A6-  AC 00 CO  5025 .2  LDY KEY
10A9-  30 07  5030  BMI .3
10AB-  CA  5035  DEX
10AC-  D0 F8  5040  BNE .2
10AE-  E9 01  5045  SBC #1
10B0-  D0 F3  5050  BNE .1
10B2-  60  5055 .3  RTS
5060
5065  * ----------------------------------------------5070
5070

10B3-  A9 15  5075  PRTCRS  LDA #21
10B5-  20 8F 0A  5080  JSR FIND.BASE.ADDR
10B8-  A0 1E  5085  LDY #30
10BA-  AD 34 08  5090  LDA CRSVAL
10BD-  20 D1 10  5095  JSR PRINT.HEX.OR.DECIMAL
10C0-  AE 6D 08  5100  LDX SPACES
10C3-  F0 08  5105  BEQ .7
10C5-  A9 A0  5110 .6  LDA #SPACE
10C7-  91 28  5115  STA (BASE1),Y
10C9-  C8  5120  INY
10CA-  CA  5125  DEX
10CB-  D0 F8  5130  BNE .6
10CD-  60  5135 .7  RTS
5140
5145 * -----------------------------------------------
5150
5155 PRINT.HEX.DEC
5160
10CE- AD 2B 08 5165 LDA BYTE
5170
5175 PRINT.HEX.OR.DECIMAL
5180
10D1- 48 5185 PHA
10D2- AE 2F 08 5190 LDX HEX.OR.DEC.FLG
10D5- D0 2A 5195 BNE PRINT.DECIMAL
10D7- 8E 6D 08 5200 STX SPACES
5205
5210 * -----------------------------------------------
5215
5220 PRINT.HEX.BYTE
5225
10DA- A9 A4 5230 LDA #$A4
10DC- 91 28 5235 STA (BASE1),Y
10DE- C8 5240 INY
10DF- 68 5245 PLA
10E0- 48 5250 PHA
10E1- 4A 5255 LSR
10E2- 4A 5260 LSR
10E3- 4A 5265 LSR
10E4- 4A 5270 LSR
10E5- 09 B0 5275 ORA #$B0
10E7- C9 BA 5280 CMP #$BA
10E9- 90 02 5285 BCC .1
10EB- 69 06 5290 ADC #$06
10ED- 91 28 5295 .1 STA (BASE1),Y
10EF- C8 5300 INY
10F0- 68 5305 PLA
10F1- 29 0F 5310 AND #$0F
10F3- 09 B0 5315 ORA #$B0
10F5- C9 BA 5320 CMP #$BA
10F7- 90 02 5325 BCC .2
10F9- 69 06 5330 ADC #$06
10FB- 91 28 5335 .2 STA (BASE1),Y
10FD- C8 5340 INY
10FE- 84 24 5345 STY CH
1100- 60 5350 RTS
5355 * -----------------------------------------------
5360
5365 PRINT.DECIMAL
5370
1101- A2 02 5375 LDX #$2
1103- 8E 6D 08 5380 STX SPACES
1106- A2 B0 5385 LDX #$BO
1108- 68 5390 PLA
1109- C9 64 5395 CMP #100
110B- 90 12 5400 BCC .2
110D- E8 5405 .1 INX
110E- E9 64 5410 SBC #100
1110- C9 64 5415 CMP #100
1112- B0 F9 5420 BCS .1
1114- CE 6D 08 5425 DEC SPACES
1117- 48 5430 PHA
1118- 8A 5435 TXA
1119- 91 28 5440 STA (BASE1), Y
111B- C8 5445 INY
111C- A2 B0 5450 LDX #$BO
111E- 68 5455 PLA
111F- C9 0A 5460 .2 CMP #10
1121- 90 0A 5465 BCC .4
1123- E8 5470 .3 INX
1124- E9 0A 5475 SBC #10
1126- C9 0A 5480 CMP #10
1128- B0 F9 5485 BCS .3
112A- CE 6D 08 5490 DEC SPACES
112D- 48 5495 .4 PHA
112E- AD 6D 08 5500 LDA SPACES
1131- C9 02 5505 CMP #2
1133- F0 04 5510 BEQ .5
1135- 8A 5515 TXA
1136- 91 28 5520 STA (BASE1), Y
1138- C8 5525 INY
1139- 68 5530 .5 PLA
113A- 09 B0 5535 ORA #$BO
113C- 91 28 5540 STA (BASE1), Y
113E- C8 5545 INY
113F- 60 5550 RTS

5555

5560 *

5565

1140- A2 15 5570 FILES LDX #21
1142- 86 23 5575 STX WNDBTM
1144- A2 00 5580 LDX #0
1146- 8E 6D 08 5585 STX SPACES
1149- E8 5590 INX
114A- 8E 72 08 5595 STX SPECIAL_FUNCTION
114D- 20 58 FC 5600 JSR HOME
1150- E8 5605 INX
1151- 20 4A F9 5610 JSR PRBLANK
1154- AD 01 09 5615 LDA BUFFER+1
1157- 20 06 12 5620 JSR HEX2
115A- AD 02 09 5625  LDA BUFFER+2
115D-  20 06 12 5630  JSR HEX2
1160-  20 62 FC 5635  JSR CR.LF
1163-  20 62 FC 5640  JSR CR.LF
1166- A2 0B  5645  LDX #$0B
1168- A0 02  5650 .1  LDY #2
116A-  20 62 FC 5655  JSR CR.LF
116D-  20 09 12 5660  JSR SPCOUT
1170- BD 00 09 5665 .2  LDA BUFFER,X
1173-  20 06 12 5670  JSR HEX2
1176- E8  5675  INX
1177-  8B  5680  DEY
1178- DD F6  5685  BNE .2
117A- BD 00 09 5690  LDA BUFFER,X
117D- E8  5695  INX
117E-  2A  5700  ROL
117F-  4B  5705  PHA
1180-  90 08  5710  BCC .3
1182- A9 AA  5715  LDA #STAR
1184-  20 ED FD 5720  JSR COUNT
1187-  4C BD  11 5725  JMP .4
118A-  20 09 12 5730 .3  JSR SPCOUT
118D- A0 00  5735 .4  LDY #0
118F-  68  5740  PLA
1190-  4B  5745  LSR
1191- F0 04  5750  BEQ .6
1193- C8  5755 .5  INY
1194-  4B  5760  LSR
1195-  90 FC  5765  BCC .5
1197- B9 D2  11 5770 .6  LDA TYPE,Y
119A-  20 ED FD  5775  JSR COUNT
119D-  20 09 12  5780  JSR SPCOUT
11A0- A0 1E  5785  LDY #30
11A2-  8C 6C  08  5790  STY HCOUNT
11A5- BD 00 09  5795 .7  LDA BUFFER,X
11A8-  85 E0  5800  STA LOC
11AA-  4B  5805  LSR
11AB-  4B  5810  LSR
11AC-  4B  5815  LSR
11AD-  4B  5820  LSR
11AE-  4B  5825  LSR
11AF- A8  5830  TAY
11B0-  81 Book of Softkeys Vol. I
11B2-  29 F0  5840  AND #$FO
11B4-  1B  5845  CLC
11B5-  65 E0  5850  ADC LOC
11B7-  65 E0  5855  CMP #CTRL.AT change
11B9-  30 06  5860  BMI .8 control
11BD- 10 02 5870 BPL .8 to a
11BF- A9 AE 5875 LDA #PERIOD period.
11C1- 20 ED FD 5880 .8 JSR COUT
11C4- E8 5885 INX
11C5- CE 6C 08 5890 DEC HCOUNT
11C8- D0 DB 5895 BNE .7
11CA- E8 5900 INX
11CB- E8 5905 INX
11CC- D0 9A 5910 BNE .1
11CE- CA 5915 DEX
11CF- 86 E4 5920 STX BUFFER POINTER
11D1- 60 5925 RTS
11D2- D4 C9 C1
11D5- C2 D3 D2
11D8- C1 C2 5930 TYPE .AS -"TIABSRAB"
5935
5940 * --------------------------Filter used by BASIC
5945
11DA- AD 2B 08 5950 ASCPRINT LDA BYTE
11DD- C9 FF 5955 CMP #$FF
11DF- D0 05 5960 BNE .1
11E1- A9 A0 5965 LDA #$A0
11E3- 80 2B 08 5970 STA BYTE
11E6- 4A 5975 .1 LSR
11E7- 4A 5980 LSR
11E8- 4A 5985 LSR
11E9- 4A 5990 LSR
11EA- 4A 5995 LSR
11EB- A8 6000 TAY
11EC- B9 F6 11 6005 LDA ASCFD,Y
11EF- 18 6010 CLC
11FO- 6D 2B 08 6015 ADC BYTE
11F3- 4C ED FD 6020 JMP COUT
11F6- C0 80 80
11F9- 40 40 00
11FC- 00 C0 6025 ASCFD .HS C0808040400000C0
6030
11FE- A9 A4 6035 HEX0 LDA #$A4 "$"
1200- 20 ED FD 6040 JSR COUT
1203- AD 2B 08 6045 HEXPRINT LDA BYTE
1206- 20 DA FD 6050 HEX2 JSR PRHEX
1209- A9 A0 6055 SPCOUT LDA #SPACE
120B- 4C ED FD 6060 JMP COUT
6065
6070 * --------------------------Print Hex or Decimal
6075
120E- AE 2F 08 6080 HXBYTE LDX HEX.OR.DEC.FLG

Book of Softkeys Vol. I 82
BEQ HEXO  O = HEX
LDX BYTE
LDA #0
JSR LINPRT
JMP SPCOUT

* ----------------------------------------------

.HS 00
.HS 0000

* ----------------------------------------------

Book of Softkeys Vol. I
DiskView

By Charles Haight

This program is called DiskView. DiskView is a mini "nibbler." It will read the raw nibbilized data from a disk without regard to disk format.

This means data can be viewed on a nonstandard format disk (copy-protected) as easily as from a normal DOS formatted disk. With DiskView, a nonstandard disk can be examined to see what was changed. Often these changes are minor and a similar change can be made to your DOS. This would allow use of DiskEdit to read that disk.

To understand these changes lets examine the data pattern on a normal DOS 16 disk.

DOS formats a track by first writing a unique byte called a "sync byte." This byte (normally $FF) allows the Disk II hardware to synchronize with the data on the disk. DOS then writes an address field, some more sync bytes and the data field. At this time the data field is full of $00s. DOS goes on to write sixteen sets of address and data fields on each track. These sets of address and data fields are called sectors.

The following is a normal address field for 3.3 DOS:

D5AA96FFFFFFEAABBAAEAFBEFEAEB

It can be broken down into:

Start of address  D5 AA 96
Volume number    FF FE
Track            AA BB
Sector           AE AA
Checksum         FB EF
End of address   DE AA EB

The volume, track, sector and checksum are in a 4+4 coded format. This means that 4 bits in each byte are actual data. The first byte is rotated left and logically ANDed with the second byte to recover the data.

The data field consists of:

Start of data  D5 AA AD
Encoded data   (341 bytes)
Checksum       (1 byte)
End of data    DE AA EB

The data field is encoded in a 2+6 format. Six bits of each byte are valid data.

The basic structure of 3.2 DOS is similar to 3.3 DOS with these notable exceptions:

1. When initializing a disk, DOS 3.2 does not write a blank
data sector. Instead it just writes enough $FFs to fill the space a data sector would use. Trying to read a track/sector that has never been written to will always generate I/O errors.

2. The data is encoded in a 3+5 format which requires 410 bytes to encode 256 data bytes. This is one reason why there are only 13 sectors.

About the program

The format of DiskView is similar to DiskEdit. A full screen of hexadecimal bytes is displayed with the status prompts at the bottom of the screen. The buffer extends from $2000 to $4000 hex which is large enough to ensure reading in an entire track. The slot, drive and track are selectable. Half-tracks can be accessed by appending a "\textasciitilde .5" to the track number. The commands are:

\begin{itemize}
  \item D - change the drive
  \item L - read last track (steps by half tracks)
  \item N - read next track (steps by half tracks)
  \item P - print screen contents
  \item R - read the current track
  \item S - change the slot
  \item T - select a track or half track
  \item X - exit to basic
  \item \textasciitilde - increment buffer
  \item \textasciitilde \textasciitilde \textasciitilde - decrement buffer
\end{itemize}

Type in the program and save it to disk. Be especially careful with the data statements. When those values are poked into memory they become a machine language subroutine that is the heart of the program. Run the program. When the COMMAND prompt flashes, press the R key. The screen will fill with hex bytes that show the data stored on the disk.

\textbf{CAUTION: Utility Nibbler is DOS dependent. It calls directly into DOS to step the drive motor. DOS 3.3 and 48K of memory are needed. This program can be used to read 13 or 16-sector disks or any other Apple disk, but it will only run on a 48K Apple II (\+)
with 3.3 DOS.}

The program

\begin{verbatim}
10 TEXT : HOME : IN# 0 : PR# 0 : LOMEM: 16384 : POKE 1144,90: GOTO 90
20 KY% = PEEK (-16384) : IF KY% < 128 THEN 20
30 POKE -16368,0 : RETURN
40 FOR X = 1 TO 40: PRINT "-" ; : NEXT : RETURN
50 GOSUB 60: POKE 781,0 : POKE 1144,90: POKE TR%,0 : CALL 10%: POKE 781,255:
  POKE TR%,TK%: CALL 10%: RETURN
60 VTAB 23: HTAB 2: INVERSE : PRINT "SLOT" ; : HTAB 10: PRINT "DRIVE" ; : HTAB
  19: PRINT "TRACK" ; : NORMAL
\end{verbatim}
70 VTAB 23: HTAB 7: PRINT PEEK (S1%) / 16;: HTAB 16: PRINT PEEK (DR%) – PEEK (S1%);: HTAB 25: PRINT "*****B$B$ B$B$ PEEK (TR%) / 2
80 RETURN
90 GOSUB 540
100 IN% = PEEK (CT%): VTAB 21: HTAB 32: PRINT "PAGE***IN% – 31: GOSUB 60:
110 IF KY% = 210 THEN GOSUB 480
120 IF KY% = 211 THEN GOSUB 390
130 IF KY% = 216 THEN GOSUB 410
140 IF KY% = 212 THEN GOSUB 420
150 IF KY% = 199 THEN GOSUB 270
160 IF KY% = 196 THEN GOSUB 230
170 IF KY% = 208 THEN GOSUB 290
180 IF KY% = 136 THEN GOSUB 250
190 IF KY% = 149 THEN GOSUB 370
200 IF KY% = 204 THEN GOSUB 490
210 IF KY% = 206 THEN GOSUB 510
220 GOTO 100
230 VTAB 23: HTAB 30: INVERSE: PRINT G$"SET DRIVE":; HTAB 10: FLASH:
PRINT "DRIVE":; NORMAL: HTAB 16: PRINT ">" CHR$ (8):; GET A$: DR = VAL (A$): IF DR < 1 OR DR > 2 THEN 230
240 POKE DR%, PEEK (S1%) + DR: GOTO 50
250 IN% = IN% – 1: IF IN% < 32 THEN IN% = 32
260 POKE CT%, IN%: CALL MV%: RETURN
270 PRINT G$: IF G$ = CHR$ (7) THEN G$ = ": RETURN
280 IF G$ = "" THEN G$ = CHR$ (7): RETURN
290 VTAB 23: HTAB 30: FLASH: PRINT G$">PRINTER<":; NORMAL
300 PR# 1
310 BUFFER% = PEEK (CT%) * 256
320 PRINT: PRINT "TRACK" TK%
330 FOR X = 0 TO 255 STEP 13: FOR Y = 0 TO 12: POKE NW%, PEEK (BUFFER% + X + Y) : CALL HX%: PR INT "A" ; : NEXT Y: PR INT
340 IF PEEK (-16384) = 155 THEN 360
350 NEXT X
360 PR# 0: POKE – 16368,0: RETURN
370 IN% = IN% + 1: IF IN% > 63 THEN IN% = 63
380 POKE CT%, IN%: CALL MV%: RETURN
"SLOT":; NORMAL: HTAB 7: PRINT ">" CHR$ (8):; GET A$: KY% = VAL (A$): IF KY% < 1 OR KY% > 7 THEN 390
400 POKE S1%, KY% * 16: POKE S2%, KY% * 16: GOTO 240
410 TEXT: HOME: POKE 33,33: CALL 1002: END
420 C$ = "": VTAB 23: HTAB 30: INVERSE: PRINT "SET TRACK":; HTAB 19: FLASH:
PRINT G$"TRACK":; NORMAL: PRINT "*****CHR$ (8) CHR$ (8) CHR$ (8) ;; GET A$: C$ = C$ + A$: PRINT A$;; GET A$: C$ = C$ + A$: IF A$ = CHR$ (13) THEN 460
430 PRINT A$;
440 GET A$: C$ = C$ + A$: PRINT A$;
450 IF A$ = "." THEN GET A$: C$ = C$ + A$: PRINT A$;
460 KY = VAL (C$): IF KY < 0 OR KY > 35 THEN 420
470 TK% = KY * 2
480 POKE CT%, 32: VTAB 23: HTAB 30: FLASH: PRINT ">>>READ<<<"G$;: NORMAL:
PRINT "***": POKE TR%, TK%: GOSUB 70: CALL 10%: GOTO 60
490 TK% = TK% - 1: IF TK% < 0 THEN TK% = 71
500 GOTO 480
510 TK% = TK% + 1: IF TK% > 71 THEN TK% = 0
520 GOTO 480
530 STOP
540 FOR X = 768 TO 894: READ X%; POKE X, X%; NEXT X
560 DATA 189,136,192,169,1,133,37,32,34,252,169,0,133,36,133,30,169,13,133,31,162,1,32,74,249,166,30,189,0,32,32,218,253,162,1,32,74,249,230,30,240,7,198,31,208,235,76,75,3,32,156,252,230,37,32,34,252,169,22
570 DATA 133,34,96,169,172,32,218,253,96
580 S1% = 774:S2% = 805:DR% = 769:TR% = 789:MV% = 830:CT% = 856:B$ = CHR$(8):G$ = CHR$(7):10% = 768:NM% = 890:HX% = 889:DR = 1
600 VTAB 22: GOSUB 40: GOTO 60
### 4 plus 4 Conversion Chart

