Open-Apple™

October 1985
Vol. 1, No. 9

Apple releases new peripherals

Apple, Inc. announced several new peripherals for the Apple II family on September 17—just in time for the Christmas season. The new products include two composite color monitors that can display 80-column text, an upgraded Imagewriter II printer, and an 800K double-sided 3-1/2 inch disk drive. Not included in the announcement, though widely expected, was the AppleTalk Network card. Also announced, but without a price or availability date, was a 256K memory card, which is expandable to 1 megabyte.

At the same time, Apple announced that its sales during July and August (in the wake of this summer's Atari and Commodore product introductions) were stronger than it had expected. Apple said it was reinstating plans for a big Christmas advertising campaign, which had been cancelled earlier this year after Apple reported its first quarterly loss (see our July issue, page 49).

In my opinion, the most significant of the new Apple products are the color monitors—assuming they do what Apple's press releases say they will. These monitors, which will sell for about $400, display both color and 80-column text, yet hook up to the standard Apple video connector. Previously, the only way to get both color and 80-column text on the same screen was with RGB (red-green-blue) equipment. This was significantly more expensive. Apple announced that its new monitors—one matches the IIe styling, the other IIc styling—would *replace* its $600 RGB monitor. I hear bells tolling for RGB.

Next in significance was the lack of an AppleTalk announcement. Earlier in September Apple sent letters to people who had purchased the AppleTalk Developer's Notes for the Apple II, refunding their money and saying that the notes should be disregarded. Apparently early developers had complained about the card's performance and Apple decided to redesign it. "This redesign will allow developers to shorten and ease their design cycle," the letter said.

The new $600 Imagewriter II printer has four significant enhancements. A fast 250 character-per-second draft mode and a slow 45 character-per-second near-letter-quality mode were added to the Imagewriter's standard 180 character-per-second operation. With a special $14 ribbon, the Imagewriter II will print in color. It includes an expansion slot and Apple is producing a $100 32K memory board for the slot. Also available is a $225 single-sheet feeder. It also has one significant de-enhancement; the cable that used to come with the printer is now $30 extra.

The most insignificant products are the new "UniDisk 3.5" and the "Apple II Memory Expansion Card." The only bright things that can be said about these products is that they work on all models of the Apple II. The price and availability of the memory expansion card had not even been set at the time of Apple's announcement.

The expansion card fits in a standard Apple II slot and appears to be useful primarily as a RAM disk. Almost all of the software developers who say they are supporting the card are using it this way. Its addressing scheme is totally different from any other Apple II memory card on the market. It reportedly does not use bank switching. To read to or write from the card you repeatedly peek or poke at the same memory address in the softswitch area for the slot the card is in. The card provides a serial stream of data in response. You can move the "position in card" pointer elsewhere by poking a couple of other softswitches on the card.

Meanwhile, Applied Engineering has dropped the price of its 1-megabyte auxiliary-slot RAM card to $519 and announced that it can be expanded to 2.5 megabytes. Checkmate Technology has announced a 768K board for the Apple IIe and a 512K board the Apple IIc that include 65816 microprocessors. And Apple made no announcements about supporting AppleWorks with its memory card; something both Applied Engineering and Checkmate have accomplished. What's the big deal?

Apple's memory card almost looks significant, however, when compared to the new 3-1/2 inch disk drive. The UniDisk 3.5 costs $500 and requires a separate $69 controller card when used on a II-Plus or IIe. This controller is incompatible with 5-1/2 inch drives. The new drive does plug directly into a IIc, but the IIc requires a modification to its internal disk control firmware before the new drive will work. Apple dealers will make this modification, apparently without charge, when a IIc owner purchases the new drive.

The new drive does not support DOS 3.3 or CP/M. Programs written in Pascal must be revised to use it.

Compare the 800K UniDisk 3.5 at $568 to a 10,000K Sider hard drive at $700 and tell me how Apple expects to sell these things. Throw in the cost of a box of the special double-sided 3-1/2 inch disks the new drive requires and you are within $100 of the Sider.

The Sider supports ProDOS, DOS 3.3, Pascal, and CP/M; the UniDisk 3.5 supports only ProDOS. The Sider is three times faster than a standard floppy; the UniDisk only two times faster.

Perhaps realizing its problems, Apple notes in its press release on the UniDisk 3.5 that the drive's "mass storage capability also makes it a cost-effective backup solution for hard disk drive users."

You want a cost-effective backup solution for a Sider? Buy another one. It's cheaper and easier than anything else you can do. In addition, not only do you have your *data* backed up; you have your *equipment* backed up as well.

The UniDisk 3.5 may find a niche market as a ProDOS-only add-on drive for the Apple IIc. I can think of no other cost-effective use for it.

"HE'S LOST INTEREST IN COMPUTERS SINCE A LARGE CORPORATION BROKE INTO HIS SYSTEM AND ERASED ALL HIS BOOK REPORTS."

Butterflies turn to worms

What I like best about Logo and the Logo philosophy is that bugs are considered opportunities rather than disasters. A bug encounter to a Logo programmer is a learning experience.

I've had a lot of these learning experiences this month. Several of the public domain butterflies ***Open-Apple*** set loose upon the world earlier this year have turned to worms during the last 30 days. In this month's issue we'll investigate what happened and all come out better people for it.

Input anything bugs. Last month's *Input Absolutely Anything* program and article had both dumb mistakes and exotic ones. Turn, if you will, to page 67 and look at the second program on that page, *Install IAA.OBJ*. Look at the amazing sequence of line numbers given. We have scientifically determined that the program will work a lot better if you renumber the third through the seventh lines. Make the third line 115 and increment the line numbers by 5 until you get to line 130. The new line 125 and the old line 160 should also have the word REM after their colons (and stop that tittering).

Moving beyond pure stupidity, a much more interesting discovery about *Input Absolutely Anything* is that, under DOS 3.3, strings don't evaluate correctly with the VAL function, just as they didn't with Crossley's original input-anything routine, as mentioned on page 66. The problem is the same as Crossley's—the string characters are being stored with the high-bit set (high-value ASCII format) rather than with the high-bit clear as Applesoft expects. In addition to problems with the VAL function, string comparisions don't work reliably when some strings are in high-value ASCII and some in low-value.

(On page 66, I said this problem was originally solved in the June 1980 *Call -A.P.P.L.E.* by Eric Goez. This statement was an interpolated estimate that also turned out wrong. The Goez article, now that I have actually found it, was an Applesoft input-anything routine based on the GET command. The first input-anything routine I can find that solved the VAL problem was, appropriately enough, Val Golding's. It appeared in the July 1980 *Call -A.P.P.L.E.*.)

The difference between Crossley and me, of course, is that I knew about the potential high-bit problem. My routine takes pains to clear a character's high bit before storing it in the input buffer. I realized I was in big trouble, however, when I got the following letter from Jack James of Winfield, Ill.:

I recently wrote my own "Input Everything" routine for the same reasons you gave for writing yours. My immediate problem was comparing two text files holding LISTed versions of an Applesoft program. The program was a commercial product that packed lots of commands into single statements. The LISTed version of these statements was often longer than 255 characters. The program also included embedded backspaces and other control characters, some of which were intercepted by NXTCHAR and treated as editing commands.

