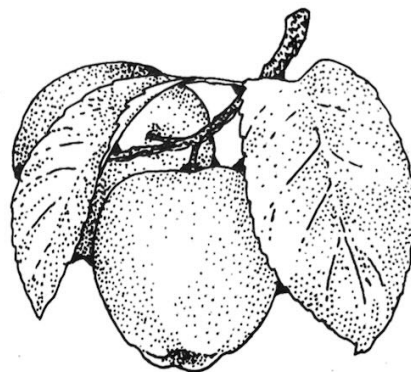


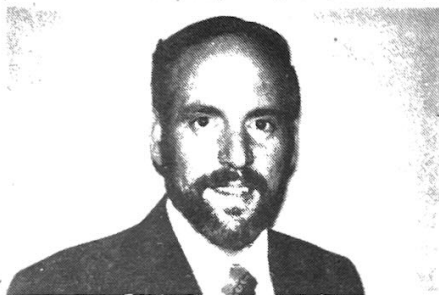
APPLE SOFTWARE PROTECTION DIGEST

\$3.00



Vol. 1, No. 1

Premiere Issue



Contents

EDITORIAL.....	1
HIDING APPLESOFT.....	2
PROGRAM LINES.....	3
APPLESOFT LINE FINDER.....	3
PROTECTION TUTORIAL—Part I.6	
Auto-Running Programs.....	6
Making Them Hard to Copy....	6
Skip Track 3 Formatter.....	6
Adding Extra Tracks.....	8
BACKING UP <i>PRINT SHOP</i>	10
Print Shop Copy Program.....	10
REVIEW: Copy II Plus.....	11
COMING NEXT ISSUE.....	11

LEARNING TO LIVE WITH PROTECTION

Welcome to the Apple Software Protection Digest. This is the first issue of what will be a monthly publication that is dedicated to the subject of protection and how it relates to software for the Apple // series of computers. If you're like me, you've doubtless seen scores of articles in the various computer publications that tackle the subject of software protection. Most of them take a stand for or against it and that's the last you hear of the subject. But more is needed. Software protection is a fact of life and we must learn to live with it.

Too often, purchasers of programs get stuck. They buy a program only to discover that it can't be used with a particular accessory board or disk drive. And, because the program's protected, it can't be modified. Spelling checkers are a good example of this. They're frequently difficult or impossible to use with non-standard or hard disk drives, even though unprotected programs work without difficulty on these drives.

Apple computer owners need a place where they can get more information about software protection. They need a forum where they can exchange ideas with others who face the same or similar problems. They need to know what software protection is, how it's implemented, what are the consequences of it, how it can be overcome if necessary and if there are any comparable unprotected alternatives to particular protected software packages.

Apple Software Protection Digest will provide you with this information and more. It will show you new ways to protect, unprotect and backup your programs. It will teach you how to prevent others from accessing your programs and it will show you how to make them more difficult to copy. In addition, you'll learn how to overcome these and other protection schemes that are in use. You'll learn how to use the powerful, but complicated nibble copy programs. You'll also learn how to *crack* or remove protection entirely from many programs.

With the programs that are included in each issue of the digest, you'll build a valuable library of utility software that will make the job of protecting, unprotecting and backing up software easier. And, on-going tutorials will keep you up-to-date on both simple and sophisticated protection techniques. In addition to all this, every month you will get reviews of hardware, software and books that are of particular importance to the software protection field.

We do not advocate software piracy, because we firmly believe that in the long-run piracy only leads to more expensive and lower quality programs and less user support. Programmers work long, hard hours to get their software working and they deserve to be compensated fairly for it. This cannot happen if software is stolen. On the other hand, the honest consumer should not be penalized and limited in his or her application of a particular program simply because the publisher decided to protect it. You may have a legitimate need to back up a program and we hope to supply you with the knowledge you need to do that. Alternatively, you may wish to protect a program that you've written so that others can't copy it. We'll show you how to do that too.

Apple Software Protection Digest welcomes your comments, tips and article contributions. If you have a problem backing up a particular program, let us know, we'll try to help. If you've discovered a way to copy or crack (unprotect) a particular program, let us know about that too. Most likely there are other people who would like to know how to do it also. Finally, if you've come up with an ingenious hardware or software oriented protection scheme, write to us about it so that we can share it with others. If your article is used, you'll receive a free 6-month subscription (or subscription extension) to our publication. Let's hear from you soon.

Jules H. Gilder
Editor & Publisher

Apple Software Protection Digest:
Publisher & Editor, Jules H. Gilder;
Contributing Editor, J. Scott Barrus.
Copyright (c) 1985 by Redlig Systems,
Inc., 2068 - 79th Street, Brooklyn, NY
11214. All rights reserved. No part of this
publication may be reproduced, or elec-
tronically transmitted or stored without
the publisher's written permission. Pub-
lished monthly at \$24 per year by Redlig
Systems, Inc., (718) 232-8429. Reprints of
prior issues available for \$3 each.

Apple is a registered trademark of Apple
Computer, Inc.

HIDING APPLESOFT PROGRAM LINES

Sometimes when you write an Applesoft program, you may find it desirable to make certain sections of it invisible. Maybe you've developed a unique way of solving a problem and you don't want others to copy your algorithm, maybe you want to bury a copyright notice in the code where it won't be easily spotted and deleted, maybe you want to use a machine language subroutine without announcing it to the world, maybe you want to include a password system to prevent unauthorized use of your program or maybe you want to hide the code that implements your copy protection scheme. Whatever the reason, the need to hide one or more Applesoft program lines arises frequently and you should know how to do. I'm going to show three ways to do the job, with the best one being the last one.

Hiding Applesoft lines is an old trick that was used in a lot of early protected software. The most common way to make a line disappear is to end it with a :REM statement and then imbed backspace characters in the REM statement. You'll need one backspace for every character that is going to be hidden plus six additional ones. Remember to include the line number and the spaces that separate it from the text of the line, in the character count. After you've put in the backspace character (and I'll tell you how to do that in a minute) then you'll need an equal amount of additional text that is going to be printed over the text that is to be hidden.

If you try to type the program line: 10 REM and then try to enter backspaces so the line can be erased, you'll quickly find that you have a problem. Go ahead try it. Instead of inserting the backspace character into the REM statement, when you type a backspace (or left arrow) from the keyboard, it moves the cursor back and prevents the line from being stored in memory. This is not what we want. Since backspaces cannot be entered into a program line directly, we're going to have to force it in. To do this, we use a process called *patching*. Patching requires that we place a dummy character in the REM statement every place where there's going to be a backspace. After that's done, we must search the computer's memory for the dummy character and replace it with the code for a backspace.

Since we'd like to automate the search and replace task, a dummy character should be chosen that's not used anywhere else in the program. Several such characters exist on the Apple //e and //c keyboard, but only one exists on the Apple II Plus keyboard, so we'll use that one. The character is the *at sign* (@).

To show you how this works, let's make the following Applesoft program line disappear:

```
10 PRINT "A"
```

If we count everything to be hidden (including the line number) we find that we'll

need thirteen back spaces. But remember I said you have to add on six to this, so the total is nineteen. Now that we've backspaced to the beginning of the display line, we have to overprint the line to make it disappear. The message you print must be at least as long as the text being hidden. Thus, the new line to be entered should look like this:

```
10 PRINT "A": REM@@@@@@@@@
@@@@@@@@@@@@@NOW THE LINE
IS HIDDEN
```

It's important that you do not leave a space between the REM and the first @. Now, if we wanted to, we could get into the monitor by typing CALL-151 and change all those *at signs* to the backspace character (which is ASCII 8). But that's tedious, so let's make the computer do it. Type in the following line with no line number so that the computer will start executing it the moment you press <RETURN>. If you're going to use this technique a lot, you might prefer to create a TEXT file with this line in it and then EXEC it every time you need it.

```
D=ASC("@"):FOR X=2049 TO PEEK
(175)+256*PEEK(176):POKE X,(PEEK
(X)<>D)*PEEK(X)+8*(PEEK(X)=D)
:NEXT
```

What this line does is set D equal to the ASCII code of the dummy character that is to be replaced. If you decide to use some character other than the @, simply place it between the quotation marks in the line above. Next the computer starts at the beginning of the Applesoft program (location 2049) and checks every byte until the end to see if it's equal to the @ character. If it is, that character is replaced with an 8, which is the backspace character, otherwise it's left alone.

