Open-Apple™

Releasing the power to everyone.

June 1985
Vol. 1, No. 5

Up where Applesoft lives

Hiking boots and backpacks, please. Fill your water bottles. This month we're going on a trek to the high reaches of your Apple's memory. There have been some changes up where Applesoft lives and an expedition is in order.

First let's get our bearings. The machine language instructions that tell your Apple's microprocessor everything it knows about Applesoft appear in memory from byte $D000 (53248) to byte $F7FF (63487). The 10,240 bytes of code in this area first appeared in 1979, when the Apple II-Plus was introduced. They remained untouched—totally unchanged—for five years. Then the Apple IIc appeared in 1984 and suddenly there were differences. More changes showed up in the enhanced IIe, which Apple introduced in March.

On this month's outing we are going to take a close look at these three versions of Applesoft and determine exactly what the differences are. It isn't necessary that you have your own maps of the territory—there is little doubt that we will return home safely—nonetheless, if you'd like to make some, here's how.

Basic map making. To make maps, you'll need to get momentary access to the two types of computers you don't already have. If you have a IIe, for example, you'll need to momentarily borrow the neighbor's IIc and the school's new enhanced IIe. Boot the borrowed computer with standard DOS 3.3. *Don't use ProDOS* and don't use a version of DOS 3.3 that moves itself to the upper reaches of memory. Then simply take a snapshot of Dr. Basic by BSAVEing the image of Applesoft in the borrowed machine. Here's how:

```
BSAVE IIC APPLESOFT IMAGE, A$D000, L$2800
```

The reason you don't want to boot with ProDOS (or a version of DOS 3.3 that moves itself) is that it will spoil your picture. Instead of Dr. Basic's gruff visage you'll get a smiling image of Uncle DOS.

If you have a IIc or an enhanced IIe and a DOS 3.3 System Master disk, you don't even have to borrow a II-Plus or IIe. The original image of Applesoft appears on the System Master disk in a file called FPBASIC. (On old System Master disks, this file is 50 sectors long—it includes an image of the Monitor as well as Applesoft. On newer System Masters, it is 42 sectors long and includes only Applesoft.)

Once you have your images, you make maps like this:

```
]BLOAD IIC APPLESOFT IMAGE, A$2000     (Load image at $2000.)
]CALL -151                             (Enter Monitor.)

*2000<D000.D010V                       (Verify that the first few bytes
                                          are identical.)
*PR#1                                  (Turn on printer.)
*2000<D000.F7FFV                       (Use the Monitor's Verify command
                                          to get a list of the differences
                                          between the image at $2000 and
                                          the one built into your Apple.)
```

If you get a list of discrepancies from the first Verify command, you've done something wrong—don't print or you'll waste a ton of paper. Just start over.

The second Verify command will send a listing to your printer of the differences between the borrowed Applesoft image you have loaded at $2000 and the Applesoft image built into your machine. The listing will show you the address of each discrepancy, the value at that byte in your built-in Applesoft, and, in parentheses, the value at that byte in the image loaded at $2000. For example, if you do this on a IIc with the FPBASIC file from a System Master disk, your listing will begin like this:

```
2000<D000.F7FFV

D034-F4 (74)
D035-03 (F7)
D04E-F4 (BB)
D04F-03 (F3)
```

The listing will go on for several pages. There are 384 bytes of differences (6 printed pages) between FPBASIC and IIc Basic; 326 bytes of differences (6 pages) between the enhanced IIe and the IIc versions; and 179 bytes of differences (3 pages) between the enhanced IIe version and the original FPBASIC. As a percentage, the differences between the three versions are quite minor—at most, less than 4 per cent of the original Applesoft image is changed.

Ampersand valley. As we begin our little amble, the first features we come upon are those between $D034 and $D06F, as shown above. (In assembly language language, we say those addresses are on "page" $D0. In this way of looking at things, the Apple II has 256 pages of memory, $00 through $FF, and each page holds 256 bytes. The first two hexadecimal digits of an address point to the page, the last two to the byte.) This particular area of Applesoft is a table of addresses. The addresses point to the machine language routines that actually execute the various Applesoft commands.

At the very back of Saveload Cave there is some strange new code.

The changes here affect the commands SHLOAD, RECALL, STORE, LOAD, and SAVE. The thing these commands have in common are that they are all *cassette tape* commands. You are probably more familiar with LOAD, SAVE, and Basic.system's STORE as DOS commands. When you put a *filename* after these commands, Uncle DOS recognizes and executes them. If you leave off the filename, Uncle DOS passes the command on to Dr. Basic, who starts fiddling with the cassette tape jacks.

Since no one in their right mind uses cassette tape for storage anymore, these commands are pretty useless. Worse than useless, really, because when you execute them by mistake (by forgetting to put a filename after a DOS command), your Apple will appear to hang. It's not really hung, of course (Dr. Basic is purposefully fiddling), but it might as well be since you have to press Reset to recover.

The IIc doesn't have cassette tape jacks, so the code that executes these commands was removed from the IIc. As we'll see later, the space was used for something else. However, though the code that executes the commands was removed, the commands themselves are still there. The changes on page $D0 point all five of them to the execution address of the ampersand

(&) command. This gives assembly language programmers the possibility of easily implementing six ampersand commands on the IIc. (To discern between the six, an ampersand routine can refer to bytes $B8-$B9; they point one byte beyond the token of the command that caused the ampersand routine to be executed. The tokens you will find are:

```
A token table (for assembly language programmers only)

    SHLOAD  $9A        LOAD  $B6
    RECALL  $A7        SAVE  $B7
    STORE   $A8        &     $AF )
```

The enhanced IIe continues to support cassette tape, so there are no changes in this area in an enhanced IIe's Applesoft.

Lower-case Creek. The next discrepancies are on pages $D5, $D6, and $D7. Scattered through this area are 21 bytes of differences. This part of Applesoft contains the routines that *parse* (try to make sense of) your commands. Also here are the routines that create program listings in response to the LIST command. Most of the changes in this area consist of hooks inserted into the command parser that cause new sections of code, stored elsewhere, to be executed during the parsing process. This new code allows lower-case input.

The changes are pretty intelligent. If you enter programs in lower case on a IIc or an enhanced IIe, you'll find when you LIST the program that commands and variable names have been converted to upper case. However, stuff inside quotation marks, REM statements, and DATA statements will not be. For example:

```
]10 print "Open-Apple V:";v;" N:";n : data June 1985 : rem issue date

]list

 10 PRINT "Open-Apple V:";V;" N:";N : DATA June 1985 : REM issue date
```

This sample also illustrates the effect of the other changes in this area, but they are more subtle. One of the patches causes program listings to begin with a blank space. This makes "escape mode editing" much easier, since it is no longer necessary to backspace over the line number. (Escape mode editing involves pressing the escape key, followed by either the arrows or the letters in the I-J-K-M diamond, to move the cursor to the beginning of the Basic line you want to edit, pressing the space bar, and then using the right arrow to re-enter the line.)

The final change allows Basic program listings to be 80 columns wide when the Apple 80-column mode is turned on.

Saveload Cave. The next changes occur on pages $D8 and $D9. In the original Applesoft, this area contained the routines that executed the cassette SAVE and LOAD commands. On the enhanced IIe this area is unchanged. On the IIc, however, the cassette routines have been removed. In their place are the routines that perform lower-case and LIST conversions. The hooks we looked at earlier on pages $D5 through $D7 all point here on a IIc.

At the very back of Saveload Cave, from $D8DD to $D911, there is some strange new code. It looks kind of scary, so let's move right along....