<table>
<thead>
<tr>
<th>AA + AA</th>
<th>AA + AB</th>
<th>AA + AE</th>
<th>AA + AF</th>
<th>AA + AA = 00</th>
<th>AA + AB = 01</th>
<th>AA + AE = 04</th>
<th>AA + AF = 05</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA + BA</td>
<td>AA + BB</td>
<td>AA + EB</td>
<td>AA + FB</td>
<td>AA + AA = 10</td>
<td>AA + AB = 11</td>
<td>AA + AE = 14</td>
<td>AA + AF = 15</td>
</tr>
<tr>
<td>AA + EA</td>
<td>AA + EB</td>
<td>AA + EE</td>
<td>AA + FE</td>
<td>AA + AA = 40</td>
<td>AA + AB = 41</td>
<td>AA + AE = 44</td>
<td>AA + AF = 45</td>
</tr>
<tr>
<td>AA + FA</td>
<td>AA + FB</td>
<td>AA + FE</td>
<td>AA + FF</td>
<td>AA + AA = 50</td>
<td>AA + AB = 51</td>
<td>AA + AE = 54</td>
<td>AA + AF = 55</td>
</tr>
<tr>
<td>AB + AA</td>
<td>AB + AB</td>
<td>AB + AE</td>
<td>AB + AF</td>
<td>AB + AA = 02</td>
<td>AB + AB = 03</td>
<td>AB + AE = 06</td>
<td>AB + AF = 07</td>
</tr>
<tr>
<td>AB + BA</td>
<td>AB + BB</td>
<td>AB + EB</td>
<td>AB + FB</td>
<td>AB + AA = 12</td>
<td>AB + AB = 13</td>
<td>AB + AE = 16</td>
<td>AB + AF = 17</td>
</tr>
<tr>
<td>AB + EA</td>
<td>AB + EB</td>
<td>AB + EE</td>
<td>AB + FE</td>
<td>AB + AA = 42</td>
<td>AB + AB = 43</td>
<td>AB + AE = 46</td>
<td>AB + AF = 47</td>
</tr>
<tr>
<td>AB + FA</td>
<td>AB + FB</td>
<td>AB + EE</td>
<td>AB + FF</td>
<td>AB + AA = 52</td>
<td>AB + AB = 53</td>
<td>AB + AE = 56</td>
<td>AB + AF = 57</td>
</tr>
<tr>
<td>AE + AA</td>
<td>AE + AB</td>
<td>AE + AE</td>
<td>AE + AF</td>
<td>AE + AA = 08</td>
<td>AE + AB = 09</td>
<td>AE + AE = 0C</td>
<td>AE + AF = 0D</td>
</tr>
<tr>
<td>AE + BA</td>
<td>AE + BB</td>
<td>AE + EB</td>
<td>AE + FB</td>
<td>AE + AA = 18</td>
<td>AE + AB = 19</td>
<td>AE + AE = 1C</td>
<td>AE + AF = 1D</td>
</tr>
<tr>
<td>AE + EA</td>
<td>AE + EB</td>
<td>AE + EE</td>
<td>AE + FE</td>
<td>AE + AA = 48</td>
<td>AE + AB = 49</td>
<td>AE + AE = 4C</td>
<td>AE + AF = 4D</td>
</tr>
<tr>
<td>AE + FA</td>
<td>AE + FB</td>
<td>AE + EE</td>
<td>AE + FF</td>
<td>AE + AA = 58</td>
<td>AE + AB = 59</td>
<td>AE + AE = 5C</td>
<td>AE + AF = 5D</td>
</tr>
<tr>
<td>AF + AA</td>
<td>AF + AB</td>
<td>AF + AE</td>
<td>AF + AF</td>
<td>AF + AA = 0A</td>
<td>AF + AB = 0B</td>
<td>AF + AE = 0E</td>
<td>AF + AF = 0F</td>
</tr>
<tr>
<td>AF + BA</td>
<td>AF + BB</td>
<td>AF + EB</td>
<td>AF + FB</td>
<td>AF + AA = 1A</td>
<td>AF + AB = 1B</td>
<td>AF + AE = 1E</td>
<td>AF + AF = 1F</td>
</tr>
<tr>
<td>AF + EA</td>
<td>AF + EB</td>
<td>AF + EE</td>
<td>AF + FF</td>
<td>AF + AA = 4A</td>
<td>AF + AB = 4B</td>
<td>AF + AE = 4E</td>
<td>AF + AF = 4F</td>
</tr>
<tr>
<td>AF + FA</td>
<td>AF + FB</td>
<td>AF + EE</td>
<td>AF + FF</td>
<td>AF + AA = 5A</td>
<td>AF + AB = 5B</td>
<td>AF + AE = 5E</td>
<td>AF + AF = 5F</td>
</tr>
<tr>
<td>BA + AA</td>
<td>BA + AB</td>
<td>BA + AE</td>
<td>BA + AF</td>
<td>BA + AA = 20</td>
<td>BA + AB = 21</td>
<td>BA + AE = 24</td>
<td>BA + AF = 25</td>
</tr>
<tr>
<td>BA + BA</td>
<td>BA + BB</td>
<td>BA + EB</td>
<td>BA + FB</td>
<td>BA + AA = 30</td>
<td>BA + AB = 31</td>
<td>BA + AE = 34</td>
<td>BA + AF = 35</td>
</tr>
<tr>
<td>BA + EA</td>
<td>BA + EB</td>
<td>BA + EE</td>
<td>BA + FE</td>
<td>BA + AA = 60</td>
<td>BA + AB = 61</td>
<td>BA + AE = 64</td>
<td>BA + AF = 65</td>
</tr>
<tr>
<td>BA + FA</td>
<td>BA + FB</td>
<td>BA + EE</td>
<td>BA + FF</td>
<td>BA + AA = 70</td>
<td>BA + AB = 71</td>
<td>BA + AE = 74</td>
<td>BA + AF = 75</td>
</tr>
<tr>
<td>BB + AA</td>
<td>BB + AB</td>
<td>BB + AE</td>
<td>BB + AF</td>
<td>BB + AA = 22</td>
<td>BB + AB = 23</td>
<td>BB + AE = 26</td>
<td>BB + AF = 27</td>
</tr>
<tr>
<td>BB + BA</td>
<td>BB + BB</td>
<td>BB + EB</td>
<td>BB + FB</td>
<td>BB + AA = 32</td>
<td>BB + AB = 33</td>
<td>BB + AE = 36</td>
<td>BB + AF = 37</td>
</tr>
<tr>
<td>BB + EA</td>
<td>BB + EB</td>
<td>BB + EE</td>
<td>BB + FF</td>
<td>BB + AA = 62</td>
<td>BB + AB = 63</td>
<td>BB + AE = 66</td>
<td>BB + AF = 67</td>
</tr>
<tr>
<td>BB + FA</td>
<td>BB + FB</td>
<td>BB + EE</td>
<td>BB + FF</td>
<td>BB + AA = 72</td>
<td>BB + AB = 73</td>
<td>BB + AE = 76</td>
<td>BB + AF = 77</td>
</tr>
<tr>
<td>BE + AA</td>
<td>BE + AB</td>
<td>BE + AE</td>
<td>BE + AF</td>
<td>BE + AA = 28</td>
<td>BE + AB = 29</td>
<td>BE + AE = 2C</td>
<td>BE + AF = 2D</td>
</tr>
<tr>
<td>BE + BA</td>
<td>BE + BB</td>
<td>BE + EB</td>
<td>BE + FB</td>
<td>BE + AA = 38</td>
<td>BE + AB = 39</td>
<td>BE + AE = 3C</td>
<td>BE + AF = 3D</td>
</tr>
<tr>
<td>BE + EA</td>
<td>BE + EB</td>
<td>BE + EE</td>
<td>BE + FF</td>
<td>BE + AA = 68</td>
<td>BE + AB = 69</td>
<td>BE + AE = 6C</td>
<td>BE + AF = 6D</td>
</tr>
<tr>
<td>BE + FA</td>
<td>BE + FB</td>
<td>BE + EE</td>
<td>BE + FF</td>
<td>BE + AA = 78</td>
<td>BE + AB = 79</td>
<td>BE + AE = 7C</td>
<td>BE + AF = 7D</td>
</tr>
<tr>
<td>BF + AA</td>
<td>BF + AB</td>
<td>BF + AE</td>
<td>BF + AF</td>
<td>BF + AA = 2A</td>
<td>BF + AB = 2B</td>
<td>BF + AE = 2E</td>
<td>BF + AF = 2F</td>
</tr>
<tr>
<td>BF + BA</td>
<td>BF + BB</td>
<td>BF + EB</td>
<td>BF + FB</td>
<td>BF + AA = 3A</td>
<td>BF + AB = 3B</td>
<td>BF + AE = 3E</td>
<td>BF + AF = 3F</td>
</tr>
<tr>
<td>BF + EA</td>
<td>BF + EB</td>
<td>BF + EE</td>
<td>BF + FF</td>
<td>BF + AA = 6A</td>
<td>BF + AB = 6B</td>
<td>BF + AE = 6E</td>
<td>BF + AF = 6F</td>
</tr>
<tr>
<td>BF + FA</td>
<td>BF + FB</td>
<td>BF + EE</td>
<td>BF + FF</td>
<td>BF + AA = 7A</td>
<td>BF + AB = 7B</td>
<td>BF + AE = 7E</td>
<td>BF + AF = 7F</td>
</tr>
<tr>
<td>EA + AA</td>
<td>EA + AB</td>
<td>EA + AE</td>
<td>EA + AF</td>
<td>EA + AA = 80</td>
<td>EA + AB = 81</td>
<td>EA + AE = 84</td>
<td>EA + AF = 85</td>
</tr>
<tr>
<td>EA + BA</td>
<td>EA + BB</td>
<td>EA + EB</td>
<td>EA + FB</td>
<td>EA + AA = 90</td>
<td>EA + AB = 91</td>
<td>EA + AE = 94</td>
<td>EA + AF = 95</td>
</tr>
<tr>
<td>EA + EA</td>
<td>EA + EB</td>
<td>EA + EE</td>
<td>EA + FE</td>
<td>EA + AA = C0</td>
<td>EA + AB = C1</td>
<td>EA + AE = C4</td>
<td>EA + AF = C5</td>
</tr>
<tr>
<td>EA + FA</td>
<td>EA + FB</td>
<td>EA + EE</td>
<td>EA + FF</td>
<td>EA + AA = D0</td>
<td>EA + AB = D1</td>
<td>EA + AE = D4</td>
<td>EA + AF = D5</td>
</tr>
<tr>
<td>EB + AA</td>
<td>EB + AB</td>
<td>EB + AE</td>
<td>EB + AF</td>
<td>EB + AA = 82</td>
<td>EB + AB = 83</td>
<td>EB + AE = 86</td>
<td>EB + AF = 87</td>
</tr>
<tr>
<td>EB + BA</td>
<td>EB + BB</td>
<td>EB + EB</td>
<td>EB + FB</td>
<td>EB + AA = 92</td>
<td>EB + AB = 93</td>
<td>EB + AE = 96</td>
<td>EB + AF = 97</td>
</tr>
<tr>
<td>EB + EA</td>
<td>EB + EB</td>
<td>EB + EE</td>
<td>EB + FF</td>
<td>EB + AA = EA</td>
<td>EB + AB = C3</td>
<td>EB + AE = C6</td>
<td>EB + AF = C7</td>
</tr>
<tr>
<td>EB + FA</td>
<td>EB + EB</td>
<td>EB + EE</td>
<td>EB + FF</td>
<td>EB + AA = FA</td>
<td>EB + AB = D2</td>
<td>EB + AE = D3</td>
<td>EB + AF = D7</td>
</tr>
<tr>
<td>EE + BA</td>
<td>EE + BB</td>
<td>EE + EB</td>
<td>EE + FB</td>
<td>EE + AA = 98</td>
<td>EE + AB = 99</td>
<td>EE + AE = 9C</td>
<td>EE + AF = 9D</td>
</tr>
<tr>
<td>EE + EA</td>
<td>EE + EB</td>
<td>EE + EE</td>
<td>EE + FE</td>
<td>EE + AA = C8</td>
<td>EE + AB = C9</td>
<td>EE + AE = CC</td>
<td>EE + AF = CD</td>
</tr>
<tr>
<td>EE + FA</td>
<td>EE + FB</td>
<td>EE + EE</td>
<td>EE + FF</td>
<td>EE + AA = D8</td>
<td>EE + AB = D9</td>
<td>EE + AE = DC</td>
<td>EE + AF = DD</td>
</tr>
<tr>
<td>EF + BA</td>
<td>EF + BB</td>
<td>EF + EE</td>
<td>EF + FF</td>
<td>EF + AA = 9A</td>
<td>EF + AB = 9B</td>
<td>EF + AE = 9E</td>
<td>EF + AF = 9F</td>
</tr>
<tr>
<td>EF + EA</td>
<td>EF + EB</td>
<td>EF + EE</td>
<td>EF + FF</td>
<td>EF + AA = CA</td>
<td>EF + AB = CB</td>
<td>EF + AE = CE</td>
<td>EF + AF = CF</td>
</tr>
<tr>
<td>EF + FA</td>
<td>EF + FB</td>
<td>EF + EE</td>
<td>EF + FF</td>
<td>EF + AA = DA</td>
<td>EF + AB = DB</td>
<td>EF + AE = DE</td>
<td>EF + AF = DF</td>
</tr>
<tr>
<td>FA + EA</td>
<td>FA + EB</td>
<td>FA + EE</td>
<td>FA + FF</td>
<td>FA + AA = E0</td>
<td>FA + AB = E1</td>
<td>FA + AE = E4</td>
<td>FA + AF = E5</td>
</tr>
<tr>
<td>FA + FA</td>
<td>FA + FB</td>
<td>FA + EE</td>
<td>FA + FF</td>
<td>FA + AA = F0</td>
<td>FA + AB = F1</td>
<td>FA + AE = F4</td>
<td>FA + AF = F5</td>
</tr>
<tr>
<td>FB + EA</td>
<td>FB + EB</td>
<td>FB + EE</td>
<td>FB + FF</td>
<td>FB + AA = E2</td>
<td>FB + AB = E3</td>
<td>FB + AE = E6</td>
<td>FB + AF = E7</td>
</tr>
<tr>
<td>FB + FA</td>
<td>FB + FB</td>
<td>FB + EE</td>
<td>FB + FF</td>
<td>FB + AA = F2</td>
<td>FB + AB = F3</td>
<td>FB + AE = DE</td>
<td>FB + AF = DF</td>
</tr>
<tr>
<td>FE + EA</td>
<td>FE + EB</td>
<td>FE + EE</td>
<td>FE + FF</td>
<td>FE + AA = E8</td>
<td>FE + AB = E9</td>
<td>FE + AE = EC</td>
<td>FE + AF = ED</td>
</tr>
<tr>
<td>FE + FA</td>
<td>FE + FB</td>
<td>FE + EE</td>
<td>FE + FF</td>
<td>FE + AA = F8</td>
<td>FE + AB = F9</td>
<td>FE + AE = FC</td>
<td>FE + AF = FD</td>
</tr>
<tr>
<td>FF + EA</td>
<td>FF + EB</td>
<td>FF + EE</td>
<td>FF + FF</td>
<td>FF + AA = E4</td>
<td>FF + AB = EB</td>
<td>FF + AE = EE</td>
<td>FF + AF = EF</td>
</tr>
<tr>
<td>FF + FA</td>
<td>FF + FB</td>
<td>FF + EE</td>
<td>FF + FF</td>
<td>FF + AA = FA</td>
<td>FF + AB = FB</td>
<td>FF + AE = FE</td>
<td>FF + AF = FF</td>
</tr>
</tbody>
</table>
Egbert II Communications Disk

(RTTY/CW/TRANSFER)
Egbert Software
W.H. Nail Co.