If you look carefully at the one line program that performs the search and replace, you'll notice that there's no IFTHEN statement in it, although from the verbal description it seems as though one is needed. IF...THEN statements cannot be used as easily from the immediate mode, so another way of doing the same thing had to be found. If you haven't already guessed by now, it has to do with the strange statement POKE X, (PEEK (X) <> D)*PEEK (X) + 8*(PEEK (X)=D).

Here we're telling the computer to look at the current location indicated by X. If the value stored there is not equal to D the expression (PEEK (X) <> D) is true and is mathematically equal to a 1, otherwise it's set equal to 0. At the same time, if the contents of location X are not equal to D, the expression 8*(PEEK (X)=D) is not true and evaluates to zero. Thus, whatever was in location X is stored back there again. However, if the contents of location X are equal to D, (PEEK (X) <> D)*PEEK (X)

becomes zero and 8*(PEEK (X)=D) becomes 8 (the backspace character), and that's what's stored in memory.

If you've been following so far, and you've typed everything in, you've noticed that when you RUN the program it prints out an A like it's supposed to, but when you try to list it, you get the message "NOW THE LINE IS HIDDEN" printed instead. Good for you. You've just implemented one of several protection techniques that were used on early Apple programs. But don't get too excited, because while this technique works nicely for normal operation of the computer, as early software publishers quickly found out, issuing one simple command overcomes all the hard work you've just done. With your invisible line program still in memory, type in the following two lines in the immediate mode (without a line number):

```
SPEED=100
LIST
```

The invisible line lists out to the screen very slowly and you can see it get erased slowly too. And, if you press Control-S before the overprinting starts, you can freeze the line on the screen for as long as you need to copy it down on a piece of paper.

As you can see, this is not a very secure way of hiding Applesoft program lines. In addition, it has a tremendous overhead, and requires a lot of additional memory for each line that is hidden. Surely there must be another way.

Hide a line between two others

If you understand how Applesoft stores a program in memory (and I'll explain that in a minute) you can hide a line between two other lines and make it completely invisible. A line hidden in this manner will function correctly, but will not be visible at all, even if you set SPEED to a very low number. Let's first see how Applesoft stores a program line in memory by looking at the following line:

```
10 PRINT 123
```

If we were to look directly into memory, we'd see that the line was stored in the following way:

Address	801	802	803	804	805	806	807	808	809
Contents	0A	08	0A	00	BA	31	32	33	00

Looking at locations \$801 and \$802 (2049 and 2050 in decimal) we see two hexadecimal numbers stored there: 0A and 08, which comprise the hexadecimal number \$080A. In 6502 microprocessor systems, hex numbers are always stored in memory with the low-order byte first. This number represents the location in memory of the start of the next line of the Applesoft program. So if we were to add another line to our program, it would start at location \$80A. Thus, the first to bytes of any Applesoft program line are called the *next line pointer*.

The next two bytes at \$803 and \$804 hold the hex equivalent of the line number. Since our line number is less than 255, only the low-order byte is used (it's set to \$0A which equals 10 in decimal). The high-order byte is set to zero. Next, on the fifth byte (\$805) we have the start of our line. \$805 contains \$BA which is the code, or token, that represents the word PRINT and the following locations contain \$31, \$32 and \$33 which are the hex ASCII codes for 1, 2 and 3. Finally, there's a zero, which serves as an end of line marker.

If you're a good detective, by now you may have guessed that this method of hiding lines changes the next line pointer so that the line you want to hide is bypassed. Let's clarify things by using the following three-line program as an example:

```
10 PRINT
20 PRINT "THIS IS A TEST"
30 PRINT
```

Let's get into the monitor by typing CALL -151. Then type 801.824. The display you get should look like this:

```
0801- 07 08 0A 00 BA 00 1D
0808- 08 14 00 BA 22 54 48 49
0810- 53 20 49 53 20 41 20 54
0818- 45 53 54 22 00 23 08 1E
0820- 00 BA 00 00 00
```

If we want to hide line 20, then all we have to do is change the pointer to it (which is at \$801). Let's change the pointer so that instead of pointing to line 20, it points to line 30. If we look at \$807, which is where line 20 begins, we see that it points to \$81D which is where line 30 begins. Now let's change the pointer to line 20 by typing:

```
801:1D
```

Next, get back to Applesoft by typing 3D0G and LIST the program. You should only see line 10 and line 30. Line 20 seems to have disappeared completely. However, if you type RUN, you'll see that line 20 is indeed still there because it prints out the message "THIS IS A TEST". The reason the line is not listed, but is executed is that the *next line pointer* is only used by the LIST, GOTO and GOSUB routines. The routine that executes a program just starts at the beginning of the program and executes everything it finds in consecutive order.

Well, it looks like we found a good way to hide an Applesoft line. We did, as long as we don't change, add or delete any lines after we've hidden the ones we want, and the lines have to be hidden under program control, after the program has been loaded. The reason is, each time we add, delete or change a line, Applesoft recalculates the next line pointers by calling a routine in the ROMs known as LINKSET (\$D4F2) and *corrects* the changes we made. Thus even if we try to delete a non-existing line, our hidden line will immediately re-appear. To verify this, just type 0 and <RETURN> to delete the

non-existent line zero. Now list the program. Line 20 has re-appeared again.

Unfortunately, when this version of Applesoft BASIC came out, there was also another version of the language, whose programs started at \$3000 instead of \$800, being used. In order to insure compatibility between programs written by both versions, DOS was changed to include a call to an automatic pointer resetting routine when a program is loaded from the disk. Thus, once again our protected program will be corrected when it is loaded into place. This is not too bad however, because there are ways to overcome that. We'll talk about automatically loading and running Applesoft programs and bypassing the pointer correction routine next time.

The best way to hide a line

There is still one more way to hide an Applesoft program line and these one has fewer problems and is more effective than the others. This technique takes advantage of a quirk in Applesoft and at the same time allows you to insert invisible identification, such as your initials, into the program as well.

To use this technique, all you do is precede each line that you want to render invisible with 5 colons. Then, we're going to change the code (\$3A) for the first colon of each line to a \$00. That's all there is to it. When you try to list the line, all you'll get is the line number. Let's try an example. Type in the following 3-line program:

```
10 PRINT
20 ::::: PRINT "THIS IS A TEST"
30 PRINT
```

Now get into the monitor by typing CALL -151 and type AF.B0 to get the location of the end of the program + 1. This gives:

```
AF.B0
00AF- 2A
00B0- 08
```

Next, display a hex dump of the program by typing 801.829 <RETURN>. You'll get the following:

```
0801- 07 08 0A 00 BA 00 22
0808- 08 14 00 3A 3A 3A 3A
0810- BA 22 54 48 49 53 20 49
0818- 53 20 41 20 54 45 53 54
0820- 22 00 28 08 1E 00 BA 00
0828- 00 00
```

Looking at the above hex dump, we see that the first colon (\$3A) is located at \$80B. Change it to a zero by typing 80B:0 <RETURN>. Now get back to Applesoft by typing 3D0G and list the program. This is what you should get:

```
10 PRINT
20
30 PRINT
```

If you run the program it will still print out the "THIS IS A TEST" message. By the

way, there's nothing magical about using colons, it's just that you can put them in without effecting the operation of the program in the *unprotected* mode. If you want, however, the second through fifth characters at the beginning of the line can be your initials or anything else. Thus you could have:

```
20::JHG PRINT "THIS IS A TEST"
```

However, if you tried to run a program with this line in it without changing the first colon to a zero, you'll get a syntax error. After you make the change however, Applesoft will ignore the remaining four characters so no error message is generated.