My program, like yours, is based on Peter Meyer's "Ultimate Input-Nearly-Anything Routine". I replaced the NXTCHAR call with Applesoft's INCHR routine ($D553) rather than the Monitor's RDKEY. INCHR calls RDKEY and then clears the high bit of the returned character. I used COUT, as you do, to echo the character and, for disk reads, to satisfy DOS's eat-the-character mechanism.

You didn't point out that our routines will read all 128 ASCII characters, including control-@ (ASCII 0), which GET will not handle. (This character must have the high-bit set when it is data on the disk. This sets it apart from the text file terminator, which is a true zero—high bit clear.)

*You were also rather cryptic about dealing with strings longer than 255 characters. These routines blindly return the first 255 characters of long strings without knowing whether the 256th is a carriage return. This means that an application program that wants to handle long strings must check the returned string length. If it is 255, there is **always** more and a subsequent read must be done. If the string is, in fact, exactly 255 characters long, that read will yield a lone carriage return, and, hence, a null string. Otherwise, of course, you'll get more.*

*In the process of developing my routine, I found (with the help of Lee Meador, Call -A.P.P.L.E.'s DOS consultant in Dallas) an unusual characteristic of DOS 3.3. Although I was clearing the high bit of each character before putting it into the input buffer at $200, the high bit was **set** on all but the last character of my returned strings. It turns out that DOS 3.3 expects page 2 to be used for the input buffer and goes through the trouble to make sure any characters it places there have the high bit set. Each time I asked DOS to read another character from the file, it would set the high bit on the **previous** character in the page-2 input buffer.*

I ended up using Applesoft's GDBUFS ($D539), as Meyer had done, to clear the high bits. The high-bit clear by DOS not only undid what I had already done, but it carries the potential of clobbering page 2 if you are using it for something else.

So it's Uncle DOS himself who's spoiling **Open-Apple's** input-anything routine. (And I thought we were friends.) The offending code inside DOS 3.3 (this isn't a problem under ProDOS) lives at $A64A-$A65D and is executed every time READ gets a character from the disk.

Here's a complete Applesoft fix for this problem. Use these lines in place of the originals given in September's *Create IAA.OBJ* (page 67—**enter B where you see b; enter 8 where you see B or 8**):

```
100 C$="0300: A9 B4 85  33 20 E3  DF 20 6C  DD A2 00  20 0C FD" : GOSUB 500
101 C$="030F: A8 29 7F  C9 0D 08  48 F0 07  C9 20 b0  03 69 40" : GOSUB 500
102 C$="031E: A8 98 20  ED FD 68  28 F0 0D  9D 00 02  E8 E0 FF" : GOSUB 500
103 C$="032D: D0 DD A9  0D 20 ED  FD 86 FD  20 39 D5  A6 FD BA" : GOSUB 500
104 C$="033C: A0 00 D1  83 F0 02  b0 1F 91  83 C8 b1  83 85 71" : GOSUB 500
105 C$="034b: 48 C8 b1  83 85 72  C5 70 68  90 0D C5  6F 90 09" : GOSUB 500
106 C$="035A: 8A A2 00  A0 02 20  E2 E5 60  8A 20 52  E4 A2 00" : GOSUB 500
107 C$="0369: A0 02 20  E2 E5 A0  00 A5 FD  91 83 C8  A5 6F 91" : GOSUB 500
108 C$="0378: 83 C8 A5  70 91 83  60" : GOSUB 500
110 FOR I=768 TO 894 : X=X+PEEK(I) : NEXT
120 IF X=16361 THEN 200
```

Assembly language programmers might want to make a note in the source code on page 68 of last month's letter. Just before the TXA command at byte $333 enter this:

```
JSR $D539   Applesoft GDBUFS clears high bits that DOS 3.3 messed up
LDX LENINS  recover string length
```

Leave the TXA instruction intact; it may seem quicker to just LDA LENINS, but both X and A must have the string length for the program to work correctly later on.

The routine James sent me is executed with a CALL command rather than with the ampersand my routine used. His Applesoft syntax is to put a comma after the CALL, followed by the string variable, like this:

```
CALL 768,A$
```

He executes this with three calls to Applesoft machine language routines:

```
JSR CHKCOM  ($DEBE) looks for and eats comma after CALL command
JSR PTRGET  ($DFE3) digs user's variable out of program
JSR CHKSTR  ($DD6C) makes sure variable is a string variable
```

If you would prefer to use the CALL syntax, all you have to do is add the JSR COMCHK instruction before the JSR PTRGET in my program. The program assembles to run at $300 (CALL 768), but can be loaded anywhere that is convenient without reassembly.

I didn't know about CHKSTR. As presented last month, *Input Absolutely Anything* will choke if you give it a numeric variable instead of a string variable. For example, "& A" instead of "& A$" will cause problems. James avoids these problems by the call to CHKSTR, which will return a TYPE MISMATCH error if the variable isn't a string variable.

The new program lines for *Create IAA.OBJ* include the string-checking call to CHKSTR and the fix-the-high-bits call to GDBUFS. The byte that controls the maximum length of strings (originally mentioned on page 68, second paragraph) is now at byte 812 ($32C). Sorry for the confusion. But chin-up, it gets worse.

Basic Basic copy program can't copy Basic programs. Geeesh, can you believe this one? Doesn't *anyone* test stuff before publishing it? July's basic Basic copy program (page 51) *copies* Applesoft programs just fine (I tested that thoroughly), it's just that these copies won't RUN and won't LIST correctly (as several subscribers have pointed out).

The problem here is that ***Open-Apple***'s basic Basic copy program does nothing to correct the *AUX_TYPE* entry in the copied file's directory. AUX_TYPE holds various kinds of information, depending on the file's type. Applesoft program files use AUX_TYPE to store the address the program was at when it was saved.

For more details about AUX_TYPE, see my comments after the letter *BSAVE, T, and CREATE* in the August issue, page 63.

July's Basic copy program always leaves AUX_TYPE set to zero. To understand the problems this causes, you have to know what an Applesoft program line looks like in memory. Enter the following program and follow along:

```
NEW
10 INPUT A$
20 PRINT A$
30 END

CALL-151

300.818

0800- 00 09 08 0A 00 B4 41 24
0808- 00 11 08 14 00 BA 41 24
0810- 00 17 08 1E 00 B0 00 00
0818- 00
```

As stored in memory, Applesoft program lines are separated by zeros. All programs must also *begin* with a zero. Enter POKE 2048,1 : RUN and notice the crazy error messages you get. This is because byte 2048 ($800) is the first byte of our Applesoft program (and most others). POKE 2048,0 is quite effective therapy for this mindlessness.

The two bytes after the zero in every program line give the address of the next program line. Thus, the 09 08 in bytes $801-802 indicate the second program line is at $0809. The 11 08 there shows the third line starts at $811, and so on.