Commatab Ridge. Proceeding onward, we next come to a series of changes on pages $DA and $DB. This area of Applesoft has the code that executes PRINT statements.

Under the PRINT umbrella are three functions for formatting output that lots of people forget about. They are TAB(, SPC(, and comma tabbing. Not be confused with HTAB and VTAB, these functions can only be used *within* a PRINT statement.

The changes in this area of Applesoft fix these functions so they work on an 80-column screen. On an 80-column IIc or enhanced IIe, each of the following lines does the same thing:

```
100 PRINT "THIS", "IS", "A", "TEST."
110 PRINT "THIS"; SPC(12); "IS"; SPC(14); "A"; SPC(15); "TEST."
120 PRINT "THIS"; TAB(17); "IS"; TAB(33); "A"; TAB(49); "TEST."

RUN
THIS            IS              A               TEST.
THIS            IS              A               TEST.
THIS            IS              A               TEST.
```

Comma tabbing means using commas, rather than the usual semicolons, between items in a PRINT statement. The items will be printed beginning in columns 1, 17, 33, 49, and 65, as long as each item is less than 16 characters long. Longer items cause columns to be skipped:

```
100 PRINT "THIS IS TEST NO",2
110 PRINT "THIS IS TEST NO.",2
RUN

THIS IS TEST NO 2
THIS IS TEST NO.                2
```

The *SPC(* function prints the specified number of spaces. Note that the value inside the parentheses can be a variable or a formula, as well as a number. The *TAB(* function is similar, but it moves the output position to a specified position on the output line.

Double-Low Forest. We are now approaching the most mysterious part of our expedition. There are a number of changes upcoming in the IIc version of Applesoft that, to the best of my knowledge, have never been documented *anywhere*. Not by Apple itself, not by other writers. First though, climb up this small hill here and look ahead to byte $E006.

The seven bytes at $E000 to $E006 contain the start-up addresses for Basic and an identification byte. A call to 57344 ($E000) coldstarts Basic (this erases any program in memory). A call to 57347 ($E003) warmstarts it. At 57350 ($E006) there is a version identification byte that is never executed or otherwise used for anything. Its contents are:

```
Contents of byte 57350 ($E006)

II-Plus and original IIe        0   ($00)
IIc                           137   ($89)
enhanced IIe                  196   ($C4)
```

Now just in front of us, at bytes $DF03 and $DF04, is a change that causes the IIc's SCRN function to execute some new code stored elsewhere. SCRN is one of Applesoft's low-resolution graphic functions—it returns the screen color at a specified coordinate. Up ahead, on pages $F1, $F2, and $F3, are a number of other changes, in the IIc version only, that point to new code. These pages hold the code that executes the low-res GR, PLOT, HLIN, and VLIN commands. Have you ever heard anything strange and wonderful about the IIc's low-resolution graphics commands? Let's study these things more closely on our return trip.

Storerecall Cave. Forging ahead, we next come to Storerecall Cave, also in page $F3. This is where the cassette tape commands STORE and RECALL live in the original and enhanced IIe versions of Applesoft. In the IIc version, however; well, poke your head in—you'll see that this is where someone has stored all the new code that the low-res commands now use.

Shloadhtab Cave. The final area of differences is on page $F7. The first part of this area is home to the cassette command SHLOAD in the original II-Plus Applesoft. The last 25 bytes or so belong to HTAB.

What can be confusing is that there are three "caves" in the IIc version of Applesoft, all formed by removing the code that executes cassette tape commands—but there is only one cave on the enhanced IIe. You may wonder how even this cave was hollowed out, since the enhanced IIe still supports the SHLOAD command.

A close examination of the machine language code at $F775-F786 reveals that SHLOAD was pushed down a rabbit hole. It now lives below ground in the area where the original IIe had diagnostic routines (invoked by pressing solid-apple/control/reset). The enhanced IIe still does diagnostics, but not as many as the original IIe.

This rabbit hole is an underground version of pages $C1 through $C7, the area where the firmware on peripheral cards normally appears. SHLOAD itself now starts at $C500. To access this area, you have to throw the "SLOTCXROM" softswitches. A poke to $C007 (49159) lets you see the underground ROM in this address range, a poke to $C006 (49158) lets you see what's on peripheral cards. (By the way, Table 6-5 in Apple's IIe reference manual, page 133, has these softswitches backwards.)

Come back to the main level now and walk all the way back to the end of this cave. Here, at the very top of Applesoft, we find some changes to the HTAB command. The changes make HTAB work correctly in 80-column mode.

An extremely interesting feature of the HTAB modifications is right at the end of Applesoft—byte $F7FE. On the IIc, this byte holds the code for the assembly language command "branch always." This is one of the new commands only available on the 65C02 microprocessor—the earlier 6502 doesn't support it. Of course, the 65C02 is built into the IIc, thus you would expect it to use some of the new commands. The strange part is that on the enhanced IIe, this byte holds the command for "branch on not equal," which also happens to work here. But this is an old instruction—available in the standard 6502 command set—even though the enhanced IIe also has the newer 65C02.

Thinking back, however, you'll be hardpressed to remember seeing *any* of the new 65C02 commands in enhanced IIe Applesoft. That's because there aren't any. I've been told there aren't any in the rest of the enhanced IIe firmware, either, though I haven't checked that out myself. If Apple doesn't use the new 65C02 commands in the software that's built into the machine, who do you suppose *will* use them?

Let's go back now to the front of this cave. With SHLOAD out of the way—simply gone on the IIc and down the rabbit hole on the enhanced IIe—we find the cave filled with different things depending on which computer we're looking at. The enhanced IIe has all the lower-case and LIST patches in this cave. On the IIc, we found these earlier, in Saveload Cave.

The Double-Low-Res Spectacular. On the IIc, this area contains still more changes to the low-res graphics commands. To find out what these changes are all about, let's return to the Double-Low Forest and Storerecall Cave. A very interesting little routine lies at $F3CB on the IIc. It is called from several different places in the new code. This routine reads an undocumented softswitch at $C079, as well as softswitches that indicate whether the 80-column screen is being used.

A little experimentation with the unknown softswitch at $C079 shows that it indicates whether the IIc's double-resolution graphics have been turned on (high bit set, single-resolution; clear, double-resolution). According the IIc reference manual's Table 5-8, the softswitch for this is supposed to be at $C07F, not $C079; both addresses seem to work, however—*on the Apple IIc.* On IIes, both original and enhanced, these softswitches don't exist.

The routine we have under the microscope, the one at $F3CB, returns with the carry set if both double-resolution and an 80-column screen display are turned on; the carry is cleared otherwise. By looking at the routines that call this one, we can ascertain that they appear to allow the low-resolution graphics commands to work in an 80 by 48 mode if double-resolution graphics are turned on. *But there is no built-in command to invoke this new mode.*

If you have a IIc, try this series of commands:

```
PR#3                   (turn on the 80-column screen)
POKE 49246,0           (turn on double-resolution graphics)
GR                     (start-up low-res graphics)

COLOR=3                (set color to something other than black)
PLOT 50,10             (place a pixel at column 50, line 10)
X=SCRN(50,10)          (capture color at 50,10 in X)

HLIN 0,79 AT 20        (draw a line across the screen)
VLIN 0,39 AT 40        (draw a line down the screen)
```

Now how do you like that? *Here's a completely new graphics mode,* ***complete with Applesoft support,*** *and Apple's manuals don't breathe a word about it.*

Since it's probably been awhile since most of you worked with low-resolution graphics commands, here are some tips for experimenting with this new mode. First, **don't forget to set the color to something.** If you forget to use the COLOR command, your PLOTs and LINs will be invisible. You'll insist there isn't any such thing as *double-low-resolution graphics,* but there is. Second, **beware of putting an E at the end of the HLIN and VLIN commands.** If you do, Applesoft thinks the E is a variable and draws all lines beginning at whatever value E is (usually zero).