It's easy to insert patches into a program line when the line is at the beginning of the program. However, it's a pain in the neck to do it when the line is in the middle of a large program. To make this task easier, the Utility program this month has been designed to make the task easier. It's called Applesoft Line Finder and it will locate any line in the program that you specify, display a hex dump of just that line, and leave you in the monitor mode so you can make any desired changes. Of course, if you want to make all the changes automatically, you can write a short Applesoft program that gets appended to the current program and run that, or you can write a separate machine language program to do it.

To make life interesting, and to give you some incentive, we'll give a free 6 month subscription (or extension) to the best machine language and Applesoft programs that automatically make the required changes. And, if your program automatically changes the remaining four colons to an identification string that the user enters, we'll make it seven months instead of six. Go to it!

APPLESOFT LINE FINDER

Earlier, we had a short discussion on the way a line of an Applesoft program is stored in memory. Without repeating that discussion in detail, let's just review a few pertinent facts. With ROM or language card Applesoft, program storage normally starts at location \$801. The first two bytes of an Applesoft line contain a pointer to the location in memory of the next Applesoft line. The next two bytes are reserved for the hex representation of the line number. Then, the actual text of the line is stored with Applesoft keywords replaced by one-byte tokens. Finally, the line is terminated with a zero.

The program APPLESOFT LINE FINDER, takes a line number that is passed to it by the ampersand command and uses some of the routines in the Apple ROMs to first locate the position of the line in memory and then display the line in hex up to and including the terminating zero byte. The program then leaves you in the monitor mode so that you can make any changes desired in the line just displayed.

The program starts at location \$2DA, which is the upper part of the input buffer. To use it, the program is loaded and then activated by a CALL 730. Since the program is located in pages 2 and 3 of memory, it can be loaded and run at any time during an Applesoft program's development, without affecting the Applesoft program.

The first part of the program, which starts on line 1360, clears the screen, prints out the program title and sets up the ampersand jump locations on page three to point to a routine that locates the Applesoft line. Immediately following this short routine, is the text that it prints out. The reason the text is placed here up front, is that it is going to be used once, the first time the program is run, and thus is expendable. So we won't have to worry about part of our program, which is stored in the input buffer, being wiped out if a long line of text is entered.

The actual program that finds and displays Applesoft lines starts on line 1660, where an Applesoft routine called LINGET (\$DA0C) is called. LINGET is the routine that is used to check get the line number of an Applesoft line that is being entered from the keyboard. It uses TXTPTR, which is the text pointer in the CHRGET routine, and reads the number that TXTPTR is pointing to. It takes this number, converts it to hexadecimal and stores it in LINNUM and LINNUM+1 (\$50 and \$51). Because this routine is the same one that Applesoft uses to check line numbers, it has the same limitations, namely it is only good for line numbers up to and including 63999.

If you want to display lines greater than that, the JSR LINGET should be replaced by a JSR FRMNUM (\$DD67), immediately followed by a JSR GETADR (\$E752).

Once the line number has been converted to hex and stored in LINNUM, another Applesoft ROM routine, FNDLIN (\$D61A), is called (line 1670). FNDLIN will start at the beginning of the Applesoft program and search for the line number that is currently stored in LINNUM (and of course LINNUM+1). If the line is found, its beginning address is stored in two page zero locations called LOWTR and LOWTR+1 (\$9B and \$9C). Also, if the number is found the carry bit is set. If the number is not found, the next highest line number, if there is one, is stored in LOWTR and the carry bit is cleared.

Upon returning from FNDLIN, the first thing the program does is to test the carry-bit to see if the line number was found (line 1680). If it was not found, the program branches to line 1940 where a message to the user is printed that rings the bell and tells him that no such line exists in the program. If the line does exist, the Y-register and memory location TEMP are both set to zero (lines 1690 and 1700) and the program jumps to a subroutine that prints out the two-byte address of the data that are going to be displayed on the next line of the video display (line 1710). This subroutine, which is called

```

1000 *****
1010 ***
1020 ***      APPLESOFT LINE FINDER
1030 ***
1080 *****
1090 *
1100 *
1110 *
1120 *      .OR $2DA
1130 *
1140 *
1150 * EQUATES
1160 *
0008- 1170 TEMP      .EQ $8
0018- 1180 TXTPTR   .EQ $18
003C- 1190 A1L     .EQ $3C
009B- 1200 LOWTR   .EQ $9B
03F5- 1210 AMPERSD .EQ $3F5
D61A- 1220 FNDLIN  .EQ $D61A
DA0C- 1230 LINGET  .EQ $DA0C
F941- 1240 PRNTAX .EQ $F941
FC58- 1250 HOME   .EQ $FC58
FD8E- 1260 CROUT  .EQ $FD8E
FD8E- 1270 PRBYTE .EQ $FD8E
FDDA- 1280 COUT   .EQ $FDDA
FDED- 1290 MONZ   .EQ $FF69
FF69- 1300 *
      1310 *
      1320 * This is where the program title is
      1330 * printed out and the ampersand (&) vector
      1340 * jump is set up.
      1350 *
02DA- 20 58 FC 1360 JSR HOME      Clear the screen.
02DD- A9 F4 1370 LDA #TEXT1   Get the address of the
02DF- A0 02 1380 LDY /TEXT1   text to be printed.
02E1- 20 A8 03 1390 JSR MSGPRT  Print it.
02E4- A2 4C 1400 LDX #$4C    Get a JMP op code and
02E6- A9 49 1410 LDA #START  the low and high bytes
02E8- A0 03 1420 LDY /START  of START's address and
02EA- 8E F5 03 1430 STX AMPERSD store them in locations
02ED- 8D F6 03 1440 STA AMPERSD+1 $3F5, $3F6 and $3F7.
02F0- 8C F7 03 1450 STY AMPERSD+2
02F3- 60 1460 RTS
      1470 *
      1480 *
      1490 * This is the text for the title and
      1500 * copyright notice.
      1510 *

02F4- C1 D0 D0
02F7- CC C5 D3
02FA- CF C6 D4
02FD- A0 CC C9
0300- CE C5 A0
0303- C6 C9 CE
0306- C4 C5 D2 1520 TEXT1 .AS -"APPLESOFT LINE FINDER"
0309- 8D 8D 1530 .HS 8D8D
030B- C2 D9 A0
030E- CA D5 CC
0311- C5 D3 A0
0314- C8 AE A0
0317- C7 C9 CC
031A- C4 C5 D2 1540 .AS -"BY JULES H. GILDER"
031D- 8D 1550 .HS 8D
031E- C3 CF D0
0321- D9 D2 C9
0324- C7 C8 D4
0327- A0 A8 C3
032A- A9 A0 B1
032D- B9 B8 B2 1560 .AS -"COPYRIGHT (C) 1982"
0330- 8D 1570 .HS 8D
0331- C1 CC CC
0334- A0 D2 C9
0337- C7 C8 D4
033A- D3 A0 D2
033D- C5 D3 C5
0340- D2 D6 C5
0343- C4 1580 .AS -"ALL RIGHTS RESERVED"
0344- 8D 8D 8D 1590 .HS 8D8D8D8D00
0347- 8D 00 1600 *
      1610 *
      1620 * This part of the program is the main
      1630 * loop. It gets the line number, finds
      1640 * it in memory and displays it in hex.
      1650 *
0349- 20 0C DA 1660 START JSR LINGET Convert number after & to hex.

```


PRTADDR, starts on line 2050 and begins by printing a carriage return and then a space (lines 2050 to 2070). Next, the X-register is set up as a displayed byte counter (line 2080) and is used to permit the display of only eight bytes of data per line. Then the subroutine prints out the address that is stored in LOWTR and LOWTR+1, high-order byte first (lines 2090 to 2120). Finally, a colon is printed out and the program returns to the caller via the RTS in the COUT routine (lines 2130 and 2140).

After printing out the starting memory address of the line of data to be displayed on the screen, a space is printed (lines 1720 and 1730) and eight bytes of data are printed. The byte to be printed is retrieved in line 1740 and checked to see if it is a zero in line 1750. If it is a zero, TEMP is tested to see if five or more bytes have already been printed (lines 1760 and 1770). The reason for this is that for line numbers below 255, the fourth byte, which is the high-order byte of the line number is set to zero. This is not the zero we wish to detect, but rather the zero that terminates the Applesoft program line.