The next two bytes of a program line indicate the line number. The 0A 00 at bytes $803-804 is hexadecmial for 10, the 14 00 at $80A-80B is hex for 20, and so on. This is followed by a series of *tokens* and ASCII characters. The token for INPUT is $84, as you can see at byte $805. At byte $80D you can see PRINT's $BA token, and at byte $815 END's $80. The Applesoft manuals have lists of the tokens for all Applesoft commands.

Note that the third line of the program points to a non-existant fourth line at $817. Bytes $817-818 both hold zeros. This pair of zeros, along with the zero separator, marks the end of the program.

When a program is saved while residing at one location and then rerun at a new location, all of the next-line pointers have to be recalculated or the program won't execute. DOS 3.3 calls a routine in Applesoft at $D4F2 that handles this. This Applesoft routine ends, however, by warmstarting Applesoft rather than by returning to its caller; DOS 3.3 has to pull some severe brute force tricks to get control back after calling the routine.

Unlike DOS 3.3, Basic.system simply resets the pointers itself. It calculates the difference between the program's current address and the address it was saved at and adds that difference to all the pointers. Obviously, if AUX_TYPE says a program lived at $0000 when it was saved, this will mess up the loaded version. Programs subjected to such treatment neither RUN nor LIST properly.

As noted in July (page 52) the incorrect AUX_TYPE also causes problems with text and binary files. I squirmed out of all this then with the line, "These problems are fixable, but not this month." But, alas, now my bugs have come home to roost.

The difficulty here—the reason I weasled around last time—is that there is no way to directly reset a file's AUX_TYPE bytes with Basic.system commands. We have to use the ProDOS Machine Language Interface for this.

The ProDOS machine language interface. The MLI, as its name implies, expects to be called from machine or assembly language. In general, you get ready for a call by setting up a parameter table somewhere in memory that holds the information ProDOS will need to execute the command. Then you jump to a subroutine at the location known as MLIENTRY, which is at the beginning of the ProDOS global page ($BF00). (The ProDOS global page is a 256-byte area the ProDOS kernel uses for storing various pieces of information that application programs sometimes need access to.)

Each ProDOS command has a specific parameter-table format. These vary from one command to another. You tell the ProDOS kernel which command you want to execute and where in memory you've stored its parameter table by embedding this information in your machine language program. The command byte and two bytes that point to the parameter table follow the JSR to $BF00, like this:

```
JSR MLIENTRY     ($BF00)
.DA #$C4         command
.DA $BEB4        address of parameter table
BCS ERROR        your error-handling routine
```

The ProDOS kernel will dig the command and parameter table address out of your program and return back to your next instruction, which should always check for an error.

Trying to use the machine language interface from inside an Applesoft program can get very complex, but I finally found a way we can avoid most of the complications (this time). Basic.system has its own global page (at $BE00), which includes a routine called GOSYSTEM. Basic.system always uses GOSYSTEM to call the machine language interface. In addition, the Basic.system global page contains all of the parameter tables that Basic.system itself uses when calling the kernel.

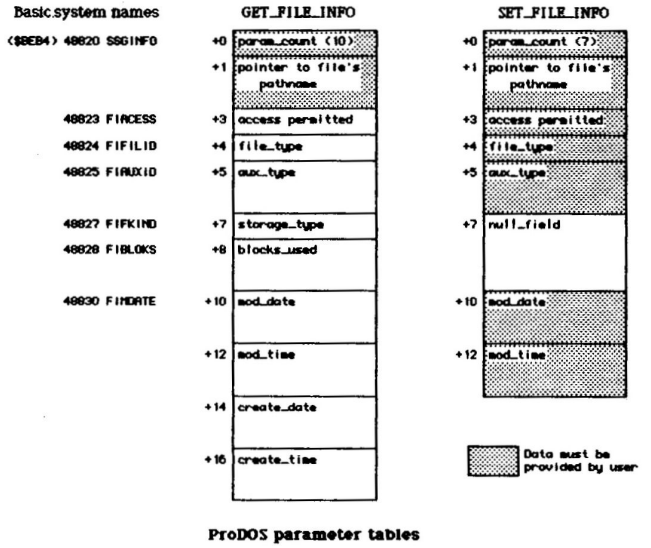
Unlike the MLI, which expects an embedded command, GOSYSTEM expects the command number to be in the A register. It embeds it where the kernel will find it later. Then, using this same command number as an index, GOSYSTEM figures out where in the global page the parameter table for that command is. It embeds that address also, then does its JSR to the MLI. On return, GOSYSTEM checks to see if an error occurred; if so, it converts the error number returned by the kernel into a Basic.system equivalent before passing control back to the caller. It does no other error handling, however.

Basic.system doesn't care if we use its parameter tables for our own MLI calls. By poking a short machine language routine into some borrowed space in the Basic.system global page, in fact, we can use GOSYSTEM to make MLI calls for us from Applesoft. This is what keeps the number of required pokes to a minimum.

Inside the Basic.system global page is a 19 byte area ($BE58-BE6B) that is used only when a DOS command is being executed. The numeric values of the command parameters (the $2000 in BLOAD RED.LEAVES,A$2000, for example) are stored here when a command is parsed or translated into DOS talk. As long as a command isn't being executed, we can temporarily borrow some space in this area for a small machine language routine. Here's the one we'll use:

```
BE58:A9 00      LDA #$command number
BE5A:20 70 BE   JSR GOSYSTEM ($BE70)
BE5D:B0 AA      BCS ERROUT   ($BE09--entry to Basic.system's error handler)
BE5F:60         RTS
```

The two MLI commands that are used to diddle with AUX_TYPE (or other information stored in a file's directory entry) are GET_FILE_INFO ($C4), and SET_FILE_INFO ($C3). Since the two parameter tables used by these commands are nearly identical, Basic.system uses the global page area from byte 48820 ($BEB4) to byte 48837 ($BEC5) for both of them. Here's what the tables look like:



ProDOS parameter tables

GET_FILE_INFO tells you what is currently stored in a file's directory entry. The GET_FILE_INFO parameter table holds ten items. Because some of this information can't be changed, the SET_FILE_INFO parameter table holds just seven of these ten. The first item in each table, the parameter count, indicates how many parameters are in the table. The ProDOS kernel uses this count to make sure the command and parameter table combination it has been given are compatible.

All the kernel needs to execute GET_FILE_INFO is a parameter count of ten and a pointer to the pathname of the file being examined. The kernel itself fills out the rest of the table. For SET_FILE_INFO, on the other hand, the kernel wants the parameter count to be seven and expects the rest of the table to hold the values to be stored in the file's directory entry.

Poking all this into the parameter table could be quite tedious; particularly the pathname-related pokes. Fortunately, however, from inside the basic Basic copy program we can make a few assumptions about what Basic.system has left in this particular table. For example, after either a BLOAD or BSAVE command, the pathname pointer will, in fact, point to the pathname of the file that was just accessed. This will not be true for long, however—Basic.system uses the pathname buffer for lots of stuff—but it will be true long enough to solve our problems.