Requirements:
Apple ][ Plus, //, or compatible
DOS 3.3 Master Disk
Egbert II Communications Disk
DEMUFFIN program (see Muffins)
Blank disk

By Keith S. Goldstein, MD

Since I haven’t read anything about the Egbert II disk, other than the ad marketing it, I’ll assume that not too many people are aware of the ingenious ideas that are packed into this system disk. But before I show you how to crack and modify the disk, I’ll describe and explain the system a bit. This is only a very brief overview and isn’t meant to be comprehensive. The disk is chock full of pleasant surprises.

The Egbert II Communications disk contains three very powerful main programs. The RTTY (Radio Teletype) program receives and transmits radio teletype signals. It generates RTTY tones on transmit and decodes the RTTY tones on receive. Some other goodies it has are a “Mailbox” option and an audio frequency counter option.

The CW (Continuous Waves) program does the same for Morse Code, while the Transfer program allows you to send and receive Applesoft, Integer and Binary programs over the telephone or radio. The special attraction is that all of the tone generation and decoding is done from within the program, so there is NO HARDWARE INTERFACE REQUIRED!

The cassette I/O plugs are used from the rear of the computer. Simply plug the cassette input into the speaker/earphone jack of any communications receiver (or telephone amplifier for the Transfer program) and plug the cassette output into the microphone jack of the transceiver. That’s all you need to send/receive RTTY or CW.

For the non-amateur radio operator, you can use the RTTY program to receive foreign and domestic wire news services and telegrams such as those supplied by UPI.

The CW (Morse Code) program will decode or send Morse Code at rates from three to 125 words per minute. It is a great teacher for those interested in learning Morse Code or those who simply want to eavesdrop on radio hams.
Both the RTTY and CW programs use the on-board speaker-to-monitor arrangement. The Transfer program will work just as well with any transceiver, or you can transfer programs over the telephone without a modem! All you need is a cheap amplifier (like the Radio Shack 277-1008 which costs $11.00) and a telephone pick-up coil (also available at the 'Shack for about $2.00). It sure beats the price of a modem!

Now that your mouth is watering at the possibilities of getting the news and listening in on private and government teletype communications without messing around with any extra hardware, order one! I heartily recommend this disk. It's worth every penny.

Unfortunately, once you have the disk you will find some unfriendly little annoyances programmed-in there. Haven't you ever wished that you could go from receive to transmit RTTY without that darn CW (Morse Code) identifier breaking your concentration? Doesn't having your name and serial number flashed at you constantly annoy you? Wouldn't you just love to have the programs load super fast by using a speedy DOS? Wouldn't you like to be able to modify the programs to your heart’s content? Wouldn't you just like to have a backup copy and don't have a nibble-copier? You can find out how to do these things and more below.

The Lock

The Egbert II Communications Disk uses several simple and yet very effective tricks to prevent the user from discovering its secrets. The major copy-protection scheme it incorporates is its DOS and the way the disk is initialized. It won't allow any standard copy programs or nibble-copiers (without changing parameters) to duplicate it.

To unprotect the programs on the original disk, each one must be loaded by the Egbert DOS and saved by a normal DOS onto a standard disk. After you have transferred the programs they will all, eventually, bomb into the monitor because the author included some checking routines to be sure that you are using his custom DOS. We will defeat those routines, too.

The author was quite thorough in his protection scheme. It is impossible to stop the computer by using reset or any normal method. Egbert DOS is a standard 3.2 with many modifications and patches and the author patched over the INIT command with his own routines. Among other things, this patch will set the Run Flag ($D6) to FF and disable the CATALOG, INT, and FP commands at every disk access (pretty shrewd, eh?) Also, since the programs must access normal DOS disks, Egbert has the normal DOS routines loaded and moved into place immediately after any of the programs have been loaded. This makes it
impossible to use the resident DOS to access the Egbert disk for
the purpose of transferring the programs once any of them starts
running.

These problems can be solved by using a modification of the
DOS 3.3 Master Disk program called MUFFIN. The modified
MUFFIN is called DEMUFFIN PLUS.

As you recall, MUFFIN is a machine-language program which
will transfer programs or files from DOS 3.2 to DOS 3.3. In
general, it reads from one disk format and writes to a different
disk format. The modification to MUFFIN allows it to use
whatever DOS is present in the machine to read from the locked
disk and to write the file out onto a standard DOS 3.3 disk. This
gem is just what is needed to transfer the Egbert files to a DOS
3.3 disk.

The Softkey

Here are the steps to follow in order to transfer the Egbert
files to a standard DOS 3.3 disk:

1) Boot the system master and format a blank disk

    INIT HELLO

2) Delete the HELLO program

    DELETE HELLO

3) Load DEMUFFIN PLUS in a safe place where the boot won’t
    mess it up

    BLOAD DEMUFFIN PLUS, A$6000

    The Egbert DOS will “lock you out” of the machine once it is
    loaded so we will have to allow the Egbert DOS to load in and
    then we will have to stop it before it initializes itself (i.e: takes
    over). This can best be done by a technique called Boot-Tracing.
    The process takes a little time but the satisfaction and knowledge
    gained is well worth the effort.

    In general, this technique involves loading a chunk or stage of
    the boot into memory, examining it, moving it to a safe place in
    RAM, modifying it to work at the new location and stopping the
    boot (after loading each new stage) so that we will always be in
    control of the computer. In this way, the Egbert DOS can be
    allowed to fully load itself and then be forced to halt.

4) Put the Egbert disk in Drive 1, Slot 6 and enter the monitor

    CALL -151

5) Move the bootcode in the controller card to RAM

    8600<C600.C6FFM
6) Make the moved bootcode JuMP into the monitor after loading track 0, sector 0

*86F8:4C 59 FF N 8600G

_The Drive will stay on for the remainder of this process._

7) Move the first boot stage to a new location

8000<800.9FFM

8) Now we must change a few locations so it will work at this new location

8003:BD 00 80

9) Make this new stage JuMP into the monitor when finished

8049:4C 59 FF

10) Now we have to tell the first stage where our new second stage has been moved to

86F8:4C 01 80

11) Load the third stage of the boot

8600G

12) Since the last stage ended with a JuMP to location $301, we know where this third stage of the boot was just loaded. To be certain that it won't be over-written by the next stage, we will move it to a safe place

8300<300.3FFM

13) Again we have to change a few locations so that this stage will function properly at this new location in memory

8313:AD CC 83

833C:AD CC 83

14) The jump out of the fourth stage is not immediate, but only after many jumps to a certain subroutine does it continue on to the next stage of the boot. Therefore, we'll place a short program for this stage to jump to. We will also check it to see if it is going to the subroutine again (and if so, let the program continue) or if not, then stop and JuMP into the monitor. We will place our little program at $8400 but we need to intercept the program to JuMP to our little routine.

8343:4C 00 84

15) Enter this little routine

8400:A5 3E C9 D5 D0 03 6C 3E

8408:00 4C 59 FF
16) Now we mustn't forget to let our stage three know where this stage four was moved to so that it will be able to continue to load another chunk of the DOS for us.

8049:4C 01 83

17) At this point we will let the computer use all of our routines to load in stage four of the boot.

8600G

18) Now we've got almost all of the DOS loaded. Let's see where the final stage has been loaded.

83CC

(The number you should see is $B6. Add 1 to it ($B6+1) = $B700; therefore, our next JuMP will be to $B700.)

19) Since we know where this last portion has been loaded, we are ready to complete the boot and have it stop just before it begins to start up the DOS. Let's move this fourth stage out of the way.

5700<B700.BFFF

20) Now, we see the familiar DOS initialization routine JuMP near the beginning of all this stuff we have just moved. Once it has finished loading itself into the machine, let's have it JuMP into the monitor instead of starting the EGBERT DOS.

5747:4C 59 FF

21) We also must not forget to let the previous third stage boot know where we have moved this final stage to.

8409:4C 00 57

22) We are now ready to allow the entire EGBERT DOS to be read into the machine and it will stop just before it can take control, which is exactly what we want!

8600G

(The disk will stop spinning now since the boot has finished.)

23) Since we stopped the EGBERT DOS from being able to initialize itself, it wasn't able to fill-in its page-3 vector table. In order for our previously entered DEMUFFIN PLUS to function, it needs these vectors intact. This can be easily accomplished since the page-3 vector table image already exists within the Egbert DOS image. Just move it to page-3.

3D0<9E51.9E7
24) Move the DEMUFFIN PLUS program back to $803 and start it running

\[ 803<6000.8000M \]
\[ 803G \]

25) Select Convert Files from the menu. For "File Name?" enter "=" (The equals sign is the Wildcard character). Transfer all files from the original disk to the standard initialized DOS 3.3 disk.

*(Do not attempt to transfer the first seven of those files shown in INVERSE as they are DUMMY FILES and will cause errors if you try to copy them.)*

**Fixing the Files**

Now that you have all of the programs on a standard DOS 3.3 disk, you are ready to remove the checks for the non-standard DOS so they will function correctly. Most of the changes that will be made will remove a POKE 214,255 that sets the Run Flag.

26) Boot your DOS 3.3 Master Disk. Remove it and insert your new Egbert DOS 3.3 disk

27) Load and modify the HELLO program

```
LOAD HELLO
2 INVERSE : FOR I = 1 TO 40 : PRINT "@" ; : NEXT :
    VTAB 15 : FOR I = 1 TO 40 : PRINT "@" ; : NEXT :
    FOR I = 2 TO 14 : VTAB I : HTAB 1 : PRINT
    "@" ; : HTAB 40 : PRINT "@" ; : NEXT : NORMAL
    : VTAB 3 : HTAB 7 : PRINT "EGBERT.COMMUNICATIONS.DISK" : PRINT
SAVE HELLO
```

28) Modify the MAIN program

```
LOAD MAIN
8010 VTAB 3 : HTAB 7 : PRINT "EGBERT.COMMUNICATIONS.DISK" : VTAB 5 :
    HTAB 16 : PRINT "MAIN.MENU" : RETURN
SAVE MAIN
```
29) Modify the RTTY program

```
LOAD RTTY
3 B$ = CHR$(4) : GOSUB 91 : VTAB 16 : HTAB 11 :
FLASH : PRINT "LOADING PROGRAMS" :
NORMAL : D$ = CHR$(219) : E$ = CHR$(221) :
PRINT B$; "BLOAD^COMBO^1^8^83, D1" : PRINT B$; "BLOAD^SPL, D1"
79 TEXT : HOME : PRINT "BYE!" : PRINT CHR$(7) :
END
85 ONERR GOTO 73
SAVE RTTY
```

30) Modify the ECW program

```
LOAD ECW
30 POKE 115,0 : POKE 116,147 : POKE 111,0 : POKE
112,147
110 REM
120 REM
200 REM
SAVE ECW
```

31) Modify the XFER program

```
LOAD XFER
135 REM
137 REM
SAVE XFER
```

32) Modify the TRANSFER] program

```
LOAD TRANSFER]
675 REM
1035 REM
2011 REM
2020 REM
2030 REM
2040 REM
2050 REM
2060 REM
2070 REM
2080 REM
SAVE TRANSFER]
```

33) Modify the BUFFER/MESSAGE program

```
LOAD BUFFER/MESSAGE
145 REM
150 REM
220 VTAB 23 : PRINT "THIS WILL TAKE ABOUT 30 SECONDS" : VTAB 17
```
225 REM
SAVE BUFFER/MESSAGE

34) Modify the COMBO 1-8-83 program

**BLOAD COMBO 1-8-83**

This program checks for the non-standard patch to the CATALOG command on the locked disk. It checks for a $60 in the DOS. If it is there, the program continues. If it isn’t there, the RTTY program bombs. The normal DOS 3.3 value is a $20. To enable it to work perfectly with DOS 3.3, one change is required which makes the check routine look for the normal DOS 3.3 value of $20 instead of the $60. (Incidentally, the $60 of the non-standard DOS disables the CATALOG command; it causes the command CATALOG to be ignored). The change is as follows:

```
CALL -151
54BC:20
BSAVE COMBO 1-8-83, A$5000, L$0AD5
```

35) Fix the same CATALOG patch in the RCV program

**BLOAD RCV**

```
4302:20
BSAVE RCV, A$4000, L$58B
```

36) Fix the CATALOG patch in the XMT program

**BLOAD XMT**

```
52C2:20
BSAVE XMT, A$5000, L$3E8
```

37) Delete the image of DOS 3.3 on the disk

**DELETE DOS 3.3**

38) Delete the DOS mover from the disk

**DELETE DOS MOVE 3.3**

You now have a fully functional DOS 3.3 version of the entire EGBERT RTTY/CW/TRANSFER system to customize at your discretion.

For starters, here is how to obliterate the serial number of the diskette.

**LOAD RTTY**
91 HOME: INVERSE: FOR I = 1 TO 40: VTAB 1:
PRINT "@" CHRS (8): VTAB 7: PRINT "@":
NEXT: FOR I = 2 TO 6: VTAB I: HTAB 1:
PRINT "@": HTAB 40: PRINT "@": NEXT:
NORMAL: VTAB 3: HTAB 9: PRINT
"EBERT^^RTTY^^PROGRAM": HTAB 9: PRINT
"WRITTEN^^BY^^G.W.^^EBERT": HTAB 14: POKE 34,7: RETURN

The following are immediate execution commands.

F2 = PEEK (175) + PEEK (176) * 256 - 8
FOR A = 1 TO 4: POKE F2+A, 0: NEXT
SAVE RTTY

Again in immediate execution mode type

LOAD ECW
260 REM
370 REM
F2 = PEEK (175) + PEEK (176) * 256 - 8
FOR A = 1 TO 4: POKE F2 + A, 0: NEXT
SAVE ECW

Then continue into the monitor

CALL -151
BLOAD COMBO 1-8-83
50D2:FF
50D3<50D2.50E8M
BSAVE COMBO 1-8-83, A$5000, L$AD5

And the next

BLOAD RCV
414E:00
414F<414E.415BM
415C:FF FF
BSAVE RCV,A$4000,L$58B

And that’s all there is to it.

To aid you in you customizing, here is a list of programs with brief descriptions. Have fun!

HELLO: Boot-up title.
MAIN: Main menu. Uses PRINT SET, BUFFER/MESSAGE, MESSAGE.OBJ, CODE, RTTY, ECW, XFER files
RTTY: RTTY program body, uses COMBO 1-8-83, SPL files.
COMBO 1-8-83: RTTY machine language portion
SPL: Printer spooler machine language portion.
PRINT SET: Printer set-up program. Uses SPL.
ECW: CW program body. Uses XMT, RCV, SPL, GP files.
XMT: CW transmit machine language portion.
RCV: CW receive machine language portion.
GP: CW game paddle overlay in machine language.
CODE: Contains the number of programs on the disk.
XFER: Transfer title and set-up program.
TRANSFER: TRANSFER program body. Uses XFER 3800.
XFER 3800: TRANSFER machine language portion.
BUFFER/MESSAGE: Buffer/message program. Creates messages, prints the buffer; uses MESSAGES, MESSAGE.OBJ files.
MESSAGE.OBJ: Contains the canned messages and saved files. Not locked on the original disk.
DOS 3.3: Overlay of standard DOS 3.3
DOS MOVE 3.3: Relocates DOS 3.3 and overwrites the Egbert DOS with standard DOS 3.3
Getting On The Right Track

By Robert Linden

Requirements:
Apple ][+ or compatible
Apple type disk drive
Bit copier, sector editor or other program that seeks specific tracks on demand
STABILO fine-point pen, or other similar marker

When making backups of copy-protected programs, there will be times when the backup will not boot. It might keep rebooting continuously, spin with no head movement, stop, or do something else it shouldn't. Often this is the result of just a few tracks being incorrectly backed-up. Finding these tracks quickly will speed up your task greatly. Here's how:

Turn off your computer. Remove the screws holding the cover on your disk drive (Warning: This will void your warranty) and slide the cover to the rear and off the drive.

Turn on your computer and boot a program that seeks specific tracks on demand. Now you will need to find both the frame that holds the read/write head and the cam that drives this frame (See Figure 1).

Have the drive seek track 0, then track 22 (hex) while you are viewing the interior of the drive from one side. The object moving rapidly over (and under) the disk is the frame that holds the read/write head. Below this frame you will see a three-to-four-inch round object which turns only when the frame moves. This is the cam that drives the frame holding the read/write head.

A common method of indicating tracks for future use is to place a reference mark on the read/write frame and then, as the drive is stepped through the tracks by the track-seeking program, to mark each track on a nearby, motionless part of the drive.

The problem with this method is that the marks are as close together as the tracks on the disk. To greatly increase the distance between the track marks I prefer marking the cam instead.

If your cam is made of shiny plastic you will need a fiber-tip pen intended for writing on plastic, such as a STABILO fine-point. If your cam is made of rough plastic, similar to Bakelite, you could use paint for greater visibility. In this case, use a water-based, model paint to avoid any risk of the paint dissolving part of the cam. Use a fine-tipped brush and, if you like, several different colors for ease in identifying the tracks. Whichever method you use, make a test mark on the cam to be sure the marks will adhere.
We have recently become aware of a figure omission in paragraph 3 of 'Getting On The Right Track' by Robert Linden (page 99).

Figure 1
First, place a reference mark on the most easily visible spot that is directly next to the cam.

Then have the drive seek track 0 and place a mark on the vertical edge of the cam. Make sure this mark lines up with the reference mark.