If five or more bytes have been printed already, we know that this zero represents the end of the Applesoft line, so the program jumps to a routine (on line 1850), that prints out the zero, then prints out a carriage return (line 1870) and finally jumps to a routine in the F8 ROM called MONZ (\$FF69) which leaves the user in the monitor mode (line 1880). If for some reason you wish to return to the program that called the APPLESOFT LINE FINDER instead of being left in the monitor, it is only necessary to replace the JMP MONZ in line 1880 with an RTS.

If less than five bytes have been printed, we know that this is not the end of the line and we print the zero out just as we would print any other byte (lines 1790 and 1800). Then the program jumps to a routine on line 2210 that increments the two-byte LOWTR pointer and also increments TEMP. After that, the X-register is decremented and tested to see if eight bytes have been printed already (lines 1820 and 1830). If not the program branches to line 1720 where the next byte is retrieved and printed. Otherwise, it branches to line 1710 where the address of the next byte to be displayed is printed. This process continues until the terminating zero of the Applesoft program line is encountered.

The subroutine located in lines 2300 to 2400 is a message printing routine. Following this routine, on line 2480, is the text for the error message that says the line doesn't exist.

Unlike most programs that use the ampersand, this one is meant to be used primarily from the immediate mode rather than being called from a running program. However, as I indicated earlier, if you want it to return to a program that called it, the change that has to be made is trivial.

continued on page 8

034C-	20	1A	D6	1670	JSR FNDLIN	Put address of line in LOWTR.
034F-	90	2E		1680	BCC NOLINE	Line doesn't exist.
0351-	A0	00		1690	LDY #\$0	Zero the Y-register.
0353-	84	08		1700	STY TEMP	and TEMP.
0355-	20	86	03	1710	NXTLIN JSR PRTADDR	Print address of line.
0358-	A9	A0		1720	PRTSPC LDA #\$A0	Print a space.
035A-	20	ED	FD	1730	JSR COUT	
035D-	B1	9B		1740	LDA (LOWTR),Y	Get the next byte in the line
035F-	D0	08		1750	BNE PRINTIT	If it's not zero, print it.
0361-	A5	08		1760	LDA TEMP	It is zero, did we pass
0363-	C9	05		1770	CMP #\$5	the fifth byte?
0365-	B0	0D		1780	BCS DONE	Yes, print it and end up.
0367-	A9	00		1790	LDA #\$0	No, print it and continue.
0369-	20	DA	FD	1800	PRINTIT JSR PRBYTE	Print byte in accumulator.
036C-	20	9F	03	1810	JSR INCR	Increment LOWTR and TEMP.
036F-	CA			1820	DEX	Decrease X by one.
0370-	F0	E3		1830	BEQ NXTLIN	X=0 start a new line.
0372-	D0	E4		1840	BNE PRTSPC	Get and print next byte.
0374-	A9	00		1850	DONE LDA #\$0	The last byte is a zero
0376-	20	DA	FD	1860	JSR PRBYTE	so print it.
0379-	20	8E	FD	1870	JSR CROUT	Print a carriage return.
037C-	4C	69	FF	1880	JMP MONZ	Jump to the monitor.
				1890 *		
				1900 *		
				1910 *	Tell the user the line he requested	
				1920 *	does not exist.	
				1930 *		
037F-	A9	BE		1940	NOLINE LDA #TEXT2	Point to text to be
0381-	A0	03		1950	LDY /TEXT2	printed.
0383-	4C	A8	03	1960	JMP MSGPRT	Print it.
				1970 *		
				1980 *		
				1990 *	This section of the program prints	
				2000 *	out a carriage return, a space and then	
				2010 *	the address in memory of the first byte	
				2020 *	displayed on the line, followed by a	
				2030 *	colon.	
				2040 *		
0386-	20	8E	FD	2050	PRTADDR JSR CROUT	Print a carriage return.
0389-	A9	A0		2060	LDA #\$A0	Print out a space.
038B-	20	ED	FD	2070	JSR COUT	
038E-	A2	08		2080	LDX #\$8	Count 8 bytes per line.
0390-	A5	9C		2090	LDA LOWTR+1	Print out the address of
0392-	20	DA	FD	2100	JSR PRBYTE	the first byte on the
0395-	A5	9B		2110	LDA LOWTR	line, high byte first.
0397-	20	DA	FD	2120	JSR PRBYTE	
039A-	A9	BA		2130	LDA #\$BA	Then print a colon.
039C-	4C	ED	FD	2140	JMP COUT	
				2150 *		
				2160 *		
				2170 *	Here, the pointer to the contents of	
				2180 *	the line is incremented. Location	
				2190 *	TEMP is incremented too.	
				2200 *		
039F-	E6	9B		2210	INCR INC LOWTR	
03A1-	D0	02		2220	BNE INCTEMP	
03A3-	E6	9C		2230	INC LOWTR+1	
03A5-	E6	08		2240	INCTEMP INC TEMP	
03A7-	60			2250	RTS	
				2260 *		
				2270 *		
				2280 *	This is the message printing routine.	
				2290 *		
03A8-	85	18		2300	MSGPRT STA TXTPTR	
03AA-	84	19		2310	STY TXTPTR+1	
03AC-	A0	00		2320	LDY #\$0	
03AE-	B1	18		2330	LOOP LDA (TXTPTR),Y	
03B0-	F0	0B		2340	BEQ ENDPRT	
03B2-	20	ED	FD	2350	JSR COUT	
03B5-	E6	18		2360	INC TXTPTR	
03B7-	D0	F5		2370	BNE LOOP	
03B9-	E6	19		2380	INC TXTPTR+1	
03BB-	D0	F1		2390	BNE LOOP	
03BD-	60			2400	ENDPRT RTS	
				2410 *		
				2420 *		
				2430 *	This is the text that tells the user	
				2440 *	that the requested line doesn't exist	
				2450 *	in the program. A bell is also rung	
				2460 *	to alert the user to the error.	
				2470 *		
				2480	TEXT2 .HS 8D	
03BE-	8D					
03BF-	CE	CF	A0			
03C2-	D3	D5	C3			
03C5-	C8	A0	CC			
03C8-	C9	CE	C5	2490	.AS -"NO SUCH LINE"	
03CB-	87	8D	00	2500	.HS 878D00	

PROTECTION TUTORIAL — Part I

Copy protection wasn't always a problem for Apple // owners. In the early days of computing (the late 1970s) no programs were copy protected and all could be easily backed up by using the COPY or COPYA programs provided on the System Master diskette. However, as time went by, some manufacturers discovered that many more copies of these programs were available than they had produced. In addition, they were getting calls from people with questions from people who never bought the program. To combat this, they developed ways to automatically run programs when they were loaded in and made it difficult to copy those programs with the standard copy programs.

Making it run automatically

It's not as difficult as you might think to make machine language programs run automatically as soon as they are loaded. All you have to do is place the address of the start of your machine language program in locations \$36 and \$37. This should be done by the loading process, so that when the loading is completed and the computer attempts to print out the prompt character, control is automatically transferred to your program. Of course, one of the first things your program should do is restore the proper values to locations \$36 and \$37. Since it's not possible to make the changes to these two locations and save them out to the disk, you must do it somewhere else in memory, save it out to the disk and then change the loading address on the disk itself or BLOAD the program at \$36.

If you have a short machine language program (less than 500 bytes long) you can let it reside in pages 2 and 3 of memory (\$200 to \$3FF). If it's longer than that, you'll have to place it at \$800 and above. In the later case, to retain the auto-run capability, you'll have to save out the screen area as well. Let's see how a program to run automatically. To begin with, we'll first assume that the program will fit in pages 2 and 3 of memory and that its starting address is at \$200. We'll use a simple program that clears the screen and prints an "A" as an example.