To fix the copy program, we'll call GET_FILE_INFO immediately after BLOADing the file to recover the original file's true AUX_TYPE; after BSAVEing the copy of the file we'll poke the original file's AUX_TYPE into the parameter table and call SET_FILE_INFO to fix the duplicate's directory entry. The additional lines you need to add to July's program (page 51) are:

```
405 X1=0 : X2=0 : CMD=0

455 CMD=196 : GOSUB 600 : X1=PEEK(48825) : X2=PEEK(48826)
475 CMD=195 : POKE 48820,7 : POKE 48825,X1 : POKE 48826,X2 : GOSUB 600

600 REM  *** MLI caller for use with Basic.system ***
601 POKE 48728,169 : POKE 48729,CMD : POKE 48730,32
602 POKE 48731,112 : POKE 48732,190 : POKE 48733,176
603 POKE 48734,170 : POKE 48735,96  : CALL 48728 : RETURN
```

Line 405 allocates memory for the three new variables we have to use (X1, X2, CMD). Line 455, which is executed immediately after the original file is BLOADed, does a GET_FILE_INFO on the file and saves the AUX_TYPE bytes as X1 and X2.

Line 475 pokes these values back into the parameter table after the new file is BSAVEd, and does a SET_FILE_INFO to update the new file's AUX_TYPE field.

Lines 600 to 603 make a subroutine that pokes our little MLI caller into the Basic.system global page and executes it. Note that CMD must be given the value of a ProDOS command before using this subroutine.

***Incinerator* burns up page zero.** I've taken a lot of grief about subscribers not being able to tell the Bs from the 8s in March's *Incinerator* program (pages 17-19) but otherwise it seemed bullet-proof. The *Incinerator* is a high-speed garbage collector for DOS 3.3 written by Bill Basham of DiversiDOS fame. However, this month a subscriber discovered that his DOS commands wouldn't execute and his program wouldn't LIST when the *Incinerator* was in use and memory was almost full.

I traced the problem to the modifications I made to the *Incinerator*. Those modifications made the program automatically check the garbage cans every time a return was printed and empty them if necessary. Basham's original was called manually with an ampersand command.

His technique works better. The *Incinerator* uses a large number of zero-page locations whenever garbage is actually collected. If collection occurs while something else (such as a LIST or PRINT command) is using those locations, problems will occur.

Our subscriber's program was using the common trick of defining D$ as CHR$(13) + CHR$(4). This technique, which works only with DOS 3.3, assures everyone that there is always a carriage return before control-Ds. However, in this case the CHR$(13)s also forced garbage collection even as Dr Basic was in the midst of printing DOS commands. This messed up enough zero-page locations that the DOS commands weren't executed.

There are two possible solutions. One is to leave the *Incinerator* connected to DOS as it is now. However, test the potential negative effects garbage collection could have on your program by poking a zero at byte 48816 ($BEB0). When there's a zero at this location, complete garbage collection will occur every time a return is printed. If the program executes properly, it is compatible with the *Incinerator* as written. Change byte 48816 back to a four and go. Don't expect the LIST command or the D$ = CHR$(13) + CHR$(4) trick to work, however.

The other solution is to BLOAD the *Incinerator* at $4000 and change bytes 16517-18-19 to 234. This will prevent the high-speed garbage collection routines from being connected to DOS. To run them, you can either connect them to the ampersand hook or execute them directly with a CALL 48815 ($BEAF). This last technique is very dangerous, however. *That particular call will initialize and erase the disk in the active drive if the Incinerator has not been installed.* Perhaps a better syntax would be IF PEEK(48815)=169 THEN CALL 48815. That makes sure the *Incinerator* is what's at 48815 rather than the usual RWTS INIT routine.

Miscellanea

Apple II hacker profile. Apple's marketing experts say that hackers—the kind of people who read ***Open-Apple***—are mostly young, underfunded, and not responsible for a significant portion of Apple II sales. Surprise, they're wrong. *Call -A.P.P.L.E.* recently did a survey (I suspect ***Open-Apple*** readers are very similar) and discovered that their readers' median age is 42; three-fourths are college graduates (45 per cent have master's degrees or higher); nearly two-thirds have incomes above $40,000 (44 per cent have incomes above $50,000); more than two-thirds own or use two or more computers (35 per cent own or use 3 or more; 20 per cent, 4 or more); and two-fifths of them authorize the purchase of computer equipment for their employer or organization. Ninety-one per cent are males; 48 per cent have spouses who use their computer.

More than three-fourths of *Call -A.P.P.L.E.*'s readers are programmers. About 72 per cent consider themselves *intermediate or expert* in Basic; 24 per cent in assembly language; 19 per cent in Pascal; and 3 per cent in Logo. Of course, 100 per cent of them use Apples. This is a group Apple should want to keep happy, but when was the last time you saw Apple's marketing wizards place an ad in this magazine?

RAM prices are dropping rapidly. You can save big money by buying RAM cards with a minimum amount of memory and installing your own chips. This month you could get a whole megabyte of 256K chips from Microprocessors Unlimited (24000 S Peoria Ave, Beggs, Okla. 74421, 918-267-4961) for only $86.08. These were NEC 150-nanosecond 256K chips at $2.69 each. (It takes eight of them to make 256K *bytes* of memory). These NEC chips are what Applied Engineering put in the last RamWorks card I got from them. Apple's new memory card had Hitachi 200-nanosecond chips in the photo they sent me. Hitachi chips are a little more expensive, but they supposedly run cooler. (The nanosecond rating refers to the chip's response time. It's appended to the chip's part number. For example, "-15" indicates a 150-nanosecond chip.) Microprocessors Unlimited includes with each order very specific instructions for installing the chips without zapping them with static electricity.

A Value Added Reseller or VAR is someone who buys computer equipment directly from the manufacturer, adds software or special hardware to create a system dedicated to a specific use, and resells it to end users. In the July issue (page 50) I suggested Apple should expend more effort cultivating these kind of people. Before that issue even hit the mail I got an unsolicited brochure from Apple describing their VAR program. All Apple VAR sales are handled by a sales agent, Professional Computer Marketing Associates. For more information, contact PCMAS's Dirk Eastman at 800-821-1779 (800-824-6277 in California) or Tom Babecky, Apple's VAR manager in Cupertino.

Franklin Computer seems to have resuscitated itself. It has announced a new Apple IIe/IIc-compatible ACE 2000 that will retail for $699. The machine reportedly has a steel case, 90-key detachable keyboard with numeric keypad, 67-watt power supply, internal fan, and the Franklin equivalent of Apple's MouseText characters. The machine is made in Taiwan; Franklin says it only has to sell a couple of thousand a month to make money. Schools are the targeted buyers.