The primary advantage of double-low-resolution graphics, compared to the high-resolution graphics that Applesoft also supports, is that double-low supports sixteen complete colors rather than the six sort-of available in hi-res. Of course, double-*high* graphics can also do 16 colors (at 140 by 192 resolution), but Applesoft can't help you with that—it has no double-high-resolution commands.

A Double-Low Bug. The spectacular, hidden double-low-resolution graphics routines in the Apple IIc even have a spectacular, hidden bug. Since, in our graphic try-out earlier we said COLOR=3, and since we did a PLOT 50,10, the X=SCRN(50,10) command should have set X to 3. But now enter PRINT X. You'll find it equals 2. Try this:

```
10 *** SCRN Tester ***

100 GR
110 COLOR=5
120 HLIN 0,20 AT 10
130 FOR I=0 TO 20 : PRINT SCRN(I,10);" "; : NEXT
```

When you run this program, it gives you a series of fours and fives. It should give you a series of fives without fours. The SCRN command doesn't work right in double-low-resolution. The reason is pretty complicated. It begins with the fact that double-low-resolution color bytes that are stored in the auxiliary display page have to be rolled one bit to the right to compensate for some shortcomings in the Apple's color abilities. The new SCRN code has to take this into account when fetching values from the auxiliary page, and it does. Except that it *shifts* left when it should *roll* left (this is assembly language talk, for those of you who are puzzled by this paragraph). Consequently, SCRN always gets odd-numbered colors wrong when they are stored in the auxiliary display page.

Since we've gone this far, we might as well fix it, don't you think? Try this little program:

```
10 REM *** Applesoft Tune-up ***

100 TEXT : HOME : VTAB 12 : PRINT "Just a minute here..." : PRINT
110 X=PEEK(49281) : X=PEEK(49281) : REM read from $D000-$FFFF ROM, write to RAM

120 FOR P=208 TO 255
130 : PRINT "Working on page ";P
140 : FOR B=0 TO 255
150 : : I=B+(P*256)
160 : : POKE I,PEEK(I)
170 : NEXT
180 NEXT

190 X=PEEK(49283) : X=PEEK(49283) : REM read RAM (switches to RAM Applesoft)
200 POKE 63415,42 : REM change ASL to ROL
```

This program works by loading a copy of Applesoft into the "language card" area of memory. This area of RAM shares the address range normally used by Applesoft and the Monitor. The softswitches that control this memory allow reading and writing to be activated *independently.*

APPLESOFT TUNE-UP takes advantage of this in lines 120 to 180. At first these lines may make no sense at all. What they're doing is copying Applesoft and the Monitor from their normal home in ROM to the equivalent address range in RAM. Line 190 then turns on the new RAM version of Applesoft, and line 200 modifies it to fix the problem with the SCRN command.

Now try SCRN TESTER again.

Home again, home again, jiggidy jig. As we return from our expedition to the high reaches of Apple memory, consider the things inside Applesoft that Apple's wizards *didn't* change. While Applesoft is a pretty solid language, it does have some bugs in it. The surprise is that none of them, not even the easy ones, were fixed.

For example, there is a well-documented bug at $D970 that causes a RETURN WITHOUT GOSUB error when there isn't one (see **? Applesoft Error,** by Cornelis Bongers, in *Call -A.P.P.L.E. in Depth #1: All About Applesoft,* page 100). On the one hand, I can understand Apple's reluctance to make more changes inside Applesoft—compatibility problems could arise. On the other hand, however, the bug at $D970 could be fixed by changing the value of *one byte.* I don't think that would cause many compatibility problems. There are several other changes like this that could have been made.

But now that we have our own private version of Applesoft in RAM, what Apple did or didn't do doesn't make much difference. We can fix this bug and make any other changes we like as well. How about adding a DLGR (double-low graphics) command that does a POKE 49246,0 and a GR automatically? Add these lines to the end of the APPLESOFT TUNE-UP program:

```
210 C$="D970:86" : REM fix RETURN WITHOUT GOSUB problem
215 GOSUB 500

220 C$="D012:35 FB" : REM adr table position of TEXT, use Monitor to make space
225 GOSUB 500
230 C$="D06C:98 F3" : REM adr table position of LOAD, use for DLGR
235 GOSUB 500
240 C$="D1BC:44 4C 47 D2" : REM change 'LOAD' to 'DLGR' in cmd name table
245 GOSUB 500

250 C$="D90E:AD 53 C0 60" : REM GR and DLGR patches
255 GOSUB 500
260 C$="F390:AD 5F C0 AD 56 C0 4C EC D8 AD 5E C0 4C 93 F3"
265 GOSUB 500

280 PRINT CHR$(4);"BSAVE FPBASIC IIC.1, A$D000, L$3000"
290 END

500 C$=C$+" N D9C6G" : REM S.H. Lam routine
510 FOR I=1 TO LEN (C$) : POKE 511+I, ASC(MID$(C$,I,1))+128 : NEXT
520 POKE 72,0 : CALL -144 : RETURN
```

After running the complete APPLESOFT TUNE-UP, you'll be able to switch between double-low-resolution graphics and regular low-resolution with the commands DLGR and GR, *if* you have the 80-column screen active. (I tried to make DLGR turn on the 80-column screen, too, but DOS gets disconnected if you do it the easy way. To tell DOS what you're doing takes more space than is readily available—this is no doubt why the equivalent of DLGR wasn't built into IIc Applesoft.)

The program will also save an image of the modified Applesoft and the Monitor in a file called FPBASIC IIC.1 The purpose of this file is to speed up reloading this new version of Applesoft (APPLESOFT TUNE-UP itself takes over two-and-a-half minutes to execute). Here's how it works:

```
10 REM *** Load My Basic ***
100 TEXT : HOME : VTAB 12 : PRINT "Just a minute here..." : PRINT

110 X=PEEK(49281) : X=PEEK(49281) : REM read ROM, write RAM
120 PRINT CHR$(4);"BLOAD FPBASIC IIC.1"
190 X=PEEK(49280) : REM read RAM
```

Now suppose you ran LOAD MY BASIC on a IIe or II-Plus. Could you use the IIc version of Applesoft to add double-low-res Basic commands to an older machine? Since the II-Plus doesn't support double-resolution, it's out immediately. A IIe would need a 65C02 microprocessor to execute IIc Applesoft, so an enhanced IIe is a possibility. But it doesn't work. The problem is our old friend $C079. This is the byte that indicates whether double-resolution graphics are active on a IIc. On a IIe it flips back and forth pretty much at random.

Nonetheless, the *techniques* we've looked at here can be used on a IIe or a 64K II-Plus to make special versions of Applesoft. Just remember you have to use an unrelocated version of DOS 3.3 to do these tricks. ProDOS uses the language card for its own purposes, so with it there's no room for your personal version of Applesoft.

If this sounds interesting to you, I suggest you contract S-C Software (P.O. Box 280300, Dallas, Texas 75228; 214-324-2050) and get their program called *S-C Documentor* ($50). Using the Applesoft ROMS in your computer, this program creates a complete, well-commented, source code listing of Applesoft. The code is put into files formatted for the S-C Macro Assembler. Modifications are much easier when you don't have to disassemble every line of code yourself. This article, for example, would have been unimaginable without the *Documentor's* help.