Since we already said that we can't assemble our auto-run program in the final location it is going to reside in, let's choose another convenient spot. In this case we'll start with \$836. At \$836 we'll place the address of the start of our program, which we said would be \$200. So from the monitor, type in:

```
836:00 20
```

Now we have to leave the same amount of space between \$836 and the start of our program, as there is between \$36 and \$200. So, our program which should be assembled to operate at \$200, is temporarily stored at \$A00 and here it is:

```
A00:A9 BD 85 36 A9 9E 85 37 20 58 FC
A9 C1 20 ED FD 4C 00 E0
```

Next, we want to transfer the contents of page 1, which is the stack area for the 6502, to its equivalent higher memory location. In order for the 6502 microprocessor to operate properly, it expects certain values in certain locations on the stack. By transferring the stack to higher memory, we can preserve those locations and load them back in when we load our program. We can do this by typing from the monitor:

```
900<100.1FFM
```

Finally, we save the whole thing out to disk as one file by typing:

```
BSAVE TEST,A$836,L$1DD
```

where \$1DD is the length of memory between \$836 and \$A12 which is the end of our program. To automatically run our program, we just have to BLOAD TEST,A\$36. Try it. If you want to eliminate the need to add the A\$36 to the load command, you can use a track and sector editor program to modify the two bytes on the disk that tell DOS where to BLOAD the program. We'll take a look at how to do that next time.

How to make it harder to copy

Until now, we've only talked about auto running machine language programs. While this is an essential part of copy protection, it's not enough. If you want to prevent unauthorized copying of a program, you have to prevent copy programs from reading the disk. One of the earliest ways this was done was to leave one or more tracks on a diskette unformatted. Usually this was track 3. When

ordinary copy programs copy a diskette, they do it one track at a time. After copying DOS onto the new diskette (DOS is on tracks 0, 1 and 2) the program would attempt to copy track three. Since it was unformatted, the program was unable to read it and an I/O ERROR was caused, crashing the copy program. This was a fairly effective method of preventing copying but eventually copy programs came out that either ignored the I/O ERROR or copied only those sectors that were marked as being used on the Volume Table of Contents — VTOC — which is located on track 17 (\$11), sector 0.

Erasing or leaving a track unformatted on a disk is still an effective way to keep people from making casual copies. Although most nibble copy programs (Copy][Plus, Locksmith, Essential Data Duplicator) will easily overcome this obstacle, most Apple // users do not own one of these programs, and rely on COPYA to backup their diskettes.

Those of you who would like to experiment with producing disks with an unformatted track, can do so with the aid of the Applesoft program listed below. This program temporarily modifies DOS 3.3's formatting routine so that it will leave track 3 unformatted. You must use new or erased diskettes because this program simply skips over track three. If you are using a previously formatted diskette, which had track 3 formatted, then it will remain formatted.

A patch to DOS is inserted in line 120 which tells the formatting routine to jump to location \$300 (768) where there is a short routine to check and see if track 3 has been reached yet. If it has, the formatter's track counter is incremented by one so that track 3 will be skipped and not formatted. This is followed by the original four bytes that were removed from the formatter code to make room for the jump-to-patch instructions. Finally, this routine jumps back to the formatter which then finishes its job.

Leaving track 3 unformatted is not enough. If you want to prevent any crashes

continued on page 8

```
10 TEXT : HOME
20 A$ = "SKIP TRACK 3 FORMATTER"
30 PRINT TAB( INT ( LEN (A$) / 2 ));A$
40 PRINT : PRINT : INPUT "ENTER SLOT NUMBER: ";SLT
50 PRINT : INPUT "ENTER DRIVE NUMBER: ";DRV
60 IOB = 47080:TRK = IOB + 4:SCT = IOB + 5
80 FOR X = 768 TO 787
90 READ Y
100 POKE X,Y
110 NEXT X
120 POKE 48891,76: POKE 48892,0: POKE 48893,3
130 PRINT CHR$(4);"INITHELLO,S";SLT;"D",DRV
140 POKE 48891,165: POKE 48892,68: POKE 48893,201
150 POKE TRK,17: POKE SCT,0: POKE IOB + 3,0
160 CMD = 47092: POKE CMD,1: CALL 781
170 BUF = PEEK (47088) + 256 * PEEK (47089)
180 POKE BUF + 68,0: POKE BUF + 69,0: POKE CMD,2
190 CALL 781: POKE CMD,0
200 PRINT : PRINT : INPUT "DO YOU WANT TO FORMAT ANOTHER DISK? ";A$
210 IF LEFT$(A$,1) = "Y" OR LEFT$(A$,1) = "y" THEN RUN
220 DATA 165,68,201,3,208,2,230,68,201,35,76,255,190
230 DATA 160,232,169,183,76,217,3
997 REM
998 REM COPYRIGHT 1985 BY JULES H. GILDER
999 REM ALL RIGHTS RESERVED
```

SOFTWARE PROTECTION TECHNIQUES EXPOSED!

Now, for the first time, owners of Apple // series computers can learn all about the tricks and techniques used to protect Apple software. Apple Software Protection Digest, a new monthly publication, will show you how to protect, unprotect and backup your software.


- Prevent others from accessing your programs
- Make your programs difficult to copy
- Overcome protection schemes on commercial software
- Build a library of protection-oriented utility programs
- Get help with your specific problems
- Learn about the latest advances in protection hardware and software

All this and more can be yours by subscribing to the Apple Software Protection Digest. A one-year subscription is \$24, two years is \$42.

SUBSCRIBE TODAY!

APPLE SOFTWARE PROTECTION DIGEST

Vol. 1, No. 1 Premiere Issue



LEARNING TO LIVE WITH PROTECTION

With the programs that are included in each issue of the digest, you'll build a valuable library of utility software that will make the job of protecting, unprotecting and backing up software easier. And, on-going tutorials will keep you up-to-date on both simple and sophisticated protection techniques. In addition to all this, every month you will get reviews of hardware, software and books that are of particular importance to the software protection field.

We do not advocate software piracy, because we firmly believe that in the long-run piracy only leads to more expensive and lower quality programs and less user support. Programmers work long, hard hours to get their software working and they deserve to be compensated fairly for it. This cannot happen if software is stolen. On the other hand, the honest consumer should not be penalized and limited in his or her application of a particular program simply because the publisher decided to protect it. You may have a legitimate need to go back to a program and we hope to supply you with the knowledge you need to do that. Alternatively, you may wish to protect a program that you've written so that others can't copy it. We'll show you how to do that too.

Apple Software Protection Digest welcomes your comments, tips and article contributions. If you have a problem backing up a particular program, let us know, we'll try to help. If you've discovered a way to copy or crack (unprotect) a particular program, let us know about that too. Most likely there are other people who would like to know how to do it also. Finally, if you've come up with an ingenious hardware or software oriented protection scheme, write to us about it so that we can share it with others. If your article is used, you'll receive a free 6-month subscription (or subscription extension) to our publication. Let's hear from you soon.

Julian H. Gilder
Editor & Publisher

Contents

EDITORIAL	1
HIDING APPLESOFT	2
PROGRAM LINES	2
APPROXIMATELY FINDER	3
PROTECTION TUTORIAL - Part 1	4
Auto-Running Programs	6
Making Them Hard to Copy	6
Ship Track 3 Formatter Programs	8
Adding Extra Tracks	8
BACKING UP PRINT SHOP	10
Print Shop Copy Program	10
REVIEW: Copy II Plus	11
COMING NEXT ISSUE	11

Apple Software Protection Digest: Publisher & Editor, Julian H. Gilder; Contributing Editor, J. Scott Barnes; Copyright © 1985 by Redlig Systems, Inc., 2068 - 79th Street, Brooklyn, NY 11214. All rights reserved. No part of this publication may be reproduced, in electronically transmitted or stored without the publisher's written permission. Published monthly at \$24 per year by Redlig Systems, Inc. (718) 232-8429. Reprints of prior issues available for \$5 each.

Apple is a registered trademark of Apple Computer, Inc.

REDLIG SYSTEMS, INC., Dept. A1357
2068 - 79th St., Brooklyn, NY 11214

Please enter my _____ year subscription to Apple Software Protection Digest.