Steve Jobs has taken his Macintosh development team and gone home. The Apple II would never have become world-famous without him, but his contributions to the Apple II family ended years ago. The only Apple computer that has ever been a commercial success is the one Steve Wozniak designed. The mark of Jobs—a closed system and lots of publicity—just doesn't sell. Typically, Jobs turned his letter of resignation in to several newspapers before delivering it to Apple vice-chairman A.C. Markkula, according to the *Wall Street Journal*.

Jobs' biggest contribution to Apple was its classy image. I had hoped Sculley would retain that. However, late word is that the women trading gold futures, playing basketball, and carrying sledgehammers are slated to disappear from Apple's ads in favor of copy like, "Your suit is a vice-presidential shade of blue. Impeccably tailored: jacket sleeve five inches off the tip of your thumb. Your tie discreetly suggests the position you want, as opposed to the position you've got." (Please excuse me while I lose my lunch.) I suspect this campaign will lessen Apple's sales to traditional customers more than it will increase sales to clotheshorses. The world goes on. Even Sculley's former foe Coca-Cola abandoned the real thing.

Ask (or tell) Uncle DOS

Apple Assembly Line

You've mentioned the publication *Apple Assembly Line* several times but I've been unable to find its address. Would you please mention it?

Tony Alley
APO N.Y.

Apple Assembly Line *is a monthly newsletter published by S-C Software; P.O. Box 280300; Dallas, TX 75228; 214-324-2050. It has been covering assembly language programming for the Apple II since October, 1980. The current prices are $18 per year for bulk mail, $21 for first class mail, and $32 for international airmail. (It's heavy stuff.)*

S-C Software also produces an assembler for the Apple II and sells various books and software helpful to assembly language programmers and learners. If you need assistance with assembly language on the Apple, these folks have products and experience that can help. I have done business with them for several years and find them completely reliable.

A few weeks ago I ordered the ProDOS update to S-C's assembler and got a copy of the June 1984 ***Assembly Line****, which had been missing from my collection. It disclosed the rather amazing news that the POKE 72,0 ($48:0) trick for avoiding trouble with the Monitor's G(o) command doesn't always work. When you issue a G(o) command from within the Monitor, the contents of byte 72 are placed in the 6502's status register. This was discussed in greater detail here in March (page 20) and April (pages 30-31).*

If byte 72 contains garbage when the G(o) command is executed, your computer can begin to do very strange things. For years experts have recommended putting a zero in 72 before exiting the Monitor with the G(o) command. But now it turns out that a zero in 72 will enable interrupts. Interrupts have been available on the Apple since 1977, but were rarely used until the last couple of years. Consequently, no one noticed this interaction before.

Nowadays, however, it would be important on many systems. ***Apple Assembly Line*** *recommends poking a 4 in 72 rather than a zero. The most frequent use of POKE 72,0 in* ***Open-Apple*** *has been as part of the S.H. Lam technique. It might be worth your time to correct line 520 (to POKE 72,4) in the programs on pages 13, 24, 43, and 55 if you save our back issues.*

Apple II phone machine

I wrote a program that answers my phone...big hairy deal...anyone with a modem can do that. But this isn't any ordinary modem, it's a Novation Apple Cat. Why anyone buys a Hayes I do not know for this thing does what a Hayes can do for about the same price plus oodles more. In the Novation manual there's a demo program that shows how to have the modem answer the phone with speech using a Votrax system wired to the auxiliary connector on the card (you won't find that on a Hayes). Once connected, the program would get the modem to answer the phone and have the Votrax speak to the person on the other end.

Since I don't have a Votrax I looked around for some other way and found one. The synthesizer is called *Software Automatic Mouth (S.A.M)*. It consists of a card and a software routine. Just by accident (honestly!) I set the slot number for speech output to my modem slot. Low-and-behold voice over my phone WITHOUT wires and no need for the SAM card! I was estatic...my fingers went wild at the keyboard. I wrote a program I call the *Answering Machine*, then the *Originate Machine* (a phone number dial-out database). When I put the two together I had *Phone Functions*, which won first place at the computer fair here in Whitby.

Andrew Reeves-Hall
19 Glenmount Ct
Whitby, ONT L1N 5M7 Canada

I find this kind of software very exciting. I've included your address so that people who want more information can contact you directly. I'm convinced that the Apple II can be turned into a telephone that would give AT&T and other telephone manufacturers the heebie-jeebies. (Here's something interesting to do with an Apple II-Plus.)

For several years now the major Savings and Loan in Kansas, Capitol Federal, has offered a telephone bill payment service. It consists of a synthetic voice that answers the phone and tells you what to enter. You respond by pushing buttons on a touch-tone phone. Systems like this can now be built out of Apple IIs.

They could be used to take orders, to quote prices, to leave messages, and to retrieve messages. And all this could be done at a price the big manufacturers couldn't touch. I think I hear opportunity knocking.

Another single-drive CONVERT

I find the IIc /UTILITIES disk unsatisfactory for converting files between ProDOS and DOS 3.3 on my single-drive Apple IIc. Since the program will only convert an entire disk, a lot of disk switching is involved just to convert a single file.

I couldn't get CONVERT from the /USERS.DISK to work either. You presented one solution for this problem in the September issue (page 72). My solution involves an extra step, but it still seems to save time: use the ProDOS /RAM disk. To convert DOS 3.3 to ProDOS, use /RAM as the destination prefix. When finished with CONVERT, run FILER to copy the file from /RAM to your ProDOS disk. To convert the other way, run FILER first to transfer the ProDOS files to /RAM and then convert them onto a DOS 3.3 disk.

Another /RAM disk tip: When starting work on a new program I transfer the utility programs I am likely to need to a new disk (GPLE and APA for Applesoft programs, for example). This often involves several source disks, and could mean some serious disk switching with a single drive. An easier way is to use the /RAM disk as a staging area: use FILER to copy all the individual files or programs from their respective disks to /RAM and then copy them from /RAM to a fresh disk with the "=" wildcard.

Andrew K. Messersmith
Ft. Lauderdale, Fla.

Auxiliary memory fix

At the junior college where my job is to keep the Apples running, we've had several IIes develop problems with auxiliary memory. Symptoms included machines locking up, unusual characters showing up in *Apple Writer* files, and programs failing to detect the second 64K bank of memory.

The manual that came with Applied Engineering's Z-80 card tipped me off to check the chip at position B-2 on the IIe motherboard. This chip is a 74 LS-245. Apparently the quality of this chip varies greatly from manufacturer to manufacturer. Applied Engineering suggests replacing this chip with one made by Texas Instruments. I have found that this simple change fixes auxiliary memory problems at least half the time.

As it happens, there is usually a Texas Instruments 74 LS-245 at position B-4 on Apple auxiliary memory cards. I've had good results by just switching these two chips, though it would probably be better to buy a new Texas Instruments chip for the motherboard.

When these problems started, I also called the *Call -A.P.P.L.E.* hotline and was told a few Apple IIes were manufactured with a 74 LS-109 at motherboard position C-1. This chip is supposed to be a 74 S-109 or 74 F-109; the LS-109 won't work here. I haven't actually found any IIes with this problem, however.

Jim Luther
Kansas City, Mo.