A CP/M Primer

by Jim Whitehead

Editor's note: I would like to write everything in ***Open-Apple*** *myself, but there are some things I just don't have time to investigate. When I find something that's really good in one of these areas, I'll try to publish it for you. CP/M is an example of something I don't know much about, but lots of Apple users are interested in it. The following is the first article I've ever seen on CP/M that doesn't use the word "BIOS." It is reprinted, with permission, from the March 1985 issue of* ***NEAT Notes****, the newsletter of New England Apple Tree (P.O. Box 2519, Woburn, Mass. 01888).*

At last month's NEAT meeting, there was an emphasis on CP/M. However, since CP/M is by no means easy to learn, there may be many club members who are totally in the dark about what went on, or who don't know how to use the stuff they picked up at the meeting. Therefore, here is a CP/M tutorial.

CP/M is an operating system, such as DOS 3.3 or ProDOS. However, CP/M is different. DOS 3.3 and ProDOS were primarily designed to take advantage of Applesoft, which is built into the Apple. CP/M, on the other hand, was designed to be independent of any particular language. Thus, while DOS 3.3 and ProDOS contain special features for the sake of Applesoft, CP/M is only concerned with operating the hardware of your system—the disk drives, printer, modem, and other peripherals. The full name for CP/M indicates the limits of its functions—Control Program for Microprocessors.

CP/M is software—a program—just as DOS 3.3 is software. This often confuses Apple owners because they have to buy a piece of hardware—a "CP/M board"—to use it. The reason for this is that CP/M is written in 8080 machine language, not the 6502 machine language native to the Apple II. To run 8080 programs, you need an 8080 (or a more advanced Z-80) microprocessor. Most CP/M boards give your Apple a Z-80 microprocessor on a plug-in card. What the card does is allow its onboard Z-80 microprocessor to converse with the Apple's I/O slots and memory while executing CP/M. Some boards contain their own memory in addition to other built-in functions, but their purpose is basically the same—to allow CP/M programs to run on an Apple II computer.

Why should you want to run CP/M applications? Because much of the best software available for microcomputers, particularly for business applications, executes only on CP/M machines. *WordStar* cannot be run on a 6502, nor is *dBase II* available for un-CP/M'ed Apples. Microsoft's *Multiplan* even works correctly with your Videx 80-column card when running under CP/M. (Under DOS 3.3, the screen is shifted to the left about 1/2 inch, and is very annoying to work with.) This brings up another highlight of CP/M. If you have an 80-column card, you seldom need to worry whether it will work with CP/M software. In fact, unlike DOS 3.3, under CP/M it's the 40-column users who are discriminated against—a satisfying feeling for those of us who have plunked down good money for an 80-column card just to be left stranded by most programs. For these and other good reasons, CP/M is well worth purchasing.

However, like most computer products that actually do the neat things they're supposed to do, CP/M can't be fully utilized without plowing through long manuals full of computerese and double-talk. This tutorial will attempt to convey basic mental attitudes needed for using CP/M. If you start thinking the CP/M way, learning new programs and utilities becomes a snap. However, CP/M does things in a manner that differs greatly from DOS 3.3, so you'll need to throw out all of your preconceptions of how an operating system should be.

The first radical change between DOS 3.3 and CP/M is the way disk drives are named. Say goodbye to drive 1 and drive 2, all drives are letters now. Old drive 1 becomes drive A: (the colon is very important) and drive 2 becomes drive B:, etc. Plus, since there are no slots in CP/M, a third disk drive in slot 5 (a DOS 3.3 Slot 5, Drive 1) would be called drive C:.

The next difference between DOS 3.3 and CP/M is the hardest one to take. CP/M limits filename lengths to only eight characters. This often necessitates the creation of mnemonic file names, such as PADDPROG instead of PADDLE PROGRAM. This also brings up one major problem with CP/M—after a month or so, who can remember what the mnemonics RTATR4 or LIPSTR stand for?

CP/M does have one other part to its filenames, however, called the extension. This is a three letter suffix tacked onto the main filename with the glue of a dot (.). File extensions are supposed to give some idea of what is in the file. For example, text files from a word processor should have an extension of .TXT, standing for text. Basic files automatically have a .BAS added to them, and other applications usually have their own specific extensions. The filename extension is the only means within CP/M of differentiating between separate file types.

A full filename in CP/M has all of the elements just discussed, exactly in the following order:

```
{drive}{filename}.{extension}
```

In CP/M, the drive being referred to is placed before the filename, instead of after it. There is no ",A:" syntax in CP/M like the ",D1" syntax in DOS 3.3. On occasion, the beginning drive name can be omitted, thus automatically referring to the default drive. Which one is the default drive? If you haven't figured it out by now, the CP/M prompt is significant. Upon booting up with CP/M, the prompt is a "A>". The "A" indicates the default drive, and the ">" is for decoration. To change the default drive, type a "B:" (or any other valid drive) at the CP/M prompt. (Make sure a disk is in the drive!) The default drive is assumed every time you omit a drive specification in a filename. Therefore, if CP/M mysteriously can't find a file you know is there, check the disk drive specified and make sure it wasn't omitted.

Once the format for filenames is understood, the next most important difference between DOS 3.3 and CP/M is the "=". In DOS 3.3, the "=" is often used as a wildcard character, allowing any text to be automatically substituted in its place. In CP/M, the "*" is the wildcard character and "=" is used in another way. It designates what file is to be acted upon and what file is to be used in an action. As in the Basic statement *A=A+1*, in CP/M the "=" does not indicate an equality, but tells the computer to "make these things equal." Following this definition, the "=" is interpreted as: make the first thing equal to the second thing. The "things" are usually filenames.

In PIP, the FID of CP/M, the equality is straight. The second "thing", a file on a disk, is copied into the first "thing", thus making them "equal". In APDOS, the DOS 3.3 to CP/M conversion program that comes with the Microsoft Softcard, a straight equality is again made. The exception is that the first "thing" is a CP/M file and it is made "equal" to the second "thing" by means of a file conversion. Thus, while the "=" is used in both PIP and APDOS to make an equality between files, what the programs are actually doing varies greatly.

Now let's look at some of the other frequently used CP/M commands. Instead of CATALOG, CP/M has a DIR(ectory) command. This command is entered in the form:

```
DIR {drive}{filename}
```

Typing DIR will catalog the default drive, typing DIR B: will catalog drive B:, etc. Typing DIR followed by a filename acts like the verify command under ProDOS, simply demonstrating that a file exists.

Other useful commands are called using the format:

```
{command} {filename to act on}
```

These commands are:

TYPE—displays the contents of a .TXT or .ASM file. (Or a file with any extension, if it contains text.)

STAT—displays how much space a file uses on a disk. When called like DIR, (i.e. STAT B:) it will display only how much space is left on the designated disk. This command also has many other functions, consult a manual for more details.

ERA—Erases the filename specified. Type ERA *.* to completely erase a disk of all files. In this example, the asterisks are wildcards and the default drive is scrubbed.

As mentioned earlier, one program that is essential to using CPM is called PIP (Peripheral Interchange Program). This program copies files from one disk to another. It also has the ability to transfer files to the various devices connected to your computer. Using PIP, a .TXT file can be transferred to a printer or to a modem. For bare copying functions, use this program with the like this:

```
PIP {newfile}={oldfile}
```

This command gets PIP to copy oldfile onto newfile, just as our knowledge of the "=" would indicate.