If you feel uneasy about touching the interior of the drive while it’s on, or if you’re not sure about what you can and can’t touch, you should turn off the computer after seeking a track and then turn it on again after you have finished the marking for that track.

Next, have the drive seek track 18 (hex). On my drive this will turn the cam one complete revolution to the mark made for track 0. If your drive is different, find the track that does line up to the mark for track 0. On top of the cam above the mark write a small 0/18. Do NOT place any marks in the spiral groove that is engraved on top of the cam. This groove is used by the cam to move the read/write head, so take care not to gum it up.

Note where the read/write frame is in relation to the center of the cam. Now have the drive seek track 0 again. You will notice that the read/write frame has moved much closer to the edge of the cam. This is how you can tell if the drive is on track 0 or track 18.

Now work your way around the cam, seeking each track, marking it and labeling it, until you get to track 22. I label each track in hex (i.e. base 16) instead of base 10 since most references to the tracks are in hex. Note that the marks for tracks 18-22 (hex) will overlap with the marks for tracks 0-10 (hex). In the case where two tracks use the same marks you must take note of the read/write head frame in relation to the cam in order to distinguish which track is being accessed.

Boot the disk to be backed-up while you watch the cam to see if the drive seeks any \(\frac{1}{2}\) tracks or anything past track 22. Using this information, try to make a backup. If the backup will not boot properly, watch the tracks over which the drive goes before the backup fails. These are the tracks which have one or more bad sectors on them. If the drive stops or spins continuously on one track, re-do that one track. If the drive seeks the same tracks over and over again, when it did not do that on the original disk, then re-do those tracks. Good luck!
Hard Hat Mack
Electronic Arts

Requirements:
Apple II Plus or compatible
Hard Hat Mack disk
Blank initialized disk with no "HELLO" program
Some knowledge of boot code tracing or machine language

By Rich Lyon

Hard Hat Mack is an addictive construction-site game with three different levels. I was first introduced to it at the local computer store and couldn't stop playing it. While there, I took some time to examine the boot code on the game disk and found it to have a very strange boot code, one like I had never seen before. About a month later I decided to buy the disk for, mainly, two reasons: I liked the game, and I wanted to face the challenge of breaking the copy-protection scheme.

For those of you who are not familiar with the boot process, here is a general explanation. When any disk is booted on the Apple, control is transferred to the boot program which is at $C600. If your disk controller card is in slot 5, the program will be found at $C500. It will be assumed that the card is in slot 6 to keep things simple. When executed, this program will read in track 0, sector 0 from the disk and put it in at $800. It will, then, jump to $801.

Depending on the disk, from this point another boot stage will be loaded in and, eventually, the main program will be read into memory and executed.

When it comes to copy-protected disks, almost every disk is different. The unique thing about Hard Hat Mack is that the first boot stage loaded in takes 16 pages of memory. In most cases, boot 1 only occupies one page of memory. The advantage of this lengthy boot stage is that this is the only boot stage. From here on, the game is loaded right in.

Blue-Collar Boot Code Tracing

Here are the steps used to boot code trace Hard Hat Mack:

1) Enter the Apple's monitor
   CALL -151
2) Memory move the boot program down to a page in RAM so it can be modified to load in the next boot stage

\[9600<6C00,6CFF\]

3) Change the JMP $801 to a JMP $B047

\[96F9:47\]

Why jump to $B047? After tracing the code for the first time, I ended up jumping there upon exiting the first boot stage. From there on I jumped to $B047 immediately.

4) Put a short routine at $B047 to shut off the drive motor and return to the monitor. A JMP $FF59 will jump to the monitor

\[B047:8D\]

5) Insert the original Hard Hat Mack disk and type

\[9600G\]

This will execute the first boot stage to load the next boot stage into $800. This will take about five seconds because it has to load in 16 pages of memory. Usually, this boot stage occupies only one page of memory but, if you check the value at $800, you will find a $10 (16 decimal) where normally you would find a $1. This number tells the first boot stage how many sectors to read in.

6) Memory move pages $8 through $18 to $BO00.

\[B000<800,1800\]

If you list through the boot stage at $800 (801L) you will find that all it does is the memory move and then jumps to $B047.

Now, rather than modify the code at page $8, it is easier to put it where it belongs and jump directly there from boot number 1. The next step is finding the jump to the start of the program. In other words, a JMP instruction to somewhere other than within the boot stage.

The only jumps I found were two indirect ones to $42. At first I thought these were used (at least one of them) to jump to the start of the program. I traced them and found that they were not used to exit the boot. That left me knowing that I was faced with a problem.

Somewhere within this lengthy boot stage is a hidden or a coded jump. Rather than trace through everything that looked suspicious, I decided to try for a one-in-a-million shot.

I had traced the boot code about ten times prior to this and remembered one place where I had halted the boot code and most of the program had been loaded in. I went over it again and stopped in that place. Then I paged through memory and looked for something that might be the start of the game. It didn’t take much looking because I found something interesting.
right at $800. Without even testing it, I assumed that it was the start of the program.

Now, my next step was to boot code trace the disk again and halt it in the same place. But instead of coming to a complete stop, I would have to call a short routine to cover up the first three bytes at $800 with a 4C 59 FF. That way, if the boot code jumped to $800, the start of the program would cause a jump into the monitor.

Once everything was set, I executed the boot and waited with high hopes. Just as the game was about to start, I heard a beep and the cursor appeared. Indeed, $800 was the start of the game. The place I interrupted the boot stage was at $BBC4 and at that location was a JMP $BBD4. What I did there was to set $BBC4 to jump to $B100 and at $B100 I put the routine to cover up the start of the program with a JMP $FF59. $B100 is a safe place to put data because it is only the data for the Electronic Arts logo.

7) Set a jump to $B100 at $BBC4
   
   BBC4:4C 00 B1

8) Enter the routine to cover up the start of the program
   
   B100:A9 4C 8D 00 08 A9 59 8D
   B108:01 08 A9 FF 8D 02 08 4C
   B110:D4 BB

9) Reboot the disk
   
   9600G

This will load in the entire game and return control to you. When the prompt appears you are ready for the last step before saving the game. Right now we want to restore what was originally at $800 before the routine at $B100 covered it up. It was a JSR $2204.

10) Restore the code that was at $800
   
   800:20 04 22

Next, we will reboot DOS. First, we must move page $8 to a safe place or it will be overwritten when we reboot. The question is where to put it.

Paging through memory, I found an area that looked like it contained “garbage.” Actually, I concluded that all memory from $3400 to $3FFF was unused because the game did nothing with it before clearing the hi-res page.

11) Memory move page $8 to page $34

   3400<800.8FFM
12) Put in a blank slave disk with no "HELLO" program and type

   65P

13) Now, after booting DOS, enter the monitor again

   CALL -151

14) Next, we will move page $8 back to its proper place from page $34

   800<3400.34FFM

The game could be saved now but it would not work.

   When the space bar is pressed to begin the game, a check is
done to the disk to make sure that the Hard Hat Mack disk is
present. The only problem is in finding where the disk is
accessed. nowing that this happens when the space bar is
pressed, when you do that, look for a read from the keyboard
and a check. I found this in the subroutine at $BC8. There was
also an LDA $C000 and, further on, a CMP #$A0. When the
space bar is pressed, this subroutine sets a flag byte and returns.

   The next step is to find out where the subroutine at $BC8 is
called from. I found this at $84E. After calling the subroutine it
checks the flag and, if it is set, continues.

   At $864 is a JSR $4D34. This is part of the game beginning
sequence. The subroutine at $4D34 does a lot of playing with the
stack. By tracing the PLAs and PHAs, I found that it leaves two
extra values on the stack and then does an RTS. This is a
disguised jump. Confused? When an RTS is executed, the two
top values are taken off the stack and the computer jumps to the
address of those two values, plus one. When I checked the two
values left on the stack, I found $FF and $04. Adding one to
$04FF you get $0500 and that's where it was going.

   The next text page is $0500 and there was nothing there upon
exiting the boot. At $803 there is a JSR $3300 and if you list
through $3300 you will see that it moves $3000 through $32FF
down to $500 so when you list through $3000 you are actually
seeing what will be at $0500.

   Looking at $3000 there are disk access commands in the
assembly. Therefore, you can assume that this is where the disk
is checked. All that we have to do to remove this disk check is
to change the subroutine at $4D34 so it does not push two extra
values on the stack. Simply change the PHA at $4D53 to a PLA
so that instead of pushing on the second value it would pull off
the first, hence, leaving the stack the same.

15) Change the operation at $4D34 from PHA to PLA

   4D53:68
All we have to do now is save the game to disk. Since DOS does not allow us to save a file longer than $7FFF bytes and we need to save $8D00, we have to change a byte in DOS.

16) Patch DOS so that we can save this long a file

A964:FF

17) Finally save the BRUNable version of Hard Hat Mack

BSAVE HARD HAT MACK,A$800,L$8D00
This will save all memory from $800 to $94FF. Actually, the game loads in past $9500 but, after testing the game, I found it to work fine. All the memory above $9500 is just “garbage” memory.

Finally, if you wish to compress the file remember that pages $34 to $3F are free. This will save you 12 sectors on your disk. I often shorten game files as much as possible.
Hidden Locations Revealed

By Enrique Gamez

Requirements:
Apple ][ or ][+ only (will not work with the //e)
Disk Organizer II by Sensible Software
Small-gauge insulated wire (no. 24)
16-pin DIP socket

We've been taking for granted that it's possible to break into any program by just switching to the old monitor F8 ROM and hitting reset. However, with Disk Organizer II, this causes a carriage return and the text page to scroll, thus losing any information placed on the first line of text page 1. This information is vital when trying to perform a softkey.

To solve the scrolling problem I discovered a previously inaccessible set of locations for the first line on the text page. My technique involves gaining control over the screen soft switches that display text page 1. Preventing it from scrolling and allowing recovery of the needed information from the first line. I'm sure this technique is used by many other programs, so read on even if you don't own Disk Organizer II.

Hidden Addresses

I first noticed some indirect references in the assembly code to locations $400-426. The makers of Disk Organizer II tried to conceal jump addresses by storing them in the plowable first line of the text page. These crucial entry points are for the routines which perform the delete, rename, exhume, move, purge and change boot tasks.

In following this procedure you'll force the Apple to display the text page, no matter what the program in memory would like to do. You'll learn a little about soft switches on the way and, most importantly, how to gain control over them.

Technical Background

Having a memory-mapped screen is very convenient; writing to any position on any screen becomes as simple as POKEing a value or STAing a specific byte. However, not so convenient is the experience of having some locations self-modify as you're...
trying to read them. Have you ever done a hexdump of the $400 to $7FF area while viewing the text page? Total nonsense.

The screen soft switches are what allow you a "window" into the Apple. By flipping a switch here and there you can literally browse through memory (without changing anything there).

Screen Switching Demo

Type the following "Screen Switching Demo" and watch what happens. If you goof up, just turn off the computer, reboot, and start over. Though you may lose sight of what you're typing, keep going. You'll just need to be a more careful typist.

<table>
<thead>
<tr>
<th>Type this</th>
<th>Explanation</th>
<th>View window</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL -151</td>
<td>Enter monitor</td>
<td>$400-7FF</td>
</tr>
<tr>
<td>C054:0</td>
<td>Select page 1. Nothing happens.</td>
<td>$400-7FF</td>
</tr>
<tr>
<td>C053:0</td>
<td>Select mixed screen. Nothing happens.</td>
<td>$400-7FF</td>
</tr>
<tr>
<td>C051:0</td>
<td>Select text screen.</td>
<td>$400-7FF</td>
</tr>
<tr>
<td>C050:0</td>
<td>Select Lo-res graphics, mixed.</td>
<td>$400-7FF</td>
</tr>
<tr>
<td>C052:0</td>
<td>Select full screen graphics.</td>
<td>$400-7FF</td>
</tr>
<tr>
<td>C051:0</td>
<td>Back to text.</td>
<td>$400-7FF</td>
</tr>
<tr>
<td>C055:0</td>
<td>Select text page 2. What a mess.</td>
<td>$800-BFF</td>
</tr>
<tr>
<td>C050:0</td>
<td>Select Lo-res graphics page 2.</td>
<td>$800-BFF</td>
</tr>
<tr>
<td>C053:0</td>
<td>Select mixed text and graphics page 2</td>
<td>$800-BFF</td>
</tr>
<tr>
<td>C052:0</td>
<td>Select full screen</td>
<td>$800-BFF</td>
</tr>
<tr>
<td>C057:0</td>
<td>Select Hi-res page 2.</td>
<td>$4000-5FFFF</td>
</tr>
<tr>
<td>C054:0</td>
<td>Select Hi-res page 1</td>
<td>$2000-2FFFF</td>
</tr>
<tr>
<td>C053:0</td>
<td>Select mixed text and graphics page 1</td>
<td>$2000-2FFFF</td>
</tr>
<tr>
<td>C055:0</td>
<td>Select mixed text and graphics page 2</td>
<td>$4000-5FFFF, $400-7FF</td>
</tr>
</tbody>
</table>

What controls and decodes these little switches you’ve just been throwing is the IC F14 chip (labeled SN74259N). Each switch controls a different aspect of what is placed on the screen. That’s why a certain byte can show up as a flashing character if in the text mode, as colored blocks if the lo-res graphics switch has been thrown, or even as a series of dots if $C057 is accessed.

To the right is a diagram of the chip in question.

The integrated circuit (IC) gets its power through the two pins not shown; 8 and 16. By convention, in a 16-pin package the +5V (Vcc) connection goes to pin 16 (“HI”). Pin 8 is 0V, or ground (“LO”).

<table>
<thead>
<tr>
<th>Switch</th>
<th>3 A2 Z0 4 O</th>
</tr>
</thead>
<tbody>
<tr>
<td>select 2 A1 Z1 5 U</td>
<td></td>
</tr>
<tr>
<td>inputs 1 A0 Z2 6 T</td>
<td></td>
</tr>
<tr>
<td>Z3 7 P</td>
<td></td>
</tr>
<tr>
<td>CLR Z4 9 U</td>
<td></td>
</tr>
<tr>
<td>D Z5 10 T</td>
<td></td>
</tr>
<tr>
<td>Z6 11 S</td>
<td></td>
</tr>
<tr>
<td>E Z7 12</td>
<td></td>
</tr>
</tbody>
</table>

SN74LS259 (F14)
Notice the half-moon notch in Figure 2. It should point toward the keyboard.

One nice thing about working with logic circuits at such low voltages (0-5 volts) is that you can force certain lines low or high without any damage to the ICs, if you're careful.

**NOTE: VERY IMPORTANT. Don't connect pin 8 to pin 16. That would short out the power supply.**

As you may have noticed from following the screen-switching demo, you need to throw two or three switches to get to a certain point. With the chip disconnected, there's no circuitry to hold the switches "in position" so to speak, so you'll have to physically wire some pins HI and some LO. Needless to say, it could get rather hairy.

Because of this, I've figured out the correct combination for this application and soldered a jumper-socket (see figure 2) that I can quickly plug in to check if a particular program tries to use this protection technique. Disk Organizer II does.

![Figure 2](image)

This chart shows the results of the author's own experiments with the switch outputs.

<table>
<thead>
<tr>
<th>(+) = HI</th>
<th>(−) = LO</th>
<th>(0) = OPEN, no connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 5 6 7 9 10 11 12</td>
<td><strong>Effect</strong></td>
<td></td>
</tr>
<tr>
<td>+ 0 − 0 0 0 0 0 0</td>
<td>Text page 1</td>
<td></td>
</tr>
<tr>
<td>+ 0 0 0 0 0 0 0 0</td>
<td>Text page 2</td>
<td></td>
</tr>
<tr>
<td>0 − − − 0 0 0 0 0</td>
<td>Lo-res page 1</td>
<td></td>
</tr>
<tr>
<td>0 − 0 − 0 0 0 0 0</td>
<td>Lo-res page 2</td>
<td></td>
</tr>
<tr>
<td>0 0 − − 0 0 0 0 0</td>
<td>Mixed text &amp; Lo-res page 1</td>
<td></td>
</tr>
<tr>
<td>0 0 0 − − 0 0 0 0 0</td>
<td>Mixed text &amp; Lo-res page 2</td>
<td></td>
</tr>
<tr>
<td>0 − − 0 0 0 0 0 0</td>
<td>Hi-res page 1</td>
<td></td>
</tr>
<tr>
<td>0 − 0 0 0 0 0 0 0</td>
<td>Hi-res page 2</td>
<td></td>
</tr>
<tr>
<td>0 0 − 0 0 0 0 0 0</td>
<td>Mixed text &amp; Hi-res page 1</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0</td>
<td>Mixed text &amp; Hi-res page 2</td>
<td></td>
</tr>
</tbody>
</table>
Controlling The Soft Switches

1) Turn off the computer.

2) Carefully remove IC F14. (Remember: Without this decoder chip, any page flipping signals sent by the program (or ROM) to pins 1, 2, and 3 have no physical connection with the output pins 4-7 and 9-12. Therefore, you are free to throw your own.)