☐ Enclosed is my check for _____

☐ Please charge my credit card: ☐ VISA ☐ MasterCard ☐ American Express

Card Number _____ Exp. Date _____

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

Protection Tutorial

continued from page 6

caused by DOS's attempt to store something on track 3, you must tell DOS that track 3 is not available for storage. This can be done by modifying the appropriate bytes (\$44 and \$45) on track 17 (\$11) sector 0. This sector has a special name. It's called the Volume Table of Contents, or VTOC for short and it keeps a record of which tracks are free and which aren't. By changing bytes \$44 and \$45 on this sector to zeroes, we effectively notify DOS that track 3 cannot be used to store anything. This is done by the code starting at line 150. Line 150 sets up the track and sector, while line 160 sets up the input/output block (IOB) that the next disk operation we do is going to be a read from the disk. The actual disk access is done by a short machine language routine located at 781 (\$30D). This program uses DOS's internal read or write a track or sector (RWTS) routine. Once the sector has been read and is stored in memory, line 180 changes the appropriate bytes to zero and sets up the RWTS so that it will write the sector back out to the diskette. This is done in line 190, after which the RWTS command is changed from a write command to a null command.

Add extra tracks to your disks

Another method that is used to prevent people from backing up programs is to add

extra tracks to your diskette. While the normal Apple DOS 3.3 diskette is formatted for 35 tracks (numbered 0 through 34) most people are not aware that it is possible to format a diskette with more tracks. In the early days, a single extra track was added and critical information was stored on this track. Since normal copy programs would only copy 35 tracks, the copy wouldn't work because the critical 36th track was missing. However, it didn't take long for the nibble copy programs to incorporate a 36th track capability into their programs. This coupled with the fact that early Apple drives had difficulty accessing the extra track, gradually caused this protection technique to fall out of favor. It is well worth considering again today, however, because drives that are currently available can format a diskette with as many as 40 tracks on them. In addition to giving you extra storage space on the same diskette, you should know that today's nibble copiers still only go up to 36 tracks.

To be safe, and minimize problems with other drives, you ought to consider adding only two tracks to the normal diskette (for a total of 37). This will give you an extra 8K of storage space on the diskette and defeat most copy programs currently available. Making diskettes with the additional tracks is easy. Here's how:

- 1) First boot DOS 3.3 as usual.
- 2) Next, enter the monitor by typing CALL -151

- 3) Change location \$AEB5 from \$8C to \$90 to add one track or 94 to add two tracks
- 4) Change location \$B3EF from \$23 to either \$24 or \$25 (one or two extra tracks)
- 5) Make the same change to location \$BEFE.
- 6) INIT a blank disk with the newly modified DOS. It will now have 1 or 2 extra tracks on it.

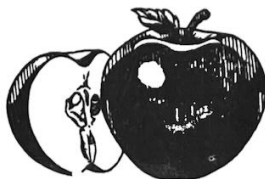
Putting the extra tracks on the disk is not enough. You must now tell DOS that they're available for use. To do this you must change track 17, sector 0 (the VTOC). To do this use a track and sector editor (such as the one found in Copy II Plus) and read in track 17 (\$11), sector 0. Next, change byte number \$34 from \$23 to \$24 or \$25 — depending on how many tracks you've added. If you've added just one track, change bytes numbered \$C4 and \$C5 from 0 to \$FF. If you've added two tracks, change bytes \$C6 and \$C7 to \$FFs also. Now, just write the sector back out to the disk. That's all there is to it. If you check the disk space on this diskette with FID, you'll find that you now have 32 more sectors available.

Next Issue: All About Modified Disk Formats and RWTS

Applesoft Line Finder

continued from page 5

To use APPLESOFT LINE FINDER, just type in an ampersand, followed by the line number like this, &10. This will cause line 10 of the current Applesoft program to be displayed on the screen in hexadecimal form and leave you in the monitor mode so that changes can be made to it. Since a colon is used to separate the address from the displayed data, it is only necessary to move your cursor up to the line that is going to be changed and copy everything with the right arrow key except those items that are going to be modified. It couldn't be simpler.



*Take out a 2-year subscription
to Apple Software Protection Digest
and save over 12%.*

FREE BONUS!

*If you subscribe for 2 years
before October 1985
we'll send you our
Programmer's Number Conversion System
FREE.*

BECOME AN ASSEMBLY LANGUAGE PROGRAMMING WHIZ

You've spent a lot of time learning Apple assembly language and finally know the difference between BEQ and BCS. Now it's time to put your new-found knowledge to work. Time to throw away your Applesoft programming manual and write programs that make your Apple work like a super-charged, super-fast computer. Time to graduate from the Applesoft BASIC used by beginners, to the 6502 assembly language used by professionals.

To help make this transition, you need an experienced programmer to guide you. You need to develop a library of subroutines that make programming in assembly language as easy as programming in BASIC. You need to learn all the tricks that take experienced assembly language programmers years to acquire. Most important of all, you need the book, "*Now That You Know Apple Assembly Language: What Can You Do With It?*" because it contains all this information and more.

It shows you how, step-by-step

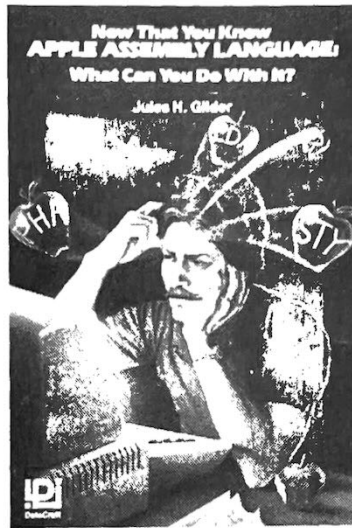
"*Now That You Know Apple Assembly Language: What Can You Do With It?*" will take you step-by-step through the assembly language programming experience. You'll delve into the mysteries of the 6502 stack and learn how to use it to increase the power and versatility of your programs. You'll also learn how to use the Apple's built-in routines to minimize the amount of coding you must do.

Control the output and the input

Frequently it's desirable to gain total control of the computer's output. This book shows you how to *steal control away from the Apple's normal output routines and redirect it to your own program*. Thus if you wanted, you could see the normally invisible control characters, display text on your screen as black on white instead of the normal white on black, format text sent to a printer into pages and much more.

Expand the power of your Apple by *stealing control away from the normal input routines*. Do things like adding a screen print capability, or *convert part of the normal keyboard into a numeric keypad*. It's even possible to *produce self-modifying programs by EXECing in commands from RAM instead of from the disk drive*. Think about the possibilities that offers for protecting your programs. When you want to go back to Applesoft programming, *you'll be able to do it faster with the aid of Applesoft Shorthand*, an assembly language program that types in one or more Applesoft commands at the press of a key, or use another program in the book to *automatically count the number of lines in your Applesoft program*.

With this book you'll also learn about *generating tones and how to figure out the frequency, producing sound effects, teaching your Apple to send Morse code, restoring accidentally erased Applesoft programs, adding new commands to Applesoft and running two Applesoft programs in memory together, to name a few*.



Everything is explained

Unlike other books that merely consist of a collection of programs, this one explains what's happening, where and why. You get detailed descriptions of how the programs work and detailed program listings with virtually every line of code explained. Nothing is left to chance or misinterpretation.

Order now, get 2 FREE gifts

The book costs only \$19.95 plus \$2 for shipping and handling. Order now and you'll also get a **FREE Programmer's Number Conversion System** that makes it easy to convert between binary,

hexadecimal and decimal numbers. No calculators are required. You'll convert numbers almost instantly and wonder how you ever got along without it.

As an extra bonus for prompt ordering, you'll receive a **FREE coupon worth \$5 off** the price of a disk with all the assembled programs on it or a disk that contains the source code. These disks normally sell for \$15 each. We're offering these **FREE** gifts for a limited time only, so hurry! **Order today!**

Money-back guarantee*

We're so confident that you'll find this book invaluable and want it in your library, that we're offering a 10-day, no-questions-asked, money-back guarantee. Order the book. Read it and try the programs for ten days. At the end of ten days if you don't think it's worth every penny you paid for it, just send it back in resalable condition and we'll refund your money immediately, no questions asked.