Another command that is used with the "=" is REN. REN is equivalent to DOS 3.3's RENAME command, but it is called in a different fashion. With DOS 3.3, to change the name of a file you enter RENAME {oldname},{newname}, and the file is renamed. To do the same thing under CP/M, you must enter REN {newname}={oldname}.

There are many other commands available to CP/M. CP/M is limited only by the machine it is running on. All CP/M files ending in ".COM" are commands. Hence, though syntactically PIP and REN are summoned the same way, they are different because PIP is a ".COM" file while REN is an intrinsic command of CP/M. This is a powerful idea, for it allows a multitude of commands to be added to and deleted from the CP/M vocabulary at will, with only those commands that are necessary taking up precious memory space.

The full power of this "nothing but the bare necessities" idea is evident in Microsoft's MBASIC. This version of Basic is much more complex than Applesoft and is a real joy to program in. Yet all of this extra power is accomplished in less space than what Applesoft and DOS 3.3 take up. Disk commands are a part of MBASIC, allowing easy programming with no control-D's and no machine language overhead devoted to interpreting control-D's.

For those of you who are unfamiliar with MBASIC, yet who want to run MBASIC programs, here is a short explanation of how to get these programs operating.

It is somewhat more difficult to start a Basic program under CP/M than under DOS 3.3 and ProDOS. This is because you have to start up MBASIC by hand, where Applesoft starts up automatically. To get an MBASIC program running, you first have to boot CP/M. Then startup MBASIC. It's in a .COM file, so just type "MBASIC" at the CP/M prompt. After loading, MASIC will give a welcome message and then print OK. Once you see this OK, you may place a disk with programs on it in the other drive, drive B:. Then type:

```
LOAD "B:{program name}"
```

There is no need to add the .BAS extension, as it will be done automatically, but the quotation marks are essential. After the program is loaded, type "RUN" to start it. Use Control-C to exit a program at any time. Type "SYSTEM" to get back to the CP/M prompt.

If you don't have two disk drives, the procedure is more complicated. First you have to create a blank system disk. Then use PIP to copy both MBASIC-.COM and the program you want to execute onto this disk.

Another thing that often confuses beginners is that while DIR is the command for cataloging a disk from the CP/M prompt, inside MBASIC you have to use another command. It goes like this:

```
FILES "B:"
```

Once again, the quotation marks are essential.

That wraps up this CP/M tutorial. Be aware that we have only scratched the surface of CP/M here. Use this as a beginning. Consult at least two CP/M books on points you are unfamiliar with. Two viewpoints often fill in what each other missed. Good luck, and happy CP/Ming!

Miscellanea

The current version of *AppleWorks* is 1.2. If you have an older version, backup your disks and take the originals to your dealer and ask to have them updated. Version 1.2 includes a way to modify printer interface card setup strings and supports Apple's Scribe printer. Other changes are minor. Here are some setup strings to try for various interface cards:

```
Printer interface card setup strings for AppleWorks

Tymac/Microtek         control-I 99 N
Apple's own cards      control-I 80 N
Grappler/Pkaso         control-I 0 N
Practical Peripherals  control-I N
```

Apple surveyed software developers last August and the results, though a little old already, are interesting. Apple sent its survey to 3,200 certified Apple II developers and 970—30 per cent—responded. At the time, more respondents said they were developing software for the Apple II-Plus than for the Apple IIc. Over half the respondents said they were still developing II-Plus software and more than a third said they planned to continue doing so.

Three times as many developers were using DOS 3.3 as either Pascal or ProDOS. Basic was the development language used by most people, though with first and second choices added together, assembly language nudged into first place.

The most desired new features for the Apple II were larger floppies (29 per cent), a faster microprocessor (19 per cent), and more RAM (10 per cent). "Please indicate your feelings concerning 3-1/2 inch disk technology. Issues: more capacity, no DOS 3.3 support, new disk media, compatibility:" strongly in favor, 48 per cent; neutral, 32 per cent; strongly opposed, 5 per cent.

Software Is a Cheap Business To Get Into—but Many Fail was the title of a recent *Wall Street Journal* article (April 29, page 21). The article gave the results of yet another software developer survey—this one done by professors at the Georgia Institute of Technology. People who start software companies are likely to be male, well educated, and optimistic, the survey found. Most companies are started with a modest capital investment—more than half the 193 respondents to this survey started with less than $10,000. The researchers conclude that developing software for microcomputers is a "classic cottage industry."

The article also quotes Bernard Goldstein, a partner in a firm that specializes in software company mergers. Creating a new software product can't always be accomplished with lots of money and people, Goldstein says. The ability to create new software is often lost at established firms, he contends, because the talent that can do that has left or is occupied maintaining and improving existing software. Often new products are developed by people working in their spare time or in obscure businesses. Creating a software product is like writing a book, Goldstein suggests. "If you consider what the author of a book receives if he is successful and what the successful software author receives, I think the odds are better for the software author."

I agree with Goldstein. I would add that although stories abound of 13-year olds who make a $100,000 a year selling software, precious few adult Apple II software developers do anywhere near that well. However, you *can* make a comfortable living at software development *if you understand the importance and cost of marketing* and you don't get greedy. Writing books, as Goldstein suggests, is a pursuit best left to the independently wealthy.

Nibble magazine, long a profligate user of unpunctilious punctuation, is primarily responsible for the shortage of exclamation points that is plaguing the publishing industry. *Nibble* set a new record for computer magazines in May when it used four (count'em—four!) exclamation points on its cover. Many more were used inside. Publishing industry experts say it may be months before exclamation point manufacturers are able to rebuild supplies.

In Memoriam: *Access: Apple,* Time-Life's newsletter for business users of Apple computers, died with the May issue. Let us also momentarily bow our heads for IBM's dreaded Apple II-killer, the PC jr. It really was a peanut after all.

Ask (or tell) Uncle DOS

The customer is always right

Did you notice? Yes, the type **is** *bigger in the front section of this newsletter. That's because lots of paying customers said they couldn't read our old type and because some wag reviewing* ***Open-Apple*** *said the type was smaller than Epson's ultra-condensed. I couldn't take it anymore.*

Really quite expensive

Potential subscribers to ***Open-Apple***, BEWARE! My subscription has cost me $1,247.00. Because of articles in this crazy newsletter I have purchased a Sider hard drive and Applied Engineering's RAMWORKS with RGB.

I have a IIe and a II-Plus. I took the II-Plus to work to prevent computer withdrawal. Is there any way to trick the II-Plus into running *AppleWorks*? If so, then how about getting it to take advantage of a 128K Saturn card?

What do you suggest is the best way of backing up the Sider in a fairly speedy and inexpensive way?

Gary Ward
Honolulu, Hawaii

If you think ***Open-Apple*** *is expensive now, wait until I review Apple's $7,000 LaserWriter printer. Don Lancaster says it works better on a II running Apple Writer than on the Macintosh.*

I'm sorry about the dent in your wallet, but I think you've made good choices. The ProDOS/Sider/AppleWorks/RAMWORKS combination is the most significant thing to happen in the Apple II world since the IIe was introduced. Apple itself doesn't push this configuration because parts of it compete with Apple's own present and future products. Few people relying on the advice of dealers or the general press will hear about it, either. Nonetheless, it appears that thousands of experienced Apple users are putting together systems like this and finding themselves with more computer power than they ever dreamed of.