3) You may now turn on the computer and carefully experiment with pins 4-7 and 9-12, connecting some HI (to pin 16) or LO (to pin 8). Watch the results on your screen. When you want to continue, plug in an IC socket that has been wired as shown in Figure 2. Be sure it is oriented via the tab cutout toward the keyboard.

4) Once installed, boot the program in the usual way. Now convert the various screen characters back into hex code using a chart like the one in Hardcore Computing Update 2.1 (old series) or most Apple manuals. These jumpers will show you the hidden information you’ve been missing.

Epilogue

Just when you think you’ve got it beat, you always bump into another scheme, and this one has me stumped. Disk Organizer II has also cleverly hidden an important byte at $200. This is the first location in the input buffer, which is snuffed as soon as a key is typed. Any ideas?

Bibliography

Home Accountant
Continental Software

Requirements:
Home Accountant disk
One initialized disk
Apple’s FID program

By Barry May

For many months the Home Accountant has consistently ranked #1 on Softalk’s Home Top 10 list. This popular checkbook/home budget program has some very nice features, but it has some very annoying ones as well. Its three biggest faults are:

1) You cannot go back to a previous month to make an addition or correction;
2) You must wait an inordinately long time for the copyright notice and logo display to run through before you are presented with the opening menu, and
3) The program is constantly loading new modules and re-reading the data files resulting in very long waits between tasks.

Removing the copy protection allows at least two of the problems to be solved easily. The opening can be eliminated with a couple of simple changes (as shown below) and a fast DOS will speed up the disk I/O.

The protection on the Home Accountant is very simple. The address epilog has been changed from DE AA EB to DF AA EB. All that needs to be done is to change the read address routine to ignore the first byte of the epilog. This is done by changing byte $B993 (47507 in decimal) from an $AE to a $00. Now, instead of branching to the “Bad Read” routine, the computer merely branches to the next instruction, the one it would normally execute if everything was OK.

All that is left to do is to get the programs off the protected disk and onto one of yours. The easiest way to do this is to run a program that copies files using the DOS in the computer, like FID. Just copy the programs on the disk like you would if you were backing up programs from a normal disk.

1) Boot a System Master disk.
2) Change the branch

POKE 47507,0
3) Run the copy program

**BRUN FID**

Copy all the programs from the Home Accountant to an initialized disk. (Use a disk initialized with a fast DOS, if you want) and that's it!

After releasing the program from its protection, deleting lines 200 through 1110 from "Hello" and using Beagle Brothers' Pronto DOS, the time from start to menu drops from 37.7 seconds to 13.7 seconds. A fast DOS which speeds up textfiles (Diversi-DOS does this) will help even more.

Now it's up to someone else to write a routine for correcting previous months on the program.
Homeworld
Sierra On-Line, Inc.

Requirements:
Apple ][ 48K
One blank disk
CopyA from DOS master disk
Disk edit program

By Marco Hunter

Here is a quick softkey.

1) Copy with COPYA.
2) Edit track 10, sector 0A and change byte 09 from 49 to EA
   and byte 0A from C9 to 60.

That's it.
Lancaster
Silicon Valley Systems

Requirements:
48K Apple, with Applesoft
One disk drive and DOS 3.3
DOS 3.3 System Master
Lancaster
One blank disk

By Clay Harrell

Lancaster caught my eye as having unusually smooth animation and graphics. Being intrigued by the animation and playability of the game, I bought it with the intention of discovering the author’s methods of animation. But, in order to snoop through the code, it meant that I had to unprotect it first for disassembly.

The first thing to notice upon booting the game is that an Applesoft cursor appears at the bottom left of the screen. This means that the protection involves somewhat of a normal DOS and disk structure. Some protectors have begun to bypass the routine which outputs the prompt, but you can still guess that there’s a modified DOS present if the boot sounds like a normal DOS boot, but the disk won’t copy with COPYA.

To confirm my hunch that Lancaster was using a modified DOS, I booted one of my normal DOS 3.3 disks and put Lancaster in the drive and typed CATALOG. The disk drive recalibrated and made other obnoxious noises and returned the message I/O ERROR. Not that I was expecting any miracles, but why not try?

I still believed there was a somewhat normal DOS present on the disk, however, more snooping had to be done. Let’s think about what causes an I/O error, for a moment.

Whenever anything goes wrong during disk access, RWTS branches to a routine at $B942 to set the carry bit and return. The other routines in RWTS monitor the carry bit and check to see if there was a bad data read, a bad address read or some other no-no.

At $B942 there are simply two instructions: SET THE CARRY and RETURN. If we wish to defeat the DOS error checking (which we do in this case), we can change the SET THE CARRY to CLEAR THE CARRY. By making this change, you are telling RWTS not to check for any errors, assume everything is alright and go on.

Obviously, this is not good general practice since you are defeating the purpose of all the careful error checking that DOS does. But it is great for examining a modified DOS. It will
handle any changes to the epilog bytes or intentional errors in the checksum of either field, but not in the header bytes (header changes must be done by modifying the appropriate code in the subroutine).

With this in mind, we enter the monitor with CALL -151 and type B942:18 to disable the DOS error checking. Now type CATALOG and, gosh! Indeed, there is a catalog!

Now all the files are loadable (or BLOADABLE) for further snooping. But this is not the end of the protection.

Examining the HELLO program revealed an unusual file named SVS and some curious CALLs and POKEs. Upon further inspection, I came to the conclusion that the file SVS was a secondary protection involved in Lancaster. Simply preventing the loading of this file and disabling the calls to its routines was all that was really needed in the deprotection of Lancaster.

The following steps recap the procedures necessary in the deprotection of Lancaster:

1) Boot a normal DOS 3.3 disk and initialize a blank disk with the command

   INIT HELLO

2) Type

   CALL -151

to enter the monitor and then

   B942:18

to disable the DOS error checking routine.

3) Insert your DOS 3.3 System Master in a drive and run the program FID with the command

   BRUN FID

4) Copy all the files from the Lancaster disk to the blank initialized disk you just prepared.

5) Boot your DOS 3.3 System Master and put your newly created Lancaster disk in a drive.

6) Delete the Hello program from your Lancaster clone disk

   DELETE HELLO

7) Unlock the file Lancaster with the command

   UNLOCK LANCASTER

8) Rename the file Lancaster to Hello with the command

   RENAME LANCASTER,HELLO

Lancaster is now unprotected and all the code can be examined for educational and modification purposes.
Magic Window II

ARTSCI, Inc.

Requirements:
Apple II with 48K
One disk drive
One initialized blank disk

By Bobby

Magic Window II is an updated version of the old Magic Window word processor. Many new features have been added including paragraph gluing and search and replace functions. Unfortunately, a few bugs have also crept into the program and, in an effort to fix these bugs, I had to unlock the disk. Although the original program disk can be catalogued and files can be loaded and saved to it, the actual word processor is protected and does not appear on the disk catalog.

I discovered that there are four separate Magic Window programs stored as consecutive sectors of data. The four versions are:

1) 40/80 (columns) without a RAM card
2) 40/80 with a RAM card
3) 40/70/80 without a RAM card
4) 40/70/80 with a RAM card

After examining the file "BRUN MW II" I was able to determine which sectors each of the four versions were on. The boot program (BRUN MW II) first checks to see if a RAM card is present, and then loads the proper version of the program (40/80 or 40/70/80 columns). Writing a BASIC program to duplicate this function was easily done, but I still needed the actual programs from the disk so that the BASIC program could load them.

The easiest way to do this was to use the same program that Magic Window II uses to read in each of the four files (BRUN MW II). What follows is a step-by-step procedure for getting the proper routine into the computer.

The Hello Program

1) Boot the DOS 3.3 System Master

PR#6

2) Remove the Master disk and insert a blank diskette.
3) Clear the program in memory.

FP
4) Enter the Applesoft Hello program in Listing 1.
5) Initialize the disk with the program HELLO

```
INIT HELLO
```

6) Remove this disk. This will be your new Magic Window diskette.

Now we are going to load each of the four versions of Magic Window from the old Magic Window disk and save them onto the new disk. Place a write-protect tab on the original so you don’t accidentally alter the disk.

**Copying The Disk**

7) Boot the original Magic Window II disk. When the prompt appears (asking which version to load), press RESET. This bypasses all of the protection on the disk and loads in the main controller routine intact.
8) Now enter the monitor

```
CALL -151
```

(If you get a “OUT OF MEMORY” error, repeat step 8.) Since there are four versions of the program on disk, each of these must be loaded and saved separately.
9) Type

```
18:04 00 3A 0A
80F6G
```

This information tells the subroutine on which track and sector to start, the number of pages to read, and where to place the data.
10) After the disk stops spinning, place the blank disk in the drive and save the file (remember to do this from the monitor)

```
BSAVE MW II 1, A$A00, L$3A00
```

The other three files should be saved in the same manner. Don’t forget to put the blank disk back in the drive before saving each file.
11) Insert original disk and load the next file

```
18:12 00 5A 0A
80F6G
```

12) Insert backup disk and save

```
BSAVE MW II 2, A$A00, L$5A00
```

13) Insert original disk

```
18:08 00 3B 09
80F6G
```

14) Insert backup disk
BSAVE MW II 1/WITH RAM, A$900, L$3B00

15) Insert original disk

18:OC 00 3B 09
80F6G

16) Insert backup disk

BSAVE MW II 2/WITH RAM, A$900, L$3B00

17) The file SYS.OPTIONS can now be loaded from the Magic Window disk and placed on the backup. First, insert the original disk and

BLOAD SYS.OPTIONS

18) Next, insert the backup disk

BSAVE SYS.OPTIONS, A$ABD, L$D

The other Magic Window disk can be copied with COPYA from the System Master onto a blank disk.

Modifications To The Hello Program

The HELLO program allows you to select which version of Magic Window you wish to use. The program first POKEs a small machine language routine into page 3 of memory. This routine checks for a RAM card and sets certain flags depending on whether or not one was found. After this is completed you will be presented with two choices exactly like those you saw on the Magic Window II disk. The BASIC program operates in a manner similar to the original machine language program that was found on that disk.

Since each of the four files can stand alone, the HELLO program can be bypassed and the correct version of Magic Window can be BRUN directly. A program allowing you to ignore the first question and immediately skip to the proper version would consist of only one line:

10 PRINT CHR$(4) "BRUN version of Magic Window"

The following chart will allow you to choose the proper version to run:

<table>
<thead>
<tr>
<th></th>
<th>40/80</th>
<th>40/70/80</th>
</tr>
</thead>
<tbody>
<tr>
<td>With RAM</td>
<td>MN2 1/WITH RAM</td>
<td>MN2 2/WITH RAM</td>
</tr>
<tr>
<td>Without RAM</td>
<td>MN2 1</td>
<td>MN2 2</td>
</tr>
</tbody>
</table>
The Technique

The unlocking technique for Magic Window II can be used with some other software on the market. ARTSCI only protected two of the sectors on the Magic Window II disk which contained part of the loader required to load the main Magic Window menu. I simply traced the file BRUN MW II to see what it did and what other sectors it loaded into memory. The four Magic Window files could then be loaded by calling a routine that started on a given track/sector and loaded the proper number of sectors into memory, placing them at a given location. By following the previous set of directions, you told the Magic Window menu where each file was by changing locations 18, 19, 1A and 1B:

18: First track of data
19: First sector of data (always 00)
1A: Number of sectors to load
1B: The high byte of the buffer (low byte is always 00)

HELLO program listing

10 DS = CHR$(4)
20 NORMAL : TEXT : HOME
30 PRINT "MAGIC WINDOW II"
40 PRINT : PRINT
50 PRINT "PLEASE SELECT VERSION:"
60 PRINT
70 PRINT "1-40/80 COLUMNS (MORE FREE SPACE)"
80 PRINT
90 PRINT "2-40/70/80 COLUMNS (LESS FREE SPACE)"
100 PRINT : PRINT
110 PRINT "YOUR SELECTION?" ; : GET A$$
120 A$$ = VAL(A$$) : IF A < 1 OR A > 2 THEN PRINT CHR$(7) : VTAB 11 : GOTO 110
140 A$$ = "" : GOSUB 180 : CALL 768 : IF PEEK(0) THEN A$$ = "/WITH RAM"
150 HOME : VTAB 12 : HTAB 10
160 PRINT "LOADING MW II" A$$ A$$
170 PRINT DS "BRUN MW II" A$$ A$$ : END
180 FOR X = 0 TO 29 : READ B : POKE 768 + X , B : NEXT
190 RETURN
200 DATA 160,0,132,0,173,131,192,173,131,192,152,141,
0,208,205,0,208,208,208,7,200,208,244,169,1,133,0,173,129,192,96
Multiplan

Microsoft Corp.

Requirements:
Apple II/ 48K
DOS 3.3 Master disk
COPYA (from Master disk)
A disk edit program

By Bobby

Multiplan is an excellent spreadsheet program by Microsoft. It includes an unusually complete manual with a reference guide, and an auto-help mode from within the program. Multiplan allows one and only one backup to be made, which I found to be an insufficient guarantee of non-loss of data (three is my minimum backup policy for commercial software).

The Protection

The program is only protected on tracks zero through four. The protection scheme is to change the end of address mark on those tracks from $DE to a strange value. To allow the Multiplan DOS to read the unprotected disk, a modification must be done to track $00, sector $0A. Byte $0D must be changed to hold the value of $DE.

The Deprotection

1) Boot from the DOS Master disk.

PR#6

2) Fix the check for $DE in DOS.

POKE 47507,0

3) Run COPYA and follow the prompts to copy the Multiplan disk.

RUN COPYA

4) Use your disk edit program to change byte $0D on track $00, sector $0A from whatever it is to $DE.

The copy of Multiplan can now be duplicated with COPYA, or any number of other copy programs. Enjoy!

NOTE: The copy disk option (1) on the utility menu will make copies of this disk.
Pest Patrol
Sierra On-Line Inc.

Requirements:
Apple ][ with 48K
Pest Patrol disk
One initialized slave disk with HELLO program deleted
One disk drive
Some knowledge of machine language

By Ray Darrah

Pest Patrol is an outerspace shooting gallery with many diverse levels, each employing its own enemy attack patterns. Built-in options help configure the game to the player’s machine and ability level. For example, Pest Patrol may be played with the keyboard, paddles, rheostatic joystick, or Atari joystick. Although each game is somewhat different, they are all fun.

Unfortunately, the protection scheme used on Pest Patrol is such that it will continually reboot on a computer with a language card. That just about washes out all the Apple //e and Franklin Ace users.

Never fear, for Hardcore COMPUTIST has a solution to both the backup problem and the language card problem: convert Pest Patrol into a normal binary file. This will omit the booting sequence where the check for the language card resides (and the reboot subroutine). Once this is done, Pest Patrol will work on an Apple //e or Franklin Ace just as if the computer was an Apple ][ without a language card.

Where To Begin?

The first step I took in breaking Pest Patrol was to check for simple prologue or epilogue alterations. If these alterations were the problem, I easily could have made a softkey to do the job of backup and my problem would have been solved (although the program still wouldn’t run on a computer with a language card). But there were no alterations.

I noticed that the data on this disk was unlike normal data stored by DOS, so I decided to boot code trace the program.

Boot Code Tracing: The Concept

To boot a disk, the computer must be able to load track 0, sector 0. This is where the first in a sequence of programs responsible for loading the main program into memory is written.
The boot code trace disk-breaking method depends on the fact that track 0, sector 0 must always be loaded for any disk to boot. It works by tracing the steps which the computer follows during the entire process of booting a disk. First a small program in the disk controller card loads a 256-byte program stored on the disk’s track 0, sector 0. This program is loaded into memory beginning at $0800 and is responsible for loading the next program in the boot process. There may be several of these boot programs (each usually longer than the one before it) leading up to the actual loading of the main program stored on the disk. While tracing, this process is halted to examine each program before executing.

The second short program (on track 0, sector 0 of the Pest Patrol disk) immediately loads a third, larger program into memory. This third program checks for a language card and, if none is present, loads the main part of the game program. If a language card is discovered, the computer is instructed to reboot endlessly.

How To Boot Code Trace

This article is an account of how I boot code traced Pest Patrol. Since the text follows the order of my actions, a complete list of steps for copying the disk is not found until near the conclusion. This organization will help those trying to learn boot code tracing.

Refer to the procedure listed under “The Whole Thing” for a complete set of instructions for copying Pest Patrol.

Beginning A Boot-Code Trace

I started with the usual boot tracing preliminary steps:

1) Turn on your Apple (or Apple-compatible).
2) Press RESET before the computer has a chance to boot.
3) Enter the monitor
   CALL -151
4) Put zeroes in all memory locations from $0800 to $BFFF, inclusive
   800:00 N 801<800.BFFF

Placing zeroes in all RAM higher than $07FF makes it easier to discover the location in memory at which the programs load. Look for locations where the zeroes have been replaced by other code; a program has been loaded there.
5) Move the boot code from $C600 (slot 6) to $9600

**9600>C600.C6F7M**

Only the part of the boot code responsible for loading track 0, sector 0 into $0800 is transferred. The move command is halted just before the JMP to $0801 (contained in the controller card) by indicating location $C6F7, instead of the normal $C6FF, which would have included the JMP command. Since the memory has been zeroed, the boot process is halted by a BRK instead of a JMP at location $CF68 which occurs right after loading the sector. This results in the partial boot of Step 7.