Bulletin board follow-up

Thanks very much for including my questions and comments in the July issue of ***Open-Apple*** (page 53). We ordered a Sider hard drive for my department's lab and it arrived a few weeks ago in a very well-packed box. It almost seemed as though they were expecting it to be handled by the gorilla in the Samsonite luggage commercial, though thankfully this wasn't the case. We now have a ProDOS version of the GBBS bulletin board you mentioned purring along (after a few minor adjustments) on the Sider with a Hayes Smartmodem 1200.

The people at Micro Data were very helpful in getting their bulletin board on line. I also had to call technical support at First Class Peripherals and found them helpful, too.

I obtained a IIe enhancement kit and greedily popped in the new ROMs, expecting nothing less than a fanfare and fireworks upon boot-up. That didn't happen (and I'm glad), but I got the equivalent bang for the buck in terms of new features and enhancements. The direct entry of ASCII comes in handy, and many bugs were fixed. I like the new self-test routine very much. According to the source listing on page 66 of the *Enhanced IIe Programmer's Guide*, the new self-test covers both main and auxiliary memory and the MMU and IOU chips as well.

A friend mentioned that the screen-clear time when using a terminal-emulation program seemed much improved with the enhanced chips, so I set up a little test and found the new routines are over 3.5 times faster.

I read somewhere that rerunning Basic.system is the ProDOS equivalent of DOS 3.3's FP. Does doing this also reset all the MLI links and the global page? I ask this because I often edit my bulletin obard with the ProDOS version of GPLE, then execute Basic.system again so GPLE won't cause hiccups with GBBS. Is the proper way, or should I just power down and restart?

Peter Chin
Brooklyn, N.Y.

The ProDOS equivalent of FP is "-BASIC.SYSTEM". This will reload Basic.system and set up a new copy of the Basic.system global page. It will also reset the memory-protection bitmap in the ProDOS global page, however, it doesn't fix any other changes that might have been made there (none should be, but who knows?) If this might be a problem, you can do a "-PRODOS" to restart completely anew without actually powering down.

Thanks for pointing out the source-code-embedded description of the enhanced IIe self-test, which I hadn't noticed before. It says the test will run continuously as long as the open- and closed-apple keys are both depressed or the keyboard is disconnected. At the end of a successful test a "System OK" message will appear. You can execute another test cycle by pressing both apple keys again or you can exit the diagnostic routines by pressing control-reset.

Releasing the power

When I read my first issue of ***Open-Apple***, I was pleased enough to order all the back issues. I said more "Amens" reading them than I usually say at my (Baptist) church. The folks at Apple, Inc. should not read ***Open-Apple***, they should *memorize* it. There's more sense in your columns in the April and July issues than in all the collected wisdom of Apple's management, if their blunders are any indication. One hopes that the reorganization will help. But why don't they save the money and simply implement your columns?

At the university where I teach, our department (sociology—not computer science, of course) has finally persuaded the powers-that-be to begin a microcomputer lab with a dozen Apple IIes. And, at last, our computer center director (who is wedded to Hewlett-Packard) has agreed to help us interface the Apples with the HP mainframe. Naturally, it turned out there was already software to do just that. Therefore, we will have full access to all the computing power of the mainframe as *needed* (and at relatively low cost compared to buying HP equipment).

To me, this is the future: good "old" Apple IIs serving as the base of ever-expanding computer capability through enhancements and interfaces to larger computers when necessary. Most of the time, the Apples can do the job off-line. Today the HP mainframe was "down." As my colleagues sighed and fumed, I happily worked away with my word processor and data files on my Apple.

Long ago, I realized that computing was as much political as technical. I realized that the computer professionals often had a vested interest in the user's ignorance and dependence on the "experts." And that is the great thing about the Apple-led, microcomputer revolution—it is "releasing the power to everyone."

Alan G. Hill
Greenville, S.C.

Amen.

Call for Apple Writer calling help

I have a couple of questions about *Apple Writer 2.0*. One of the reasons I upgraded to the ProDOS version was for the communications capability, but I haven't been able to figure out how to get it to work with my modem (this is embarrassing because I consider myself a knowledgable Apple user). The *Apple Writer* manual provided no help at all ("Call your receiving party using the instructions that came with your modem..." Does that mean I have to exit to Basic to use my modem with *Apple Writer*? I have a Hayes Micromodem IIe and would simply like to dial the number of a local BBS and use *Apple Writer* instead of *Smartcom I*.

I would also like to know if there is any way to get *Apple Writer 2.0* to date files without a clock card, the way AppleWorks and *SuperCalc 3A* do.

My last comment is on *SuperCalc 3A*. It is a super program. I recommend it to anybody that needs a powerful spreadsheet for the IIe or the IIc.

James P. Cooney, Jr.
Billings, Mont.

I consider myself a knowledgeable Apple user, too. Maybe that's our problem. I haven't been able to figure out the ***Apple Writer*** *communications option either. Has anyone out there gotten it to work? How did you do it?*

I suspect the problem may be that you and I both have slot-resident modems. Apple's own ***Access II*** *communications software works only with modems connected to a serial card, according to a February 1985 review by Walter Mossberg that appeared in the Washington Apple Pi newsletter. Mossberg calls* ***Access II*** *"one of the worst products Apple has ever offered", in large part because its inability to work with slot-resident modems, by far the leading type of modem used with Apple IIs, isn't revealed on the outside of the package or in the tutorial program. "It is first mentioned more than 20 pages into the manual, and must come as a bitter surprise to many," Mossberg says.*

You can use the STARTUP program on the ProDOS /USERS.DISK to tell ProDOS the date and time. Exit the program and set the prefix to the directory your ***Apple Writer*** *files are in and enter "-AW.SYSTEM". The date and time you set will appear on all files saved that session. You could also modify the /USERS.DISK startup program to create a special* ***Apple Writer*** *startup program.*

SuperCalc 3A *is the most powerful spreadsheet-plus-graphics program for the Apple II that I have found. Its obvious CP/M heritage bothers me, however. Its developers either didn't bother to learn the standard Apple II user interfaces or ignored them. For example, you can't initialize new disks from inside the program. This is standard operating procedure on CP/M systems but quite unusual for commercial Apple II software. Being forced to press the escape key before pointing at cells (while building a formula) is something I find particularly bothersome, but that's probably my own problem. Although the* ***SuperCalc*** *developers are to be congratulated for including the ability to use command or exec files, they left out a true macro capacity like* ***Lotus 1-2-3*** *has. How much longer do we have to wait for that in the Apple II world?*

Okidata 92 meets AppleWorks

Jerry Cline asked how to put an Okidata 92 printer into correspondence quality when using AppleWorks (September, page 71-72).

On the Okidata 92/93 and 192/193 printers, correspondence quality and boldface (emphasized/enhanced) are not the same. Correspondence quality is obtained by sending the printer ESC 1. Emphasized or enhanced is obtained by sending ESC H or ESC T.