Checkmate Technology (509 South Rockford Dr, Tempe, AZ 85281, 800-325-7347) makes a $350 80/160-column card that allows you to run AppleWorks on a II-Plus. I have not tried the card, but I am inclined to think that if your main goal is to run AppleWorks, your $350 would be better spent making a down-payment on another IIe or a IIc. (You're right, this **is** *an expensive newsletter.) II-Pluses are best used in non-keyboard/screen intensive tasks—making copies of disks, running printers, and so on. I think the future of the II-Plus is as a network controller, but the required hardware and software hasn't been developed yet that I know of.*

The Sider comes with a program for backing up the DOS 3.3, CP/M, and Pascal areas of the disk. Apple has a ProDOS program called ***Backup II***, *which I got a review copy of sometime ago. It seems to work pretty well, however, it is currently only available by buying Apple's Profile hard disk. If Apple is smart they'll soon offer it separately, since few people are going to pay $1,300 extra for a Profile just to get it. It would make a great $40 to $50 stand-alone package, however.*

Breaking the AppleWorks barrier

I have been using the data-base module of AppleWorks for the past month and agree with your recent comments regarding its usefulness. Extra memory on a RAM card increases the size of the desktop, but is there any way to increase the 1,350 limit on the number of records per data base file that AppleWorks will accept?

Martin Kalman
Friday Harbor, Wash

I touched on the solution in last month's issue, but it's worth repeating—the current version of the software that comes with Applied Engineering's RAMWORKS allows 4,283 records per file.

Interrupts and the sun

I'm working on a Basic program that uses a Thunderclock and an analog-to-digital conversion board to periodically measure temperatures and pump status in an open-loop solar collector system. Eventually I'd like to get it into machine language so that I can use interrupts to achieve foreground/background operation but that's a ways down the road. I'd like the display to be in 80 columns but seem to get in trouble whenever I read the clock or the a/d board. Is there a way to read another slot and keep 80 columns without refreshing the screen every time? I've been interested in Apple's IIe enhancement because I've read they have finally fixed the byte $45 interrupt problem. This afternoon I read that ProDOS supports interrupts on the regular IIe. What gives? Do I need the enhancement or not?

Bill Stiles
Charleston, S.C.

To return to the 80-column card without clearing the screen do this:

```
DOS 3.3
   POKE 54,7 : POKE 55, 195 : CALL 1002

ProDOS
   PRINT CHR$(4);"PR#A$C307"
```

See the January issue (page 6) if you want to know how these tricks work.

Let's go through interrupts from the top. An ***interrupt*** *is an electronic signal from a peripheral that tells the 6502 microprocessor to immediately stop whatever it's doing and come to the aid of the peripheral.*

A typical use for interrupts would be with a high-speed modem. As each character is received, it is necessary for the computer to get it and store it someplace so that it isn't overwritten by the next character coming in. If interrupts aren't used, the modem software must continually **poll** *the modem to find out if a character has been received yet. With interrupts active, on the other hand, the modem stops the software and yells "come and get it" whenever a character is ready.*

Most clock cards can also be set up to send an interrupt every so often. Clock interrupts provide a way for your computer to check on a security system or a solar collector every once in awhile (in the background), while still being primarily used for something else (in the foreground). I too have a Thunderclock and although it has proven very reliable (and expensive), its slowest interrupt rate is 64 times a second—somewhat more often than I've ever had any use for.

Many computers have an interrupt-driven keyboard. This is a good use for interrupts, although the Apple keyboard doesn't normally support this. Products such as Apple Writer must take a peek at the keyboard about a hundred times a second, even while doing such things as scrolling the screen, to keep a fast typist's keystrokes from being lost. However, the Apple Mouse card (built into the Apple IIc) provides keyboard interrupts to the Apple II family if you really need them.

The main idea of interrupts is that after the interrupter's needs have been taken care of, the 6502 can go back to whatever it was doing before. In order to do this successfully, however, the status of the microprocessor and of the computer's memory configuration at the time the interrupt occurred has to be saved somewhere. The 6502 automatically saves its status register and program counter on the stack when an interrupt occurs, but it's up to a machine's software to take care of everything else. After the 6502 saves the status and program counter, it jumps to whatever address is stored at memory location $FFFE.

Interrupt support is built into all Apple IIs. In the original II, II-Plus, and IIe, the address stored at $FFFE points to a routine in the Monitor that saves the contents of the 6502's A register at memory location $45 before passing control to the programmer's interrupt handler. This would be great ***if no other software ever used $45 for anything.*** *Software that does use $45, however, fails with interrupts, because every time an interrupt occurs the contents of byte $45 are destroyed.*

The Apple II has a reputation of not working with interrupts because ***DOS 3.3 saves stuff in $45.*** *You can work around this bug by disabling interrupts before giving a DOS command and enabling them again afterwards (this requires assembly language tricks. By the way, both DOS 3.3 and ProDOS disable interrupts while actually reading from or writing to a disk. This is because the subroutines that actually deal with the magnetic patterns on your disks are extremely time-sensitive and don't work if interrupted. Consequently, interrupts are disabled and can remain that way for relatively long periods of time during disk access—several seconds if an error occurs that causes the disk to recalibrate itself.)*

On the enhanced IIe and IIc, Apple solved the interrupt problem by saving the A register on the 6502's stack rather than at byte $45. Thus, you can successfully interrupt DOS 3.3-based programs on one of these machines. The enhanced IIe and IIc also automatically save the memory configuration at the time of the interrupt, and put it back again when the interrupt handler is finished.

ProDOS never uses $45. Consequently, it supports interrupts on any kind of Apple. It also automatically saves the registers, bytes $FA through $FF, and 16 bytes of the stack. The interrupt handler can freely use these resources. Neither the interrupt handler nor any ProDOS program, however, should ever use $45 or $7F8 (ProDOS uses $7F8 to store the number of the slot that was active when the interrupt occurred.)

Dump 40

I am looking for a method to dump just the 40-column screen to the printer. I tried modifying your 80-column dump (February, page 15) by poking only the odd columns, but it did nasty things to my Applesoft program.

Richard Hanley
Milford, Conn.

*You can turn February's **DUMP 80** into **DUMP 40** by simply deleting lines 330, 340, and 350. In 40-column mode, it isn't necessary to poke anything to read the screen memory correctly. (If you've updated your **DUMP 80** with the modifications I gave in March (page 22) to make it run faster, you'll also have to remove the **L$+** from line 360.)*

Learning assembly language

I'm interested in learning assembly language. I am at a beginners level programming in Basic. Could you recommend any sources?

John J. Gwynn
Roslindale, Mass.

*I think the best book around for learning assembly language on the Apple is Roger Wagner's **Assembly Lines**. Since this book was published by Softalk Books, many people are under the impression it is no longer available, and it wasn't there for awhile. However, Mark Pelczarski of Penguin Software recently managed to wrestle down the rights to Softalk's books and is now distributing them.*

They're still called Softalk Books and they can be ordered from Penguin Software, P.O. Box 311, Geneva, IL 60134; orders 800-323-0884, information 312-232-1984. Apple-related books currently available and prices (add $2 for shipping) are:

```
Applesoft Isn't Hard     $14.95
  by Doug Carlston

Assembly Lines           $14.95
  by Roger Wagner

Graphically Speaking     $14.95
  by Mark Pelczarski

Macintosh Complete!      $14.95
  by Doug Clapp
```

Apple IIc hardware fix

Early Apple IIcs (serial numbers below D51000) have a hardware bug that can prevent them from working with 1200 baud modems and other high-speed peripherals. If you have a peripheral that won't work with your IIc, Apple dealers are supposed to replace the motherboard in your IIc without charge.

Jim Luther
Kansas City, Mo.