6) Insert the Pest Patrol disk.

7) Execute the partial boot

**9600G**

After completing this last step of the beginning boot procedure, the computer will beep and display the message:

96FA-A=01 X=60 Y=00 P=31 S=F0

The disk drive will keep spinning. That is to be expected because the program has been halted at an early stage due to the partial boot. You must let it continue to spin while performing the boot code trace, but opening the drive door will prevent wear on the disk.

At this point, the boot process has been halted just before executing location $0801 in memory where the short boot program on track 0, sector 0 always is loaded.

Now begins the dirty work: examining the machine language code starting at $0801 ($0800 holds the total number of consecutive sectors to be read) to locate where the next stage of the boot process resides.

**Searching The Machine Code**

At first glance, the Pest Patrol machine code looked like a valid program. However, upon closer examination, I found many things that didn't look right. For example, statements such as these:

0809- 90 78 \hspace{1cm} BCC $0883
080B- D0 01 \hspace{1cm} BNE $080E
080D- AD 20 9C \hspace{1cm} LDA $9C20
0810- 08 \hspace{1cm} PHP
0811- A0 3F \hspace{1cm} LDY #$3F
Hidden Commands

I noticed the LDA $9C20 followed by a PHP and thought, "Why would anyone care what was in location $9C20?" This is what I call an irrational command. Then I saw the preceding BNE which branched to the middle of the LDA command (20) rather than to the beginning (AD). This tipped me that the BNE might always be taken (skipping the first byte in this manner). Sure enough, when the code was disassembled and the confusing byte at $080D excluded, a hidden rational command was revealed at $080E.

```
0809- 90 78   BCC $0883
080B- D0 01   BNE $080E
080E- 20 9C 08  JSR $089C
0811- A0 3F   LDY #$3F
```

This made me dread looking at more code. What if I missed a hidden command? How long would it take to find them all?

Well, it wasn't too long before I stumbled across this wondrous piece of machine language.

```
085E- 8C 0B AA STY $AA0B
0861- 10 01   BPL $0864
0863- 4C 20 D8 JMP $D820
0866- B6 AD   LDX $AD,Y
0868- 08   PHP
0869- 03   ???
086A- F0 03   BEQ $086F
086C- D0 15   BNE $0883
```

The first thing I noticed was the ???. Whenever I see a ?? surrounded by what appears to be irrational code, I immediately think it could be a data table of some kind. But this one looked like it was right in the middle of rational code. Stepping backward, I saw JMP $D820. This and the two following statements certainly looked fishy. Then I found it: a "branch on result plus" (BPL) to the second byte in the jump instruction (20). This is what it looks like when the byte at $0863 is eliminated.

```
085E- 8C 0B AA STY $AA0B
0861- 10 01   BPL $0864

0864- 20 D8 B6 JSR $B6D8
0867- AD 08 03 LDA $0308
086A- F0 03   BEQ $086F
086C- D0 15   BNE $0883
```

Once again a hidden rational command was revealed, this one at $0864. Finding these wasn't easy, but it was worth the effort.
The other byte inserted to confuse the issue was at $086E (right after the preceding example). The best way to find these hidden commands is to look at the branches and other flow-related commands in the program. Spotting these only becomes easier with practice.

There are also two 11-byte data tables starting at $0881 and $089A. Data tables are much easier to find because they are usually referenced by another part of the program. The only tricky part is trying to determine their lengths (but this isn’t as tricky as you might think).

Five Subroutines

After spending quite some time scrutinizing this mad program, I concluded that it was comprised of five subroutines. The backbone of the program is the subroutine starting at $089C which loads three sectors into memory starting at $B500. Other subroutines include a translate-table builder at $0817, a routine to get one byte of data from the disk starting at $0872, and a reboot subroutine starting at $0883. This second program has a somewhat obvious exit to $B800 at $086F. The next step was to alter the program to stop just short of exiting.

Stopping Before The Exit

To make the sector safe to execute, I typed

86F:00

I shut the drive door, crossed my fingers, and typed

801G

(It was very hard to type this with my fingers crossed.) The disk made a strange noise, and the computer responded with a beep and the message:

0871-A=00 X=0B Y=FF P=33 S=EE

The boot process again was halted. I then had a very large program in memory (many of the higher addresses no longer contained zeroes) with an entry point of $B800.

This was the third boot program, which contained the language card check. The disk was still spinning, so I once again opened the drive door to prevent unnecessary wear and tear on my expensive Pest Patrol disk. Yes, more code tracing was ahead!

I knew I was getting closer to having the entire Pest Patrol program in memory because the number of hidden commands steadily increased. There were too many to list here but, if you’re interested, you’ll be able to find them.
Careful tracing of the program starting at $B800 revealed that it decodes a lot of memory and moves it into its proper location. It then exits at $B8A4 if you have a language card or $B8A7 if everything is okay (it also clears the text screen). From $B8A7, the program was supposed to go to $B2E0, so I placed two more breaks by typing

```
B8A4:00
B8A7:00
```

I then executed the modified program with

```
B800G
```

The screen cleared and so did my mind. I didn’t feel like tracing the code starting at $B2E0, so I listed it until I found an exit. After a few screens, I found this:

```
8375- A9 B4
8377- 48
8378- A9 BD
837A- 48
837B- 4C 7A B4
```

LOA #$84
PHA
LOA #$80
PHA
JMP $847A

It looked like the programmer who wrote Pest Patrol wished to execute a subroutine at $B47A and then intended program execution to continue at location $B4BE.

My hunch was correct. When I shut the drive door and typed the following (after some funny disk noises followed by a beep) the disk drive stopped. This is what I typed

```
B375:00
B2E0G
```

(wait for drive to stop)

```
B47AG
```

This was it. I knew that one of two things had happened. Either I had the entire Pest Patrol program in memory or things were messed up pretty badly.

I then remembered the code starting at location $B4BE. Some quick listings revealed (among the hidden statements and other sneaky stuff) that this program did a large amount of memory manipulation. After making this discovery, I found the equivalent of a JMP to location $0800 at $B466. Trace this one for yourself. It’s a nightmare!

Luckily, I found no access to the disk in this subroutine which was a load off my mind. You only live once, was my only thought as I typed

```
B466:00
B4BEG
```
Once again the monitor awaited my next command. This was the big moment. Was there going to be valid code at $0800 or was there an error in my painful tracing? I typed

800L

I was amazed at the absence of hidden commands. Instead, I found a little routine to set all the vectors at the end of page 3 to $4000. This was followed by a number of STAs to consecutive locations starting at $0000. The program then JSRs to that location. This is followed by the usual strange stuff (messing around with pointers and the like).

Finally, after breaking the program in several places and examining various locations, I surmised that the main part of this program moves $0900-$8700 into $4000-$BE00. Then it jumps to location $4003. If you wish to find this out for yourself, it is best to NOP both of the STA $03F0,Y commands. Otherwise, the BRK vector, as well as the reset vector, will be overwritten.

I observed that an 800G at this stage would start Pest Patrol. Unfortunately, once executed, I couldn't escape from it. I decided to follow my notes from the beginning to the point where I typed 800G.

The Whole Thing

Assuming the disk controller is in slot 6, the following is a brief overview of my procedure.

1) Turn on your Apple (or Apple-compatible) without a disk in the drive.
2) Hit RESET.
3) Enter the monitor
   CALL -151
4) Put zeroes in all memory locations from $0800 to $BFFF, inclusive
   800:00 N 801<800.BFFFM
5) Move the bootcode from $C600 (slot 6) to $9600
   9600<C600.C6F7M
6) Insert the Pest Patrol disk in the drive.
7) Execute the partial boot
   9600G

Don't press RESET to stop the spinning of the drive; let it turn as you complete the remainder of the procedure.
8) Type the following

86F:00
801G
B8A4:00
B8A7:00
B800G
B375:00
B2E0G
B466:00
B4BEG

After reviewing the preceding steps, I made two modifications.

9) Type

805:A9 00 8D
808:F2 03 A9 E0 8D F3 03 49
810:A5 8D F4 03 D0 0D
8DC:4C 00 40

These modifications enable reset to stop the program (you will be without DOS). They also relieve the program of its boring title page, which lasts about 20 seconds too long. In addition, they eliminate a little routine which performed some memory verification, printing "CHECKSUM ERROR" and making an awful noise if something was wrong.

All that was left was to save this modified version. A few seconds of thought and I had it. I decided to boot with a 48K slave disk (saving page $8 first, of course) and then restore page $8 and BSAVE the file.

10) Save page $8 on page $96

9600<800.8FFM

11) Insert a 48K slave disk WITH NO HELLO PROGRAM. Make sure this is a slave disk (using a master will wipe out the code).

12) Boot the disk

C600G

13) Return to the monitor (if it doesn’t work the first time, try again)

CALL -151

14) Move page 8 from page 96 back to its original location

800<9600.96FFM

15) Insert the initialized diskette on which you wish to have the Pest Patrol backup (the game uses 131 sectors, so it should be a relatively empty one).
16) BSAVE the entire program (it takes about 42 seconds)

BSAVE PEST PATROL, $800, L$7FFF

17) You now can BRUN Pest Patrol after booting normal DOS.

A Confession

To be honest with you, this was my first attempt at breaking a copy-protected disk. I found it to be much easier than I had anticipated. The entire job took me only about 15 hours. I am sure the process will take less time as I become more experienced. Well, have fun with your backup of Pest Patrol. I suggest storing the original Pest Patrol game and all of your other original disks in a dark, cool place.
Prisoner II

Eduware

Requirements:
Apple ][ 48K
DOS master disk
COPYA program
One blank disk

By David Kirsch

Prisoner II uses standard DOS for tracks $00-$34. The game also uses track $35, which contains special copy protection data, none of which is needed to run the program.

Here's how to get rid of the track $35 access.

Making A Copy

1) Boot from the DOS Master disk and use COPYA to copy the disk.

PR#6
RUN COPYA

2) Get rid of the track $35 check.

UNLOCK IF.SHAPE
BLOAD IF.SHAPE
CALL-151
57B4:BD 8C
BSAVE IF.SHAPE
LOCK IF.SHAPE
3DOG

That's it. You now have an unprotected backup copy.
A Fix For RANA Drive Owners

By Joseph W. Leathlean

I have a solution for all Rana drive controller owners who wish to do Boot Code Tracing.

While the Rana controller's ROM code is incompatible with the standard ROM and controller, the I/O addresses are supposed to be compatible since they will work with DOS 3.3. So, all Rana owners have to do is to borrow some time on a computer with a standard controller and do the following:

1) Boot a diskette with normal DOS 3.3
   
   **PR#6**

2) Save the code from the Apple controller card to a disk.
   
   **BSAVE CONTROL ROM, A$600,L$100**

3) When they want to do the boot code trace, instead of moving the controller ROM routine to $x600, just BLOAD the file CONTROL ROM at the address needed. They should be able to follow the boot code tracing procedures with no problem.
Sammy Lightfoot
Sierra On-Line, Inc.

Requirements:
Apple II with 48K
One disk drive
One blank disk
Sammy Lightfoot Diskette
DOS 3.3 System Master with COPYA
Any disk editor (such as DiskEdit)

By Eric Kinney

Sammy Lightfoot is a running/jumping/climbing type of game which is fun to play, has high quality graphics, and is easy to backup as you will soon see.

The game has three "scenes" with six levels of difficulty for each scene. Sammy, the hero of the game, is trying out for a circus act in which he bounces on trampolines, dodges giant circus balls and uses ropes to swing over flames and certain death below.

The copy-protection used seems to be a check of track 0 prior to each new scene. Several things were done to hide the code in memory. With effort, however, it can be traced and tested at various points to find where it actually checks for copy-protection.

My first thought was that it was checking for a nibble-count, but since copying track 0 with Locksmith's nibble counter didn't copy it, I suspect that it checks for something else. By tracing the machine language code and disabling various subroutines until I homed in on the right one, I discovered a place where the copy-protection could be circumvented. This was at location $989B where it does a JSR to $9E00. In assembly code, that's 20 00 9E.

I changed the bytes to EA, which is assembly code for NOP, or "No OPeration". Since the bytes I changed were 20 00 9E, it was not too difficult to scan the disk with DiskEdit until I found these three bytes, and changed them permanently. Making a backup of Sammy Lightfoot is very simple:

1) Boot up with DOS 3.3 System Master

RUN COPYA

2) Copy Sammy Lightfoot with COPYA.
3) Boot up a Disk Editor, such as DiskEdit.
4) Use your sector editor to make the following changes to the copy of Sammy Lightfoot.
You now have a working backup copy.

**Unlimited Sammys**

1) Using a sector editor, make the following changes to the disk and write the sectors back out:

<table>
<thead>
<tr>
<th>Trk</th>
<th>Sect</th>
<th>Byte</th>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>0D</td>
<td>00</td>
<td>9B</td>
<td>20</td>
<td>EA</td>
</tr>
<tr>
<td>0D</td>
<td>00</td>
<td>9C</td>
<td>00</td>
<td>EA</td>
</tr>
<tr>
<td>0D</td>
<td>00</td>
<td>9D</td>
<td>9E</td>
<td>EA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trk</th>
<th>Sect</th>
<th>Byte</th>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>0C</td>
<td>03</td>
<td>69</td>
<td>CE</td>
<td>EA</td>
</tr>
<tr>
<td>0C</td>
<td>03</td>
<td>6A</td>
<td>4F</td>
<td>EA</td>
</tr>
<tr>
<td>0C</td>
<td>03</td>
<td>6B</td>
<td>73</td>
<td>EA</td>
</tr>
<tr>
<td>0C</td>
<td>03</td>
<td>73</td>
<td>CE</td>
<td>EA</td>
</tr>
<tr>
<td>0C</td>
<td>03</td>
<td>74</td>
<td>4E</td>
<td>EA</td>
</tr>
<tr>
<td>0C</td>
<td>03</td>
<td>75</td>
<td>73</td>
<td>EA</td>
</tr>
<tr>
<td>10</td>
<td>0B</td>
<td>81</td>
<td>CF</td>
<td>EA</td>
</tr>
<tr>
<td>10</td>
<td>0B</td>
<td>82</td>
<td>4F</td>
<td>EA</td>
</tr>
<tr>
<td>10</td>
<td>0B</td>
<td>83</td>
<td>73</td>
<td>EA</td>
</tr>
<tr>
<td>10</td>
<td>0B</td>
<td>8B</td>
<td>CE</td>
<td>EA</td>
</tr>
<tr>
<td>10</td>
<td>0B</td>
<td>8C</td>
<td>4E</td>
<td>EA</td>
</tr>
<tr>
<td>10</td>
<td>0B</td>
<td>8D</td>
<td>73</td>
<td>EA</td>
</tr>
</tbody>
</table>

**APT For Old Monitor ROM**

When the game has begun play, hit RESET to get into the monitor. If you want to alter the playing level and/or the scene, use the following procedure once you are in the monitor:

1) Enter the level of play (0-B) at location $36B.
2) Enter the scene (0-3) at location $94E3.
3) Restart the game by typing

96C8G
The Screenwriter II word processor is a powerful writing tool, combining many advanced features with ease of use. The only problem is that you can’t back it up. I found this particularly upsetting because the program is stored as a series of binary files on an almost standard DOS 3.3 diskette. This means that the diskette can be copied with FID or COPYA but the data that tells the copy protection routine that the diskette is an original is lost (the information is coded into the formatting of the diskette) and the program won’t run. Fortunately, the technique to unlock this disk is very simple.

What we will do in this procedure is remove a machine language JSR (Jump to SubRoutine) instruction and bypass a particularly nasty subroutine which checks the disk to see if it is an original. If this routine found that the disk was a copy, it would clear the memory and reboot.

Step One: Make a backup of the diskette with either FID or COPYA and hide the original! I can’t stress the importance of this enough. It is too easy to make a fatal mistake and have your only copy destroyed.

IF YOU USE FID: Boot the original disk. When the main menu appears, press ©C to enter BASIC. Remove your master diskette and insert your backup. Type the following

NEW
INIT START
DELETE START

Now use FID to copy all the programs which are on the master diskette onto the backup.

IF YOU USE COPYA: Just boot any DOS 3.3 diskette

RUN COPYA

and follow its directions.
Step Two: We will now make the actual changes to the program. These changes consist of a three-byte patch to two files on the diskette. Check to make sure your master diskette is hidden (just remember how much this program cost!) Now, with the backup in the drive, do the following

1) Enter the monitor
   CALL -151
2) Load the first file
   BLOAD RPART1
3) Make the first patch
   1F90:EA EA EA
4) Save the changes
   BSAVE RPART1,A$C00,L$1400
5) Load the second file
   BLOAD EDITORPART1.OBJ0
6) Make the second patch
   1F49:EA EA EA
7) Save these changes
   BSAVE EDITORPART1.OBJ0, A$C00, L$1400

The Screenwriter II is now unlocked and can be backed-up with COPY A or FID as many times as you like without needing any further changes.

If you use a different DOS, you must arrange to BRUN the file named START upon booting. This may be accomplished by creating a HELLO file to do it or by patching DOS directly.