Redlig Systems, Inc., Dept. A 9783
2068—79th St., Brooklyn, NY 11214

Please rush me _____ copies of "**Now That You Know Apple Assembly Language: What Can You Do With It?**" at \$19.95 each plus \$2 shipping and handling. I understand that if I am not delighted with the book I may return it within 10 days for a prompt and courteous refund. In any case, the Programmer's Number Conversion System and \$5 coupon are mine to keep.

☐ Enclosed is my check for \$ _____

Please charge my credit card:

☐ American Express ☐ MasterCard ☐ Visa

Card No. _____ Exp. _____

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

*NOTE: Shipping and handling fees are not refundable.

BACKING UP THE PRINT SHOP

One of the more popular programs available for the Apple II is *The Print Shop* from Broderbund Software. This program lets your dot matrix printer produce high quality letterheads, signs and greeting cards with nice high-resolution graphic pictures. Like most software from Broderbund, this program has a copy protection scheme on the diskette that makes it difficult to produce backup copies, although there is provision for producing one backup. This is done if the user presses the ESC key while the program is booting.

In general, most of the diskette is formatted fairly normally, with the exception being that Broderbund has placed the VTOC on track 17, sector 2 instead of sector zero. Also, DOS is stored in slightly different locations on this disk, but that's not really critical. What is critical is that the 35th track (we start from zero so it's labelled track 34) is not written in a standard format and is not copyable by standard copy programs. It turns out, that the major problem that prevents *The Print Shop* from being copyable is a nibble counting routine that is located in a file called MENULIB. If the subroutine jump to the nibble count routine is disabled by replacing the JSR with NOP codes, the rest of the disk is copied, and track 17, sector 2 is stored in sector 0 (where the VTOC should normally be located), the disk will run perfectly.

The Applesoft program that follows should be added on to Apple's COPYA program. This can be done by first loading in COPYA and then typing these lines in, or more conveniently, this program can be entered via a word processor and stored in a text (T) file. From the text file, it can be EXECed into memory once COPYA has been loaded.

As a matter of policy, when copying protected programs, I generally eliminate the disk error routine by using the first two POKES in line 75. In this case, however, it's not really necessary, because the one track that gives us problems has been eliminated by the third POKES in line 75. Thus the disk with the copy on it will only have the first 34 tracks of the original disk on it. Since the copy program initializes all 35 tracks on the blank, however, the last track will be initialized and will not cause any problems when you want to make a copy of the copy.

Line 80 changes the title of the program that is displayed and line 225, 290 and 295 hook the program additions into the regular COPYA program. Lines 300 and 305 are not needed, so they should be eliminated from COPYA. A call to DOS' RWTS routine is set up in lines 400 to 460, while line 470 reads track 17, sector 2 into memory. Once in memory, one byte in this sector must be changed before it is written back out to the disk on sector 0. The last byte in the sec-

tor must be changed from a zero to a one. This is done in line 480 and the sector is then written back out to the disk. Once this operation is complete, DOS' file handler can now access the files on the disk without requiring a modification.

You may recall that I mentioned earlier that if the ESC key is pressed while the program is booting, that control is passed to The Print Shop's copy routine. Since this routine checks track 34 to see if the one permitted backup copy has already been made, and we have eliminated track 34 from the copy disk, pressing ESC during a boot could cause the program to hang up. To eliminate this, we have to eliminate the code that checks for the ESC key being pressed. Since any check of this sort would require a CMP #9B instruction, a disk scanning utility (such as the one in Copy II Plus) was used to locate the byte sequence C9 9B on the disk. It turns out that

it is in three places: track 0, sector 5; track 0, sector 10 and track 6, sector 14. In lines 490 to 510, these sectors are loaded into memory. If the correct byte sequence is found (it may be located somewhere else on other versions of the program) these two bytes are replaced with two other bytes that prevent the jump to the copy program. If the byte sequence is not located, nothing is done.

Next, the MENULIB file is loaded into memory and a check is made for the presence of the nibble count JSR (lines 530 and 540). If its not where it's supposed to be, the user is notified that this version of the program cannot be copied and execution is terminated. If it is there, however, the JSR is eliminated (line 550) and the modified file is stored back out onto the disk (lines 560 and 570). That's all there is to it. Copies made with this program can be backed up with the normal, unmodified, COPYA program.

```

75 POKE 47426,24: POKE 929,24: POKE 863,34
80 HOME : PRINT "      PRINT SHOP DUPLICATION PROGRAM": PRINT
   : PRINT
225 VTAB 5: HTAB 24: PRINT "      ": IF PEEK (713) = 1
   THEN 295
290 GOTO 600
295 VTAB 19: GOTO 400
300
305
400 IOB = 47080:TRK = IOB + 4:SCT = IOB + 5:CMD = IOB + 12
   :RD = 1:WR = 2
410 BUF = PEEK (IOB + 8) + 256 * PEEK (IOB + 9)
420 FOR X = 768 TO 774
430   READ Y: POKE X,Y
440 NEXT X
450 PRINT : PRINT "UNPROTECTING COPY"
460 POKE IOB + 3,0
470 POKE TRK,17: POKE SCT,2: POKE CMD,RD: CALL 768
480 POKE BUF + 255,1: POKE SCT,0: POKE CMD,WR: CALL 768
490 POKE TRK,0: POKE SCT,5: POKE CMD,RD: CALL 768: IF PEEK
   (BUF + 57) = 201 THEN POKE BUF + 57,169: POKE BUF + 58,1
   : POKE CMD,WR: CALL 768
500 POKE TRK,0: POKE SCT,10: POKE CMD,RD: CALL 768: IF PEEK
   (BUF + 55) = 201 THEN POKE BUF + 55,169: POKE BUF + 56,1
   : POKE CMD,WR: CALL 768
510 POKE TRK,6: POKE SCT,14: POKE CMD,RD: CALL 768: IF PEEK
   (BUF + 68) = 201 THEN POKE BUF + 68,169: POKE BUF + 69,1
   : POKE CMD,WR: CALL 768: POKE CMD,0
520 PRINT CHR$ (4);"BLOAD MENULIB"
530 CK = PEEK (30727) + 256 * PEEK (30728)
540 IF CK < > 36311 THEN 590
550 POKE 30726,234: POKE 30727,234: POKE 30728,234
560 S = PEEK (43634) + 256 * PEEK (43635):L = PEEK (43616)
   + 256 * PEEK (43617)
570 PRINT CHR$ (4);"BSAVE MENULIB,A";S;";L";L
580 TEXT : HOME : VTAB 5: PRINT "RE-RUN 'PRINT SHOP COPY' TO
   MAKE      ANOTHER COPY OF THE PROGRAM":PRINT CHR$
   (4);"FP"
590 PRINT : PRINT : PRINT CHR$ (7);"THIS VERSION OF PRINT
   SHOP CANNOT BE      COPIED WITH THIS PROGRAM."
600 PRINT CHR$ (4);"FP"
610 DATA 160,232,169,183,76,217,3
990 REM
995 REM COPYRIGHT 1985 BY JULES H. GILDER
999 REM ALL RIGHTS RESERVED

```


REVIEW: Copy II Plus

One of the handiest tools that any Apple // owner can possess is this program. Originally designed as just a nibble copy program, over the years, it has developed into a full-blown back-up and disk repair tool with an impressive set of DOS utilities. To give you an idea of just how powerful this program is, here is a list some of the things you can do with it:

- Copy any 16- or 13-sector unprotected disk
- Copy just DOS onto a disk
- Copy individual files
- Catalog a disk
- Show file lengths on the catalog
- Show control characters on the catalog
- Catalog deleted files
- Delete files
- Delete DOS to get more file space
- Lock or unlock files
- Rename files
- Alphabetize the catalog
- Format a disk
- Verify that the disk is good
- Verify that files are good
- Verify whether or not two files are identical
- Check disk drive speed
- View the contents of files
- See a map of what files are stored on the disk and where
- Edit any sector or any file
- Fix file sizes to free up wasted disk space
- Change the boot program on the disk
- Recover files that were accidentally erased by undeleting them

As you can see, the list is a long and impressive one. And the best part of all, is that

the program only costs \$39.95. While the program is mainly designed to work with DOS 3.3 and DOS 3.2 formatted diskettes, the disk copy and verify functions, as well as the sector editor, can be used with any unprotected 16- or 13-sector diskette, including ProDOS, SOS, CP/M and Pascal formats. While we obviously can't get into a detailed review of every single function of this powerful program, we will take a look at a few of the more interesting features from a software protection point of view.