Super hi-res graphics

Do you know of any way of increasing the Apple II's resolution past 560 x 192? There was mention of a card in an old *Softalk* letter that had resolution capabilities of 640 x 768. It was made by Demco Electronics and called the *Graphics Tool Kit*. Does this still exist? Where are these people? How come other companies don't make "add-ons" to increase resolution.

I love my Apple and wouldn't trade it for all the IBMs in the world, but when it comes to information about graphics there is a curious lack of information. For example, no computer store in my area has *any* graphics tablet for me to try. They all say there is no demand for them so they can't afford to stock them. Not only that, 50 per cent of the stores claim Apple doesn't make a graphic tablet anymore; what gives?

Clyde Godfrey
Ann Arbor, Mich.

Demco Electronics (10516 Grevillea Ave, Inglewood, Calif. 90304, 213-677-0801) sells its Graphics Tool Kit for $595. It includes eight disk sides of software and sample data and a peripheral card that works in slots 1 through 7. It provides 640 x 384 screen resolution (the Macintosh screen is 560 x 354) and can print at 640 x 768.

*While finding out about these higher-resolution modes can be hard, there **is** something there to find out about. I recently got a letter from a company called MicroGraphic Images (21040 Victory Blvd, Suite 210, Woodland Hills, CA 91367 818-368-3482), which is justifiably proud of itself for having created a professional-quality high-resolution graphics workstation based on the Apple IIe. Their system has 512 x 512 screen resolution with 16 colors and simultaneously uses the Apple's 6052 and a 6 MHz Z-80 card to increase the system's speed. The whole thing, including a IIe with drives, Videx Ultraterm and monochrome monitor for the "command" screen, graphics board and RGB monitor for the "design" screen, Z-80 board, graphics tablet, software, detailed manual, and heavy-duty power supply and cooling fan costs $8,795, which is about $20,000 less than the systems it competes with. If you already have the hardware, the guts of the system (software mostly) are available separately at a lower price.*

I called Diane Blake, Vice President of Creative Development at MicroGraphic Images, and asked her your question about obtaining information on super high-resolution graphics on the Apple II. She agrees with you that it's hard to obtain. She said it wouldn't actually be necessary to see a card before buying—what you need to ask about is the resolution, number of colors, and kinds of graphic primitives built into the hardware. MicroGraphic Images uses the Number Nine Graphics Card, from a company called Number Nine (stupid me, I forgot to ask for their address; MicroGraphic Images will probably sell you one if you're interested, though).

One thing to consider is that when you deal with super high-resolution, you have to move a lot more bytes around to fill a picture. Thus, super high-resolution systems on the Apple II may be slow unless you also add a faster 6502 or some other chip, as MicroGraphic Images does. And unless you are an assembly language programmer willing to devote all your time to graphics, the software that comes with any card you buy will be very important to your ultimate satisfaction.

Regarding Apple's Graphic Tablet, Blake agrees with you that it is hard to obtain but she says it is still being made. (A call to Apple confirmed this. The suggested retail price is $795.) Dealers are mixed up because Apple pulled it off the market for about a year sometime ago because of radio-interference problems that have since been solved. Blake said MicroGraphic Images became an Apple VAR (Value-Added Reseller; this means they buy direct from Apple rather than through dealers) primarily to have better access to the tablets. Blake also mentioned that there are RS-232 graphics tablets available from other manufacturers; these would work with an Apple II if you had a serial card.

Blake says most of MicroGraphic Images' potential customers are surprised to find such a high-powered system running on an Apple II. Most of their competitors use IBM-PCs. "The quality, reliability, and open architecture of the Apple II is the tie that binds this system together. On any other machine it would be slower, more expensive, and less reliable."

California-area subscribers interested in just how far the little Apple II can be pushed, as far as graphics are concerned anyhow, can see MicroGraphic Images' stuff at the West Coast National Computer Graphics Association conference in Los Angeles, June 25-27.

HCOLOR me invisible

I recently encountered a hi-res graphics program that worked well on an Apple II-Plus but didn't work at all on a IIe. After some fiddling around, I discovered the problem had to do with the default HCOLOR on power up. When the II-Plus powers up, HCOLOR defaults to 7 (white); when the IIe powers up, HCOLOR defaults to 0 (black). This can be seen by examining byte 228 ($E4).

Thus programs that fail to specify an HCOLOR will fail to work on the IIe, even though they (accidentally) work just fine on the II-Plus.

Now a question. Normally an Apple peripheral card can be identified by peeking its slot location. For example, to verify that a disk controller card is in slot 6, all one needs to do is PEEK the locations starting at 50688 ($C600). For the Apple Disk II interface, bytes 50688 through 90 always contain 162, 32, and 160.

However, this trick doesn't work for an Integer Basic ROM card or a MicroSoft Softcard. The values peeked don't remain constant over time. The same thing happens with empty slots. How come?

Michael Ching
Honolulu, Hawaii

Thanks for your HCOLOR tip. For the record, the meaning of the values you'll find by peeking at byte 228 ($E4) are: 0=black.1; 42=green; 85=violet; 127=white.1; 128=black.2; 170=orange; 213=blue; 255=white.2.

Slots 1 through 7 in all Apple IIs each have an exclusive 256-byte address range for machine language programs. A program that appears in this range is really on a ROM chip that's on a peripheral card. The program will appear within its slot's assigned address range when the card is plugged into the slot.

The address range for each slot is $Cs00 to $CsFF, where "s" is the slot number. The address range from $C800 through $CFFF can also be used by machine language programs on peripheral cards, however, only one card at a time can use this space. Whenever a reference is made to byte $CFFF, all peripheral cards are supposed to turn their $C800-$CFFF memory off. Whenever a reference is made inside a card's 256-byte program area, that card is supposed to turn its $C800-$CFFF memory on. This feature has been built into the Apple II since the beginning, and was the first use of "bank switching" on the Apple.

The system is ingenious. Most computers have to be "reconfigured" whenever a new peripheral is added or removed. This means the operating system must be modified by adding or removing machine language "drivers" that interact with the peripheral. But with the Apple II, this reconfiguration takes place automatically when a card is inserted or removed, since the required machine language programs are built into the peripheral cards. Steve Wozniak credits his friend Alan Baum with coming up with this scheme (Byte, December 1984, page A71).

However, some cards don't take advantage of it. They have no machine language drivers built into the 256-byte program space. Since there's nothing there, you'll get a different number everytime you PEEK at those addresses, just as you do if the slot is empty.

Printers to flash

I have developed a revolutionary new product that your readers might be interested in—a printer that will print flashing characters. I've been working day and night (yesterday and maybe tonight) on this project. First I tried octopus ink, but ASCII printed as octal. I diluted it 4:1 and got binary.

Finally I got a bright idea and tried ink made from the juice of Canadian fireflies.

Murray MacKenzie
Scarborough, Ontario

Readers may address inquiries to MacKenzie in care of the Northern Lights.

The wonderful world of windows

Type in and run the following Applesoft program:

```
10 TEXT : HOME : FOR I=1 TO 23 : PRINT I : NEXT
20 POKE 32,3 : HOME : END
```

Why does HOME clear the lower 16 lines of the area outside (to the left of) the new text window and not the upper eight?

Dennis Doms
Kansas City, Mo.