The procedure to patch DOS 3.3 to BRUN a binary file upon booting is:
1) Boot a DOS 3.3 diskette.
2) Then type
   POKE 40514,52
Any diskette INITed with this DOS will BRUN whatever file you specified in the INIT command.

A note to those who own Quality Software’s “Bag of Tricks”: The INIT program’s reskew function can be used to greatly increase the Screenwriter II’s efficiency in loading, saving and packing files. Reskew the Screenwriter II program diskette (your backup!) tracks 3-22 to 9 DESCENDING and the TARGET and TEXT diskettes tracks 0-22 to 6 DESCENDING.
Sneakers

Sirius Software

Requirements:
48K Apple with Applesoft in ROM
One disk drive
Snapshot Card
One blank disk

By David E. Rentzel

I used Snapshot to make a non-protected file of Sneakers. The problem is that during portions of the running game, the disk is accessed via copy-protected data to verify the original disk’s presence.

This can be defeated by making two simple monitor changes:

4FE1:60
94D3:60

The program can now be saved and run without further disk access.
Spy’s Demise

Penguin Software

Requirements:
48K Apple II Plus or equivalent
Locksmith v4.1 & Nibbles Away II
Spy’s Demise

By Peter M. Anker

I recently tried to backup a copy of Spy’s Demise by Penguin Software according to the instructions given under the Copy II+ parameter list. For the disk I have, these parameters would not work properly. I would get only the title picture, but no game.

After some experimentation with other programs, I found that the disk would copy easily by using Locksmith 4.1 for tracks 0 to 12 (error 2 on track 12 is OK) and Nibbles Away II on tracks 1, 5 and 7. Locksmith was apparently not able to copy those tracks. No parameter changes were required for either copy program and it was not necessary to copy any other tracks.
Starcross
Infocom, Inc.

Requirements:
Apple ][ 48K
DOS master disk
COPYA program
Sector Editing program
One blank disk

By Jeff Rivett

Having just completed Starcross, I can say with certainty that it is one of the finest text adventures I have ever played. The puzzles are very logical and, although some are quite difficult, they can still be mastered by pure reasoning. In other words, you don’t have to rely on luck to win the game.

You don’t have to rely on luck to make a successful copy, either. The entire game program uses only tracks $00 through $18 (0-24) and track $00 is not protected. The protection scheme on the remainder of the disk is to change the start-of-data marks, normally D5 AA AD, to D5 AA BC.

Making A Copy Of Starcross

1) Boot from the DOS Master disk.

PR#6

2) Fix the check for $AD in DOS.

POKE 47358,0

3) Run COPYA and follow the prompts to copy the Starcross disk.

RUN COPYA

4) Use your disk edit program to change the following bytes;

<table>
<thead>
<tr>
<th>Trk</th>
<th>Sct</th>
<th>Byte</th>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>FC</td>
<td>BC</td>
<td>AD</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>5D</td>
<td>BC</td>
<td>AD</td>
</tr>
</tbody>
</table>

The first modification allows the program to read the copied disk and the second allows the save game routine to write to a normal 3.3 disk.
You have deprotected Starcross. This copied version of Starcross can now be booted from slot 6.

Adding Your Own Text

Infocom programs don't use normal text files for program text. Instead, binary information is read directly off the disk into memory where some strange and wonderful alterations are performed to make it look like text to you and me. When a sector editing program is used to look at the copy disk, only the error message and the SAVE and RESTORE prompts are visible. If the text were decoded, it would be possible to add your own messages and personalize your copy.

Unfortunately, this isn't easy. After experimenting with my Starcross IOB copy for a while, I realized that the coding is probably not just a straight byte-for-ASCII-byte mapping. In fact, some values may represent whole words. I also suspect that a checksum is used on all text data because even the slightest change can cause the program to bomb. Although you may find that decoding the text in Starcross is quite a challenge, you now have the peace of mind of a backup copy on which to practice.
No List Programs

If you use DOS and would like to baffle your friends or protect your program listings from casual prying, then type the following line exactly as it is written into one of your programs. When you get to the ! type in a 8D. (The 8D is entered by holding the 8 key down and pressing the D key. The D should not print.)

The line on your screen should look like this:

0 REM IT'S NO FAIR IF YOU PEEK!FP

Save the program before you list it. When you LIST the program it should look like this:

0 REM IT'S NO FAIR IF YOU PEEK!

And that is all you will get. If you try to LIST the line again, it's not there.

If you count the characters from the 0 to the ! you'll get 33. Applesoft tries to LIST programs using 33 columns instead of the full 40. The 34th character is folded over and printed on the next line (There are exceptions). DOS gets control at the 34th character when fold over occurs and normally passes control back to Applesoft. However, if the 34th character is a 8D, then DOS thinks that it has been given a command and will process the remainder of the line accordingly. The FP at the end of the remark tells DOS to reset the Applesoft program pointers and has a similar effect as the NEW command in Applesoft. You can replace the FP with any other DOS command. How about CATALOG?

Hidden Lines

Hiding a line or changing the visible portion is another neat trick. To do this, type in the following steps exactly as shown (press return after each step):

NEW
1REM12345672 REM HELLO!
5 A = PEEK(103) + PEEK (104) * 256 + 5
10 FOR X = 0 TO 6: POKE A + X, 8: NEXT
LIST
RUN
LIST

Notice anything different?
The REMark in line #1 has been overwritten by the second half of the REMark making it appear to be line #2. Line #5 PEEKs the start of program pointer and adds an offset to it. Line #10 changes the numbers 1 through 7 in the REMark into backspaces. The result is the apparently changed REMark. A line could be completely buried using this technique. Important GOSUBs and GOTOs could be disguised as REMarks. A second Copyright notice could be hidden this way. The list is endless. (Be sure to reset SPEED to 255, afterwards).

A Bomb

Zero page location 214 ($D6) is the run flag for Applesoft. If the number stored here is greater than 127 ($80) then the program in memory will AUTO-RUN each time you try to issue a command. In order to LIST the program or change a program line, the number in location 214 would have to be changed to a value smaller than 128. If you were to insert the following lines into your program it would be difficult for the uninformed to tamper with or change the program:

2 POKE214,255
3 IF PEEK(214) <> 255 THEN NEW

Line #3 should be inserted in the program in several different places (with appropriate line numbers).

Locking Your Program Into The Run Mode

This technique is often used to prevent unauthorized tampering. It’s a neat trick to play on a friend and can be done by inserting the following line into the beginning of a program:

0 POKE 216,0: POKE 214,128: POKE1010,102: POKE1011,213: POKE1012,112: ONERR GOTO 0

Line #0 sets the RUN flag, changes the RESET vector to point to the RUN command in Applesoft and locks out the C. Now the program will restart each time you hit RESET or C.

NOTE: This will only work on an Apple with the autostart ROM.
Ultima II
Sierra On-Line

Requirements:
48K Apple II Plus or //e
One disk drive with DOS 3.3
Ultima II: Program Master, Player Master, and Galactic disk
COPYA or similar disk copy program
Three blank disks

By Brian Burns & Dan Rosenberg

Owners of Ultima II may know how hard it is to backup. The copy-protection is tough to break because the data is stored differently than on normal DOS disks. Unlocking disks like Ultima is frustrating, mostly because it is often nearly impossible. Fortunately, there are shortcuts.

Because Sierra On-Line left a big hole in the copy protection of this adventure game, the disks are COPYAble with only slight modification to DOS. Programs like Locksmith and Nibbles Away usually have a hard time copying Ultima II. But they will do the job if you prevent DOS from reading the VTOC's from the disks. The VTOC is a sector on every normal DOS disk that tells on which track and sector the catalog starts (the catalog contains all the file names on the disk), and which version of DOS is on disk (3.2 or 3.3). It also contains a table that tells which sectors are being used to store programs and which are empty.

The VTOC's on the Ultima II disks have been filled with hex $FF's, which is why DOS gives an I/O error (it thinks the catalog starts at track $FF, sector $FF). You do know, of course, that track $FF, sector $FF doesn't exist, don't you? Ultima II doesn't get errors when it is reading from its own disks because its Disk Operating System is modified and doesn't need the VTOC to load programs.

Following the softkey, there is an Advanced Playing Technique for Ultima II which allows you to change a character's strength, wisdom, armor, weapons, race, hit points, etc. in the middle of the game. First complete the Softkey, because it modifies the program so it can be used with the APT.

How to Copy

1) Boot your system master or any regular DOS 3.3 disk.
2) Enter the monitor

CALL -151
3) Type

`AFF7G`

This allows the reading of the VTOC from the normal DOS 3.3 disk into memory.

4) After the drive stops, enter

`AFF7:60`  
`AFFD:60`

This keeps DOS from writing or reading the altered VTOC from the Ultima ][ disks and thus prevents errors when copying the disks.

5) Run COPY or COPYA. Copy all three Ultima ][ disks as you would normally (yes, copy the Player Master disk this way, even though it is normally COPYAble). If you have a character disk you want to keep, also copy it.

6) Boot your System Master or any regular disk and enter the monitor again by typing

`CALL -151`

7) Enter the following short program

`300:20 F7 AF 20 0C FD 20 FD AF 60`

This program will copy the normal VTOC from the System Master to the copied Ultima ][ disks.

Put in the System Master and type

`300G`

DOS then will read the normal VTOC into memory. When a cursor appears, insert a copied Ultima ][ disk and push a key. If the Apple beeps or nothing happens, start over from Step 6. When you push a key, the drive should whir and write the normal VTOC in memory to the copied Ultima ][ disk. Repeat the procedure by putting in another of the copied disks and typing

`303G`

Also do this for the last Ultima ][ disk. If you also have a copied character disk, insert it and type

`303G`

It will put a normal VTOC on that disk, as well.

8) Now insert your copied Ultima ][ Program Master and type

`BLOAD HELLO`
(Yes, you can do this from the monitor.) Make the following changes:

72E0: A9 4C 8D F8 03 A9 79 8D
72E8: F9 03 A9 50 8D FA 03 60

Now type

UNLOCK HELLO
BSAVE HELLO, A$60000, L$1420
LOCK HELLO

This modification keeps Ultima ] ] from testing the disk to see if it is a copy (if it is, Ultima will crash), prevents it from booting the disk when reset is pushed, and sets up a ☞ Y jump back into the program for use when you alter your character in the following APT.

Your Ultima ] ] is now copied and ready to be played.

Ultima APT

Now that you can push reset in the middle of the game without booting the disk, you can edit your character to your heart’s content. For example, if you have only one unit of food and you are stranded in the middle of nowhere, miles and years away from a town, push reset. (If you are on a horse or frigate do not push reset or you will lose whatever transportation you are using. You first should get off whatever it is by pushing X for Exit and then reset. When you come back to the game, just hit B for Board.)

Hitting reset should leave you in Applesoft. Enter the monitor with

CALL -151

Now you can change your character’s food, hit points, or whatever else you need by just entering the appropriate address from the Address Chart (page 00), a colon (:) and the value you wish to have in decimal (00-99). You should only enter values in hexadecimal where noted in the list of addresses. If you ever need to know a value, type in the address and hit return. For example, to gauge your strength, enter

4E15

You should see “4E15- 16” or whatever your strength may be. You do not necessarily have to be in the middle of a game to edit your character. Simply insert the character disk, type

BLOAD PLAYER
CALL -151

and you are ready to change your character. Since your character
is stored in memory $4E00 to $4EFF, when you are done you should enter

BSAVE PLAYER, A$4E00, L$100

NOTE: If the address is two bytes, as food and hit points are, put the first two digits (in decimal), a space, and then the last two digits (in decimal, also). Say, for example, you wanted to change your food to 487. You would push reset, CALL -151, and 4E1D:04 87. To get back into the game, enter ©Y.

It is important never to save a new file onto one of the copied Ultima I disks, since this may write over the other programs on the disk. A new program means one which is not already on the disk. It is all right to save your character (file name PLAYER) to your character disk, since that file has always been there. Accidentally saving a new file on the disk may necessitate making a new copy from scratch.

The address list contains all the addresses we have found. The addresses followed by question marks are unknown, but their purposes may be revealed by further experimentation. You can do this by changing the unknown value and seeing how it affects your location and/or status. Some of the effects are strange, and it would be advisable to turn off the computer if it gets bizarre to avoid accidentally storing jumbled data on your disk.
<table>
<thead>
<tr>
<th>Location</th>
<th>Item or Characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4E00-$4E0F</td>
<td>Name (up to 15 letters)</td>
</tr>
<tr>
<td>$4E10</td>
<td>Sex (M for male, F for female)</td>
</tr>
<tr>
<td>$4E11</td>
<td>Class of character: 0=fighter, 1=cleric, 2=wizard, 3=thief</td>
</tr>
<tr>
<td>$4E12</td>
<td>Race of character: 0=human, 1=elf, 2=dwarf, 3=hobbit</td>
</tr>
<tr>
<td>$4E13-$4E14</td>
<td>Time Zone/Surface Flag</td>
</tr>
<tr>
<td>$4E15</td>
<td>Strength</td>
</tr>
<tr>
<td>$4E16</td>
<td>Agility</td>
</tr>
<tr>
<td>$4E17</td>
<td>Stamina</td>
</tr>
<tr>
<td>$4E18</td>
<td>Charisma</td>
</tr>
<tr>
<td>$4E19</td>
<td>Wisdom</td>
</tr>
<tr>
<td>$4E1A</td>
<td>Intelligence</td>
</tr>
<tr>
<td>$4E1B-$4E1C</td>
<td>Hit points</td>
</tr>
<tr>
<td>$4E1D-$4E1E</td>
<td>Food</td>
</tr>
<tr>
<td>$4E1F</td>
<td>??</td>
</tr>
<tr>
<td>$4E20-$4E21</td>
<td>Experience points. Location $4E20 is also the level of your character.</td>
</tr>
<tr>
<td>$4E22-$4E23</td>
<td>Gold pieces</td>
</tr>
<tr>
<td>$4E24-$4E25</td>
<td>Location of player on map. This is an X, Y value. For example, change these bytes when</td>
</tr>
<tr>
<td></td>
<td>you are stuck on an island to move to the nearest land mass. (Use hex values for X and</td>
</tr>
<tr>
<td></td>
<td>Y)</td>
</tr>
<tr>
<td>$4E26-$4E2A</td>
<td>??</td>
</tr>
<tr>
<td>$4E2B</td>
<td>Weapon in hand. This is the weapon you are using. You don’t have to own the weapon to</td>
</tr>
<tr>
<td></td>
<td>use it. For instance, you may change this to “phascer” without having a phaser in your</td>
</tr>
<tr>
<td></td>
<td>possession or even being able to wield it.</td>
</tr>
<tr>
<td></td>
<td>0=Hands, 1=Dagger, 2=Mace, 3=Axe, 4=Bow, 5=Sword, 6=Great sword, 7=Light sword, 8=Phaser, 9=Quick sword</td>
</tr>
<tr>
<td>$4E2C</td>
<td>Type of armour that you are wearing. The same applies to armour as to weapons. 0=Skin,</td>
</tr>
<tr>
<td></td>
<td>1=Cloth, 2=Leather, 3=Chain, 4=Plate, 5=Reflect, 6=Power</td>
</tr>
<tr>
<td>$4E2D</td>
<td>Spell. This is the spell you are ready to cast. Unfortunately, you must own the spell</td>
</tr>
<tr>
<td></td>
<td>in order to cast it, but you can change that! 0=None, 1=Light, 2=Ladder Down, 3=Ladder</td>
</tr>
<tr>
<td></td>
<td>up, 4=Passwall, 5=Surface, 6=Prayer, 7=Magic missile, 8=Blink 9=Kill</td>
</tr>
<tr>
<td>$4E2E</td>
<td>Torches</td>
</tr>
<tr>
<td>$4E2F</td>
<td>Keys</td>
</tr>
<tr>
<td>$4E30</td>
<td>Tools</td>
</tr>
<tr>
<td>$4E31-$4E40</td>
<td>??</td>
</tr>
</tbody>
</table>
Each byte of this section represents how many of each item you have (i.e. a $15 at location $4E41 means you have 15 daggers, a $78 at location $4E43 means 78 axes, etc. See $4E2B).