The way to get correspondence-quality printing is to incorporate the ESC 1 control codes into the command that govern the characters-per-inch to be printed. The command for 10 characters-per-inch is control-^. Correspondence quality at 10 characters-per-inch can be had by using ESC 1 control-^ as the control code.

Data-processing-quality printing is selected with an ESC 0. I code 12 characters-per-inch as ESC 0 control-\ to give me rough draft copy at high print speeds. I use the slower correspondence quality for finished work.

This allows the boldface command to be used for boldface or another purpose. The other purpose I use is to select an alternate character set. The Okidata 92 has a RAM memory area that can accept a second set of characters. There are software packages (Personal Touch by Okidata, for example) that allow you to load italics, script, greek, math, and other special characters sets. This needs to be done *before* you startup AppleWorks.

The Okidata prints the alternate set when it gets an ESC 2. I define boldface begin as ESC 2 and use the boldface begin command (or control-B from the keyboard) to get the alternate characters. Boldface end is then either ESC 1 or ESC 0.

Bruce W. Ristow
540 Antlers Dr.
Rochester, N.Y. 14618

I've included your complete address so that further Okidata 92—AppleWorks questions can find their way directly to you. The print samples you sent (which included a two-page demonstration of using an alternate character set to print the equations needed in mathematics and physics texts) were beautiful.

Many printers have the ability to use alternate character sets, including Apple's Dot Matrix Printer and Imagewriter. Apple's documentation barely mentions this, however, and certainly doesn't show you how to do it. Programs that provide alternate character sets for various printers are available from Third Wave Technology, 16309 Elsienna Ave, Cleveland, Ohio, 44135 (216-671-8991) and from Vilberg Brothers Computing, P.O. Box 4576, Madison, Wisc. 53711. Both companies have a reputation for reliable products at reasonable prices, but I don't have their current product listings handy.

Apple IIs in real estate

After seeing you recommend *Time Is Money* (September, page 71), I wonder if you could help me.

I have purchased two programs for property management and neither one serves the purpose. They are *The Landlord* and Continental Software's *Property Management*. When I asked Contintental for help they recommended I take the program back to my dealer and get my money back. The situation with the other is similar. Let's say I'm somewhat ticked-off.

Are there any other property management programs that you know of? Neither of the above provide "user-friendly" operation. You almost need another program to identify the entities in the programs.

Currently I am using *Time Is Money* for transaction entries and the AppleWorks data base for tenant, owner, address, rent, and other informational entries. AppleWorks generates my reports. It sure would be nice to have all this wrapped in one program, however.

Al Smith
Fresno, Calif.

I'll open this one up to our subscribers. But I doubt you will find anything much better than the ***Time Is Money****/AppleWorks combination. A company named Intuit (540 University Ave, Palo Alto, Calif. 94301 415-322-0573) recently announced a checkbook program named* ***Quicken*** *($79—128K required) that has two*

important features that I wish ***Time Is Money*** *had—an* ***AppleWorks****-like user interface and the ability to transfer data directly into AppleWorks spreadsheets. As I understand this second feature (it was disabled in the demo disk they sent me), you set up an AppleWorks spreadsheet with special labels for the data you want, then you start up* ***Quicken****, tell it about the spreadsheet file, and it will find it and fill in the data. You return to AppleWorks to see the finished spreadsheet.*

Intuit's program reminds me of the PFS series. Like Software Publishing's stuff it sacrifices features for ease-of-use and it's written in Pascal. It is stricken with the lethargic response typical of Pascal-based Apple II software. ***Quicken*** *was not the name that came to mind when I tried the program.* ***Time Is Money*** *is infinitely better and faster; I sure wish it could pass data to a spreadsheet somehow, however.*

The hard disk life, continued

Thanks for the comment (August, page 63) on the problem I had with the Sider (actually a Datamac, but apparently identical to a Sider). The bad-blocking of the files I experienced by pressing reset while printing an *Apple Writer* file only happened once (that was enough!). This has been the only anomalous behavior I have experienced with this hard disk in several hundred hours of operation. Guess I was lulled into complacency, so I hadn't bothered to back up my files.

When the bad-blocking problem occurred, the disk was on line but not active. I was printing a long *Apple Writer* file, noticed something was wrong, and reflexively hit control-reset to stop the printout. The file in RAM was uncorrupted and I finished my printout, then attempted to pull up another file to find that I was fenced out permanently. I was four levels down in the pathname and lost everything in that subdirectory. Nothing outside that subdirectory was touched, and the drive continued to work as it should have except for the reduced effective size of the drive. When I reformatted, the media certified just as it had the first time I set it up.

I have noticed a problem with the AppleWorks spreadsheet that I have never seen documented. When I rearrange an alphabetized spreadsheet with a sort on a column of numerical scores it always purges isolated but continguous blocks of cells of their formulas so that I must rebuild a large part of the template. Deleting rows or columns will occasionally corrupt other parts of the structure that are in independent regions. Apple is of no help, they only tell me to contact my dealer. Is a new revision upcoming on AppleWorks?

I wonder why Apple doesn't provide an easier way to modify only the tail end of a long path name—the way you can in *Sensible Speller*'s ProDOS version, for example. It's a nuisance to have to type in an entire pathname when you only want to jump to another twig on the same branch. Maybe I've missed something.

Donald Beaty
San Mateo, Calif.

Apple II wizard Ken Kashmarek wrote to say he suspects your hard disk problem was caused by a runaway print-to-disk operation. If the reason you pressed reset was that nothing was coming out on your printer (because you had selected 8 (print-to-disk) as ***Apple Writer****'s "print destination"), reset could easily have caused the directory entry to be mangled. I haven't come up with any explanation for what happened to you that's nearly as plausable.*

I've never experienced or heard of the formula-damage problems you mention with AppleWorks. The current AppleWorks version is 1.2; I don't know if Apple has revisions in the wings but I suspect they do. They certainly make enough money off the program to provide customer support; most marketing-oriented companies would love to have the customer feedback you tried to provide them.

You haven't missed anything that I know about regarding ProDOS pathnames. I agree that the ProDOS kernel should support some kind of command for deleting or replacing the final filename in a prefix.

Another graphic grabber

In the July issue ("Printing Graphics", page 53) you discussed a couple of cards that can interrupt a program at any time and print the graphic display. I'd like to put in a plug for Dark Star Systems' products (78 Robin Hood Way, Greenford, Middlesex UB6 7QW England, Source Mail BCJ456). I have one of their SnapShot copy cards and their Printerrupt and Shuttle software. At the time of purchase (about a year ago), I was a bit hesitant about dealing with a firm based in Europe that had no local representative.

Any fears I had proved absolutely groundless. Dark Star Systems personnel have proved to be helpful and responsive beyond the call of duty. With their card and its associated software packages there seems to be virtually nothing (digital, that is) that you can't get out of your Apple onto onto the screen, disk, or printer.

The copy card itself has 8K of RAM memory into which you load programs such as Printerrupt. This program does the same kinds of things as Thirdware's FingerPrint card and Texprint's Print-it!, but it uses the printer interface card you already have. The Shuttle software lets you run programs under different operating systems simultaneously—you do need, however, 64K of RAM for each such program you wish to have loaded at the same time.