Let's begin by explaining a little bit about ***windows*** *on the Apple II to everyone. The Apple II operating system has always provided programmers the ability to specify a "text window" for all further PRINTing to appear in. Usually this window is set to full-screen—24 lines high by either 40 or 80 columns wide. But a few simple POKEs can make the window smaller and put it anywhere on the screen. The following table shows which bytes to POKE:*

byte	use	allowable range 40-col	80-col
32 ($20)	left edge	0-39	0-79
33 ($21)	width	1-40	1-80
34 ($22)	top	0-23	0-23
35 ($23)	bottom	1-24	1-24

The following program demonstrates a way to set up windows using an integer array (variable name ends with a "%") to hold the dimensions of several different windows. It will show you some of the possibilities windows provide:

```
10 REM *** WINDOW DEMO ***

100 NORMAL : TEXT : HOME
110 SW=40 : REM determine screen width (40 or 80?)
120 IF PEEK(64435)=6 AND PEEK(49183)>127 THEN SW=80
130 NW=9 : REM number of windows used
140 DIM W%(NW,4) : REM integer array for windows

150 FOR W=1 TO NW : REM get and check dimensions
160 : READ W%(W,1) : REM  left edge
170 : READ W%(W,2) : REM  width
180 : READ W%(W,3) : REM  top
190 : READ W%(W,4) : REM  bottom

210 : IF W%(W,1)<1 OR W%(W,1)>SW THEN
        PRINT "LEFT EDGE OFF SCREEN IN WINDOW ";W
220 : IF W%(W,1)+W%(W,2) > SW+1 THEN
        PRINT "RIGHT EDGE OFF SCREEN IN WINDOW ";W
230 : IF W%(W,3)<1 OR W%(W,3)>24 THEN
        PRINT "TOP EDGE OFF SCREEN IN WINDOW ";W
240 : IF W%(W,4)<1 OR W%(W,4)>24 THEN
        PRINT "BOTTOM EDGE OFF SCREEN IN WINDOW ";W
250 : IF W%(W,2)<1 THEN
        PRINT "WIDTH NEGATIVE IN WINDOW "; W
260 : IF W%(W,3) > W%(W,4) THEN
        PRINT "TOP BELOW BOTTOM IN WINDOW ";W

270 NEXT
280 PRINT : PRINT
290 INPUT "PRESS <RETURN> TO BEGIN";W$

300 REM  LEFT  WDTH  TOP BTM
310 DATA   1,   20,   1,  12
320 DATA  20,   20,   1,  12
330 DATA   1,   20,  13,  24
340 DATA  20,   20,  13,  24
350 DATA   1,   40,  18,  24
360 DATA   1,   40,  18,  18
370 DATA  20,    1,   1,  24
380 DATA  20,    5,  12,  12
390 DATA   0,   82,  25,   0

500 INVERSE : HOME
510 INPUT "WINDOW #?";W$
520 NORMAL : TEXT : HOME
530 W=VAL(W$) : IF W<1 OR W>NW THEN END
540 POKE 32,W%(W,1)-1 : POKE 33,W%(W,2) :
        POKE 34,W%(W,3)-1 : POKE 35,W%(W,4)
550 VTAB W%(W,3) : HTAB W%(W,1) :
        REM put cursor inside new window
560 GOTO 500
```

Lines 100 to 390 initialize the program. For things to work right we need to know whether a 40-column or an 80-column screen is active. Line 110 assumes 40-columns; line 120 changes this to 80 if byte 64435 indicates the program is running on a IIe or IIc AND if byte 49183 indicates 80-column mode is active.

Line 150 sets up a loop for reading the window dimensions from the DATA statements in lines 300 to 390. Lines 210 to 260 check these dimensions to make sure they are allowable and sends messages to the screen when mistakes are found. Program initialization ends in line 290.

The main body of the program is at lines 500 to 560. Line 510 asks which window you want to activate. When you run the program, enter a number between 1 and 8 in response to this question. Entering 9 will crash the program because the dimensions for window 9 are bad. Anything else ENDs the program.

Line 540 pokes bytes 32-35 with the dimensions for the chosen window. It also makes some adjustments to the dimensions (by means of the -1 in POKE 32 and POKE 34) so they're easier for us humans. These adjustments allow you to specify window dimensions using the same screen coordinates as VTAB and HTAB. In other words, the adjustments allow you consider the top line and the left-most column as number 1, rather than 0.

Line 550 puts the cursor inside the specified window. Line 560 loops back to line 500, where the program continues. If you are using a IIe or IIc with the 80-column firmware active, line 500 will cause the window to appear as an inverse rectangle on your screen. On a II-Plus, or with the 80-column firmware off, you will have to search for the edges of the window by pressing Escape and moving the cursor around with the I-J-K-M keys. Alternatively, you could replace line 500 with this:

```
500 INVERSE : FOR I=1 TO 24 : PRINT SPC(SW) : NEXT
```

It's not elegant, but it will make your window clearly visible.

If the WINDOW DEMO gets your interest up, try GIRAFFE. However, I warn you that it has no redeeming social value whatsoever:

```
10 REM *** GIRAFFE ***

100 NORMAL : TEXT : HOME
110 SW=40 : REM determine screen width (40 or 80?)
120 IF PEEK(64435)=6 AND PEEK(49183)>127 THEN SW=80
130 POKE 33,1 : REM make window 1 column wide

200 INPUT "]";X$
210 HOME : PRINT CHR$(7);"]I FEEL ILL."
220 C=PEEK(32)
230 C=C+2
240 IF C=>SW THEN TEXT : HOME : POKE 33,1 : C=0
250 POKE 32,C
260 VTAB 1
270 GOTO 200
```

This program uses the TEXT command a couple of times. TEXT always opens the window to the full screen size (as well as turning off graphics, which is meaningless here). Pressing the Reset key also opens the screen window to full size. If you use windows in your programs, remember that your reset trap, if you have one (see the February issue, page 16), will have to fix things up.

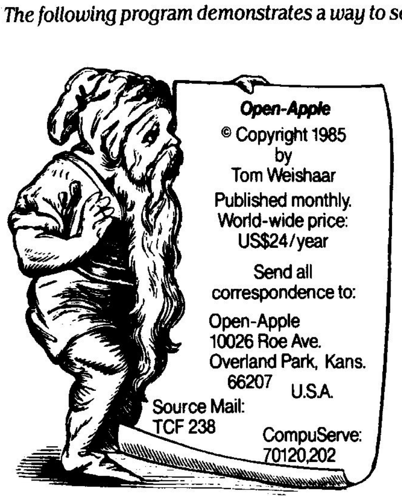
Window bugs result from a number of mistakes, but the two most common are making a window wider than the screen itself and forgetting to put the cursor inside the window. The Apple's screen display routines do not check the contents of bytes 32-35 to see if they make sense. If you put in nonsense you don't get an error message—you get erratic operation. ***This includes the possibility of destroying your program.***

The 80-column firmware in the original IIe has some of its own window bugs. To avoid them, put the left edge of all windows in an odd column (assuming you call the first column 1, not 0).

Now that the preliminaries are out of the way, let's answer the original question. POKE 32,3 moves the left edge of the "text window" to the fourth column on the screen. The problem here is that byte 33, the screen width, still holds 40 (or 80), thus right edge of the window is in column 43 (83). This is off the screen, but the screen display routines don't check for that. Thus, the HOME command starts messing with bytes it's not supposed to touch.

This time the bug is in the program, not in the Apple. To get the program to work correctly, change line 20 to:

```
20 POKE 32,3 : POKE 33,40-3 : HOME : END
```


Open-Apple
© Copyright 1985
by
Tom Weishaar
Published monthly.
World-wide price:
US$24/year
Send all
correspondence to:
Open-Apple
10026 Roe Ave.
Overland Park, Kans.
66207 U.S.A.
Source Mail:
TCF 238
CompuServe:
70120,202