To begin with, the designers of Copy II Plus Version 5.X (5.5 is the latest current version) did their homework. They realized that people want a powerful tool, but that it also must be easy to operate. It is. Central Point Software has included an Auto Copy feature into the program. When you select the bit copier mode of operation, the Copy II Plus asks you for the name of the program you want to copy. It then searches its internal list of programs and copy parameters to see if it already "knows" how to copy the program. If it does, all you have to do is press RETURN and you're off. Sometimes there are several versions of the same program, each with its own unique copy protection scheme. In that case, the program will list either an alternate, or specify a way of identifying your particular version (e.g. new ProDOS version or old ProDOS version). To choose the one you want, simply use the arrow keys to move the cursor to your choice, and press RETURN.

With over 500 entries on the latest version of the program, the list is fairly substantial. Nevertheless, you'll still find plenty of programs that have not yet made it to the list. But don't worry. Central Point has taken care of that too. About every three months,

the company comes out with a parameter update list. If you're a registered owner (you sent your card back) you can get this update for only \$5. Believe me, it's worth it. Of course, if the program you wish to copy is not included in the parameter list of Copy II Plus, you can still copy it manually, but this requires a lot more skill.

Another outstanding feature of Copy II Plus is one of the functions that is available from the sector editor operating mode: the disk scanning feature. This function will let you scan the entire diskette, or any part of it, a any series of bytes. You may enter the search text in hexadecimal, or if it is a word or phrase, you can enter the text directly. The search is quick and easy to use and is extremely helpful when you're trying to make back-up copies of disks for which you have no copy parameters. This feature in fact, was used to develop the Print Shop Copy program listed elsewhere in this issue.

While Central Point Software has done an outstanding job in developing the software, they weren't quite as good with the manual. Don't get me wrong, the manual is quite good and comprehensive. It's neatly and conveniently produced as a softcover paperback. The usefulness of the book, however, has been severely limited by the fact that it has no index. With many questions bound to arise during the use of this program, a detailed index would have been an invaluable aid. Without it, the user is forced to frequently thumb through large sections of the book until what he's looking for is found. Let's hope they fix this one major flaw to an otherwise superb product.

Price: \$39.95. **Source:** Central Point Software, Inc., 9700 SW Capitol Hwy., #100, Portland, OR 97219. **Call:** (503) 244-5782

COMING NEXT ISSUE

How to Back-Up Sensible Speller

How to Back-Up The Newsroom

How to Back-Up Wizardry

Using RAM Cards as a Back-Up Tool

Moving the CATALOG to Other Tracks

Protection Tutorial — Part II:

Modified Disk Formats and RWTS

Using COPYA to Copy Protected Disks

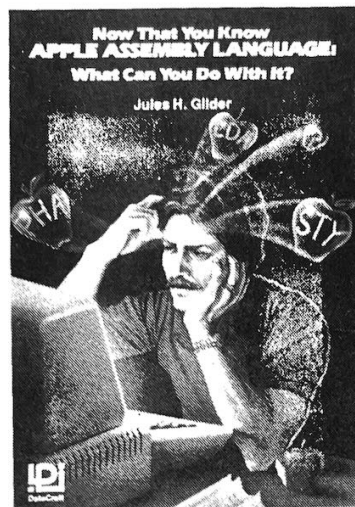
Changing the BLOAD Address of

Machine Language Programs

A Handy HEX/DEC/HEX Converter

Review: Locksmith 5.0

BECOME AN ASSEMBLY LANGUAGE PROGRAMMING WHIZ



You've spent a lot of time learning Apple assembly language and finally know the difference between BEQ and BCS. Now it's time to put your new-found knowledge to work. Time to throw away your Applesoft programming manual and write programs that make your Apple work like a super-charged, super-fast computer. Time to graduate from the Applesoft BASIC used by beginners, to the 6502 assembly language used by professionals.

To help make this transition, you need an experienced programmer to guide you. You need to develop a library of subroutines that make programming in assembly language as easy as programming in BASIC. You need to learn all the tricks that take experienced assembly language programmers years to acquire. Most important of all, you need the book, "Now that You Know Apple Assembly Language: What Can You Do With It?" because it contains all this information and more.

It shows you how, step-by-step

"Now That You Know Apple Assembly Language: What Can You Do With It?" will take you step-by-step through the assembly language programming experience. You'll delve into the mysteries of the 6502 stack and learn how to use it to increase the power and versatility of your programs. You'll also learn how to use the Apple's built-in routines to minimize the amount of coding you must do.

Control the output and the input

Frequently it's desirable to gain total control of the computer's output. This book shows you how to *steal control away from the Apple's normal output routines and redirect it to your own program*. Thus if you wanted, you could see the normally invisible control characters, display text on your screen as black on white instead of the normal white on black, format text sent to a printer into pages and much more.

Expand the power of your Apple by *stealing control away from the normal input routines*. Do things like adding a screen print capability, or *convert part of the normal keyboard into a numeric keypad*. It's even possible to *produce self-modifying programs* by EXECing in commands from RAM instead of from the disk drive. Think about the possibilities that offers for protecting your programs. When you want to go back to Applesoft programming, *you'll be able to do it faster with the aid of Applesoft Shorthand*, an assembly language program that types in one or more Applesoft commands at the press of a key, or use another program in the book to *automatically count the number of lines in your Applesoft program*.

With this book you'll also learn about *generating tones and how to figure out the frequency, producing sound effects, teaching your Apple to send Morse code, restoring accidentally erased Applesoft programs, adding new commands to Applesoft and running two Applesoft programs in memory together*, to name a few.

Everything is explained

Unlike other books that merely consist of a collection of programs, this one explains what's happening, where and why. You get detailed descriptions of how the programs work and detailed program listings with virtually every line of code explained. Nothing is left to chance or misinterpretation.

Order now, get 2 FREE gifts

The book costs only \$19.95 plus \$2 for shipping and handling. Order now and you'll also get a *FREE Programmer's Number Conversion System* that makes it easy to convert between binary,

hexadecimal and decimal numbers. No calculators are required. You'll convert numbers almost instantly and wonder how you ever got along without it.

As an extra bonus for prompt ordering, you'll receive a *FREE coupon worth \$5 off the price of a disk* with all the assembled programs on it or a disk that contains the source code. These disks normally sell for \$15 each. We're offering these FREE gifts for a limited time only, so hurry! *Order today!*

Money-back guarantee*

We're so confident that you'll find this book invaluable and want it in your library, that we're offering a 10-day, no-questions-asked, money-back guarantee. Order the book. Read it and try the programs for ten days. At the end of ten days if you don't think it's worth every penny you paid for it, just send it back in resalable condition and we'll refund your money immediately, no questions asked.

Redlig Systems, Inc., Dept. A 9783
2068—79th St., Brooklyn, NY 11214

Please rush me _____ copies of "Now That You Know Apple Assembly Language: What Can You Do With It?" at \$19.95 each plus \$2 shipping and handling. I understand that if I am not delighted with the book I may return it within 10 days for a prompt and courteous refund. In any case, the Programmer's Number Conversion System and \$5 coupon are mine to keep.

☐ Enclosed is my check for \$ _____

Please charge my credit card:

☐ American Express ☐ MasterCard ☐ Visa

Card No. _____ Exp. _____

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

*NOTE: Shipping and handling fees are not refundable.