<table>
<thead>
<tr>
<th>Location</th>
<th>Item</th>
<th>Location</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4E41</td>
<td>dagger</td>
<td>$4E42</td>
<td>mace</td>
</tr>
<tr>
<td>$4E43</td>
<td>axe</td>
<td>$4E44</td>
<td>bow</td>
</tr>
<tr>
<td>$4E45</td>
<td>sword</td>
<td>$4E46</td>
<td>great sword</td>
</tr>
<tr>
<td>$4E47</td>
<td>light saber</td>
<td>$4E48</td>
<td>phaser</td>
</tr>
<tr>
<td>$4E49</td>
<td>quick sword</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**WEAPONS**

<table>
<thead>
<tr>
<th>Location</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4E61</td>
<td>cloth</td>
</tr>
<tr>
<td>$4E63</td>
<td>chain</td>
</tr>
<tr>
<td>$4E65</td>
<td>reflecting</td>
</tr>
</tbody>
</table>

**ARMOUR**

<table>
<thead>
<tr>
<th>Location</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4E81</td>
<td>light</td>
</tr>
<tr>
<td>$4E83</td>
<td>ladder up</td>
</tr>
<tr>
<td>$4E85</td>
<td>surface</td>
</tr>
<tr>
<td>$4E87</td>
<td>magic missile</td>
</tr>
<tr>
<td>$4E89</td>
<td>kill</td>
</tr>
</tbody>
</table>

**SPELL**

<table>
<thead>
<tr>
<th>Location</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4EA0</td>
<td>ring</td>
</tr>
<tr>
<td>$4EA2</td>
<td>staff</td>
</tr>
<tr>
<td>$4EA4</td>
<td>cloak</td>
</tr>
<tr>
<td>$4EA6</td>
<td>gem</td>
</tr>
<tr>
<td>$4EA8</td>
<td>red gem</td>
</tr>
<tr>
<td>$4EAA</td>
<td>green gem</td>
</tr>
<tr>
<td>$4EAC</td>
<td>blue tassle</td>
</tr>
<tr>
<td>$4EAE</td>
<td>green idol</td>
</tr>
</tbody>
</table>

**MISC.**

<table>
<thead>
<tr>
<th>Location</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4E82</td>
<td>ladder down</td>
</tr>
<tr>
<td>$4E84</td>
<td>passwall</td>
</tr>
<tr>
<td>$4E86</td>
<td>prayer</td>
</tr>
<tr>
<td>$4E88</td>
<td>blink</td>
</tr>
<tr>
<td>$4EA1</td>
<td>wand</td>
</tr>
<tr>
<td>$4EA3</td>
<td>boots</td>
</tr>
<tr>
<td>$4EA5</td>
<td>helm</td>
</tr>
<tr>
<td>$4EA7</td>
<td>ankh</td>
</tr>
<tr>
<td>$4EA9</td>
<td>skull key</td>
</tr>
<tr>
<td>$4EAB</td>
<td>brass button</td>
</tr>
<tr>
<td>$4EAD</td>
<td>strange coin</td>
</tr>
<tr>
<td>$4EAF</td>
<td>tri-lithium</td>
</tr>
</tbody>
</table>
Ultima II

Sierra On-Line, Inc.

Requirements:
Ultima II, 3 disks
One disk drive
COPYA (On 3.3 System Master)
Sector editing program, such as DiskEdit
3 blank disks

By Pat Tilsworth

Ultima II from Sierra On-Line is the second of the three great fantasy adventures written by Lord British. Faster play, less disk flipping and greater length make this game a tremendous improvement over the first Ultima.

When trying to backup Ultima II, I noticed that the Program Master seemed to copy easily with COPYA. When booting the duplicate, however, I found that the HELLO program seemed to be checking for a nibble count. This protection scheme relies on the slight difference in speed between the original copying drive and any other drive (i.e. yours). The unique number of nibbles copied at the original drive speed is stored on the Ultima II disk and accessed by the HELLO program when Ultima II is booted. When the HELLO program compares nibble counts, the count of the duplicate will always differ from the count the program requires to run, because the duplicate was copied at another speed.

Copying Ultima II

I wasn’t about to nibble-count every track of the Player Master with a nibble copier so I set off to unprotect it. The modification needed would have to prevent the HELLO program from checking for the nibble count. This softkey also allows each Ultima II disk to be catalogued, enabling them to be used with the Ultima II Character Generator (Published in Hardcore COMPUTIST No. 4).

1) Boot the 3.3 system master disk.
   PR#6
2) Run the COPYA program.
   RUN COPYA
3) Copy all three disks of Ultima II with COPYA.
4) When finished, boot your disk editor. It will be used to
modify each Ultima ][ disk.
5) Insert the copy of the Ultima ][ Program Master into your disk drive.
6) Read track $11 (17 decimal), sector $00.
7) Modify the bytes found at the following locations (Don’t forget, the first byte of the sector is location 00.)

<table>
<thead>
<tr>
<th>Byte</th>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td>$FF</td>
<td>$11</td>
</tr>
<tr>
<td>$02</td>
<td>$FF</td>
<td>$0F</td>
</tr>
</tbody>
</table>

This modification allows the disk to be catalogued by pointing to track $11, sector $0F.
8) Write the sector back to the disk.
9) Perform Steps 6-8 on the copies of the Player Master and Galactic disks.
10) Place the copy of the Program Master into your drive.
11) Read track 3, sector C.
12) Modify the following values:

<table>
<thead>
<tr>
<th>Byte</th>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>$84</td>
<td>$20</td>
<td>$EA</td>
</tr>
<tr>
<td>$85</td>
<td>$E0</td>
<td>$EA</td>
</tr>
<tr>
<td>$86</td>
<td>$72</td>
<td>$EA</td>
</tr>
</tbody>
</table>

This final modification prevents the HELLO program from performing the nibble-count check routine at $72EO. (Location $84 was a JSR to the nibble-count check.)
13) Write the sector back to the disk.

You now possess an unprotected version of Ultima ][, and the Player Master can be catalogued for use with the Ultima ][ Character Generator. Files created under normal DOS 3.3 should not be saved to these disks since DOS 3.3 does not know where the real Ultima VTOC exists. However, a program such as FID can be used to copy all the Ultima files onto normal 3.3 disks.

In addition, since the nibble count has been bypassed, the three unprotected Ultima disks will now boot when duplicated with any program which copies an entire disk.
Visifile

VisiCorp

Requirements:
48K Apple or an Apple //e
Visifile
Two blank disks
Apple's COPYA program
A disk edit program
(An Applesoft Program Line Editor, such as Konzen's GPLE, is useful but not essential)

By Bob Bragner

Visifile is a medium-powered, somewhat overpriced data base manager. My first (original) copy got zapped when the Turkish Electric Company hiccuped during a configuration file write on the master disk. Since I didn't have a backup and I knew that the disks were protected, I packed both of them off to VisiCorp along with a check for $30 for a replacement.

After nearly two months, the disks finally made their way back across the Atlantic and Mediterranean with the enchanting message: "Undeliverable at this Address" stamped on the package! VisiCorp had apparently moved and not notified the Post Office.

By this time I had, of course, found their new address (no thanks to them) and once again shipped the disks off with a somewhat caustic letter. This time the disks were returned updated and with a backup to boot (sorry for the pun) in about three weeks. Nevertheless, having been burned once, I decided I had to have my own copyable Visifile.

Locksmith will copy Visifile, but requires a lot of annoying parameter changes, and the copy will remain protected. What I really wanted was a "cracked" version and backups squirrelled all over the place.

Pirate's Harbor published a crack for Visifile. It consisted of copying the disk with COPYA, then changing one byte in one of the sectors on the disk using a disk zap utility like Watson. However, COPYA refused to copy my disk. Every time I tried it, I got an "****** UNABLE TO WRITE ******" error when the program tried to format the blank disk. Using Watson, I was able to determine that there were no protected sectors on the original disk. FID could move all the files (except for the dummy serial number) over to another disk, but if you try to boot the result, the screen fills up with inverse "A's".

After using the FIXCAT utility from Bag of Tricks on the original, it was clear that there were some peculiar things in the
catalog track. For one thing, the volume number appeared to be 255 ($FF: an invalid volume number!) even though it showed up as 254 when you looked at the catalog. If you examine line 250 of the COPYA program you will see:

```
250 PRINT "INITXXX,S";SS; "D";SD; ":V";PEEK (714) :FT = 1
```

Checking the value at location 714 after COPYA crashes reveals that there is indeed a 255 there. So if you changed line 250 by adding a "-1" after the "PEEK (714)" then COPYA would make a perfect copy of both Visifile disks. These copies can be copied as much as you like by a normal, unaltered COPYA. However, if you try to boot Disk 1, you still get a screenful of reversed "A’s."

After a bit more snooping, it was easy to determine that the blowup occurred when the file VISIFILE.BIN on Disk 1 was BRUN. A quick disassembly of this file didn’t reveal anything significant (although there is a section where there are a bunch of reversed “A’s”) but then I remembered the Pirate’s Harbor crack: byte $2D of track $22, sector $04, was supposed to change from $0A to $0F and this sector was part of the VISIFILE.BIN file!

After making this change with Watson, I booted the resulting disk and all was well.

Here is a step-by-step procedure to crack Visifile:

1) First load COPYA.

```
LOAD COPYA
```

2) Edit line 250 by inserting "-1" after the PEEK(714).

```
250 PRINT ‘INITXXX,S’ SS ‘D’ SD ‘V’ PEEK (714) - 1 :FT = 1
```

3) If GPLE is lurking around, remove it before you attempt to make a copy.

4) Run the program and follow normal copy procedures.

```
RUN
```

(Repeat Step 4 for Visifile Disk 1 and Disk 2.)

5) Enter your favorite disk zapper and read track $22, sector $04 of the copy of disk 1. Change byte $2D from $0A to $0F. Write this change to the copied disk.

You now have a cracked copy of Visifile from which you can make all the backups you want, using normal copy procedures.

Do you want a faster sort routine? Hard disk capability? With your cracked Visifile, you are now free to modify to your heart’s content.
Visiplot/Visitrend

VisiCorp

Requirements:
48K Apple with Applesoft in ROM
One disk drive
Visiplot/Visitrend
One blank disk

By Anthony L. Barnett

A government department for which I work recently purchased Visiplot/Visitrend. Naturally, a backup disk was desired. However, the only "legal" way of obtaining one appeared to be by making an overseas order directly to VisiCorp.

This is by no means an easy procedure. So, a letter was sent to VisiCorp at the address in the manual. This was promptly returned by the US Post Office as "undeliverable at this address". Recent magazines were perused to find VisiCorp's current address and the letter posted again.

VisiCorp was asked whether the order for a backup could be placed through an Australian agent. Eventually, the terse reply of "no" was received scribbled over a standard form which advised, among other things, that our request could not be met as we had not sent our disk backup order form!

Not knowing the Locksmith parameters, I began to examine this curious disk for other means to back it up. All the programs are quite listable and FIDable but a disk check causes a spectacular crash if the original disk is not used.

I determined that the disk check occurs in the main storage program and, after studying the listing for about an hour, I determined that six bytes needed to be altered to get the backup to run.

In line 4 the "& A" should be replaced with two colons and, in line 2300, the "CALL 960" should be replaced with four colons.

As the disk check is now eliminated, the backup works slightly faster when switching to and from main storage. It is also possible to use Speed-DOS from Hardcore Computing Update 3.2 (old series). The switching between programs is then quite fast and tolerable.

It is my view that no program should be protected. Failing this, at least two copies of a business program should be provided. This can be done in the package or as a free backup on receipt of registration. Another less satisfactory means is to provide a special user copy program (usually "once only" like Multiplan).
Wizardry
Sir-Tech

Requirements:
48K Apple with Applesoft in ROM
Locksmith 4.1
One blank initialized disk
At least one disk drive
One small Phillips screwdriver
One small standard screwdriver (see Step 7, option C)

By John Samborski

According to the authors of Wizardry, their program uses "state-of-the-art copy-protection." This label fits very well, as it is truly a state-of-the-art program. Robert Woodhead and Andrew Greenberg anticipated the popularity of Wizardry when they designed their protection scheme. Unfortunately, it's the hardest disk backup chore I've ever faced.

For all who want the security of a backup of Wizardry, this article provides a complete set of instructions for making a copy. The boot side, then the scenario side, will be duplicated using Locksmith 4.1.

Copying The Boot Side

1) Boot Locksmith 4.1.
2) If using one drive, remove Locksmith and insert the Wizardry disk. If two drives are available, insert the Wizardry disk in drive 2.
3) Use the "Automatic Error Retry" option on all tracks listed.
4) Copy tracks 0-22 unsynchronized.
5) If all is well (it should be), set parameter 36 to 01.
6) Copy tracks 0A-0E synchronized.

Step 7: Adjusting The Drive Speed

The Wizardry program checks for "preservation of nibble count." Unfortunately, when this kind of protection scheme is used the drive speed must be absolutely perfect to make a successful copy. Locksmith will do the normal analysis, but when it reaches the point of writing and verifying, some strange digits will be printed on the screen such as >0010 or <000A. These figures indicate the speed difference between the original recording drive and the drive you are using. If the sign is ">" the drive is running slow. If "<" appears, it's running fast. At this point, there are three options available. Read each before
deciding which is appropriate:

A) Do nothing. The Apple will try to compensate the speed. Judging by the difference in drive speed, this can take anywhere from three minutes to three weeks. This is recommended only for perfectly adjusted drives.

B) Use the "<" and ">") keys to correct the drive speed. To do this, look at the sign in front of the digits and hit that key. For example, if >001A appears on screen, hit the ">' (shifting is unnecessary). When this key is hit, the bell will ring. Press the space bar to continue. The longer you let the bell ring, the more the speed will be adjusted. Repeat this as needed. When the speed is adjusted to within 0006 (>0006 - <0006), leave it alone and let the drive try to compensate the remainder by itself.

NOTE: For option C, use a blank disk.

C) If the drive speed is substantially off, step B is impractical. The speed will have to be compensated by adjusting a screw inside the drive with a small Phillips screwdriver and a small standard screwdriver. Follow this procedure

Turn the Apple off.

Unscrew the four Phillips-head screws which hold the drive cover in place.

Slide the cover to the rear and off of the drive so that the tiny screw which controls drive speed can be located. (It's not on the circuit board — leave all screws on the circuit board alone.) It is by the rear cover, mounted horizontally with its head to the right side of the drive. This screw will be used later to correct the drive speed.

Turn the Apple on and boot Locksmith 4.1.

Set parameter 36 to 01.

Copy tracks OA-OE synchronized.

When the digits appear on screen showing how far off the drive speed is, use the standard screwdriver to turn the small screw which controls speed. Turn the screw in the direction that was indicated by the ">' or "<": right increases the speed, left slows it down.

When the speed comes within 0009 (>0009-<0009), use the "<" and ">") keys for fine adjustment.

Replace the drive cover.

Back To The Original Procedure

8) When the digits indicate >0000, the track has been copied. The user will be prompted to insert the source disk (one drive) or, if two drives are being used, jump to the source drive.
Assure that >0000 is printed on screen before reading the next track. Sometimes the program "gets tired" of trying to synchronize the drive speed (some drives only). If >0000 isn't printed, the copy probably didn't work.

9) Finish copying the boot side, then put a write-protect tab on the copied disk.

10) Place the copy in the drive and boot it.

   If you see that pretty picture and the menu, congratulations! You're now half done.

   If the copy wasn't successful, repeat the ten steps. It works about three times out of five for me. The protection scheme is a tough one.

**Copying The Scenario Side**

The scenario side of Wizardry can be copied using the same basic procedure that was used for the boot side. Repeat Steps 1-10, but leave out Step 9 since the program writes to the disk as it goes along.

Don't be discouraged if it doesn't work the first time. This side is even tougher to copy than the boot side. On my attempts it worked about two out of nine times.

Enjoy the added peace of mind you have with a backup copy of Wizardry. I only use my backup; the original sits in a dark, dry place, safe from magnetic fields.
Looking through your last Hardcore issues, I saw on the parameter exchange how to backup Wizardry by Sir Tech. While the program uses state-of-the-art copy protection, there is a much simpler way of making backups and it works every time, not just 3 out of 5 times.

First, copy tracks 0-22 unsync with auto retry. Or if, like me, you hate using Locksmith because it is so slow, you can use COPYA to copy the disk. After copying the disk with Locksmith or COPYA, go back to Locksmith and copy track 0A-0E SYNC and change these to parameters 46=96, 21=02.

That's it, and have fun.
Zork I

Infocom, Inc.

Requirements:
Apple II
At least one disk drive
A copy of Zork
COPYA
A disk editing program such as DiskEdit

By Bobby

This copy method works on Zork versions I, II, and III. It also works for Deadline and Witness.

Zork is a challenging game and although it lacks a hi-res picture (it is all text), is one of the best adventure games I have ever attempted.

While trying to solve some of the puzzles, I started to do a little APT and found that Zork was on a protected disk. I set it aside until I had the time to examine the program but then a reader called to explain a method to unprotect Zork. Believe it or not, the COPYA program on the system master disk can be used.

How To Copy Zork

1) Insert the DOS master disk and run COPYA.

RUN COPYA

2) Once loaded, stop the program

ESC

3) Delete line 70 so the machine language subroutine will not reload.

DEL 70

4) Enter the Monitor and make the following changes:

CALL -151 Enter monitor
B925:18 60 Ignore end of data marks
B988:18 60 Ignore end of address marks
BE48:18 Ignore errors
B8FE:00 Ignore 3rd byte of start of data mark
3D0G Exit to basic

5) Restart the copy program

RUN
6) Follow the prompts to make a copy.
7) After the disk is copied, use a disk editing program to read track 0, sector 2 and make these changes:

<table>
<thead>
<tr>
<th>Byte</th>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>5D</td>
<td>BC</td>
<td>AD</td>
</tr>
<tr>
<td>FE</td>
<td>E7</td>
<td>00</td>
</tr>
</tbody>
</table>

You now have an unprotected disk that can be copied with various copy programs, including COPYA.

The disk cannot be catalogued, nor may separate files be run; it must be booted to play the game. But Zork is now open to inspection by those wishing to participate in the rapidly growing hobby of APT.