Dark Star now has a representative here in the U.S.: Greengate Productions, 2041 Pioneer Ct, Suite 15, San Mateo, CA 94403, 415-345-3064, Source Mail BCH101.

Paul Pagel
Enfield, Conn.

Any company that makes a copy card on Robin Hood Way deserves success. A comparison of the FingerPrint and Dark Star cards in the July 1985 issue of ***Apple User****, an English Apple magazine, also came down in favor of Dark Star, though the FingerPrint card reviewed was obviously an earlier version than the one we use around here.*

Next program, please

I'm running a IIe with a Sider hard drive. I modified the startup program Apple provides with ProDOS to allow selection of AppleWorks, *Apple Writer*, or *Sensible Speller* in addition to the standard FILER, CONVERT, etc.

Can you—can anyone—give me a way to quit each of those three programs and get directly back to my menu? Open-apple/control/reset does it with detours; I'm looking for the direct route.

Donald A. House
Naugatuck, Conn.

I solved this problem by renaming the first volume on my Sider "/H1" (it's simple—use the ProDOS RENAME command) and putting a second copy of Basic.system, named GO, on that volume. (FP would be another suitable name.)

Commercial-quality ProDOS-compatible software is supposed to issue a "quit" call to the ProDOS kernel at exit. The programs you mention, with the exception of FILER and CONVERT (which few people would say are commercial-quality anyhow), do this. The quit routine inside ProDOS, which Apple calls a "dispatcher," asks you to enter a prefix and a pathname. It's pretty easy to enter /H1 and GO; on my system this brings up my own menu program.

A more advanced technique would be to rewrite the ProDOS dispatcher. The ProDOS developers expected people to do this and have made it pretty easy. The dispatcher code is embedded within the ProDOS kernel at bytes $D100-$D3FF in the second 4K bank of main memory. When ProDOS executes a quit call, it moves a copy of the dispatcher to $1000 and jumps to it.

A better assembly language dispatcher could be written that would automatically run Basic.system and your Applesoft menu program. This would make a nice project for someone. I'd do it myself if I could find a time warp to hide in till it was done. Anyone who wants to try should read Apple's ProDOS Technical Notes #7 and #14 before starting. There's a slight chance your dealer's technicians would have these. They are also available in most user group software libraries on disk /IAC.43.

You can say that again

Joabs or Jahbs?

B. Walters
Boston, Mass.

According to the press relations department at Apple, it's jahbs, as in "jobs for the poor" spoken by someone from Kansas City. But who knows how you Bostonians might pronounce it?

Time short for ProDOS

The ProDOS feature that enables a Thunderclock to date-stamp files expires on December 31, 1987. On January 1, 1988, ProDOS will stamp your files 1 JAN 82. This results from the way that ProDOS derives the year.

The clock that the ProDOS designers decided to support, the Thunderclock, doesn't keep track of what year it is. So ProDOS has to perform a calculation based on the current date to figure out what day of the week it would be if it were 1984. It then subtracts the value found in Thunderclock's day of week register from this value, and uses the resulting offset as an index into a 7-byte lookup table that contains the corresponding years.

In all ProDOS versions to date this table contains the sequence $54, $54, $53, $52, $57, $56, $55. These are the equivalent of decimal 84, 84, 83, 82, 87, 86 and 85 respectively, which are the years in which these versions of ProDOS function properly. To breathe new life into either of these versions of ProDOS, you can change the first four bytes of this table to $5A, $59, $58, $58. This will extend its usefulness to the end of 1990. Keep this trick tucked away in your reference file, in case you are still using the same ProDOS five years from now. Notice that a leap year must be repeated in two successive bytes, and that the table wraps around from the last byte back to the first (years in descending order).

To make a patch, BLOAD PRODOS, A$2000, TSYS. Now you can find the offending bytes in main RAM, fix them up, then BSAVE PRODOS, A$2000, TSYS. Notice that you don't have to give the L parameter when saving—ProDOS will use the one stored in the directory.

When you BLOAD ProDOS at $2000 the table of years is at $5076 in ProDOS 1.0.1 and 1.0.2 and at

$4F76 in versions 1.1 and 1.1.1. After testing the patch to make sure everything works OK, you can copy this updated PRODOS file to all of your bootable ProDOS disks.

A second approach to the problem is to change the year-lookup routine and track the year separately in your STARTUP program. The command that picks the year out of the table is B9 B8 F1 {LDA ($F1B8),Y} in versions 1.0.1 and 1.0.2 and appears at $5050. In versions 1.1 and 1.1.1 the command is B9 B8 D7 {LDA (D7B8),Y} and is at $4F50. In either case, change it to AD EF 03 {LDA $03EF}.

Now you can keep the month and year in a text file, and have your STARTUP program compare the month currently shown on the clock to that last written to the file. If the current month is less than the month found in the file, you add 1 to the year. In any event, you rewrite the file so it contains only the current month and year. Then you POKE the year into a new "date stamp buffer" we have just created at location 1007 ($3EF). As long as this disk gets booted a couple of times a year it will never get out of date. A word processor can easily make any correction required as a result of extended dormancy.

I used this second method because I had to rewrite the ProDOS clock driver to work with my Mountain Computer AppleClock, which has neither year nor day of week. If you use the Thunderclock, it's easier to just patch up the year table once every five or six years.

Clay Ruth
Dyer, Ind.

Gotcha, Sider, ProDOS

I recently tracked down a bug in a program that was trying to PEEK at the expansion area ($C800-CFFF) firmware on an interface card. Something was mysteriously turning the expansion area off just as I tried to PEEK at it. All interface cards are supposed to turn off their expansion area ROM when the value $CFFF appears on the address bus; some cards do it for any value in the $CF00-CFFF range.

In tracking down the problem I learned that the 6502 has a bug in it that causes the wrong address to momentarily appear on the address bus when an indexed instruction crosses a page boundary. For example, the instruction LDA $CFDC,Y (which appears within Applesoft at $DF3F), with Y holding $C4 (as it does when PEEK is executed), is supposed to load the A register with the value at $D0A0, and it does. However, it also momentarily puts the address $CFA0 ($100 less than the correct value) on the address bus. This makes some peripheral cards, including the one I was trying to PEEK at, turn off their expansion ROM. Gotcha.

The version of DOS 3.3 that comes with the Sider hard drive is only four bytes different from standard DOS 3.3. You can boot off a floppy and enable the Sider with the following patch:

```
10 REM *** Sider enable patch ***

20 REM   Assumes 48K DOS 3.3

30 BD00 = 48384 : SLOT = 7
40 POKE BD00,32
50 POKE BD00+1,17
60 POKE BD00+2,192+SLOT
70 POKE BD00+3,0
```

By the way, the Sider doesn't seem to follow the MSLOT protocol for the expansion area ROM. This lightly documented protocol says that peripheral cards should store their $Cslot value at $7F8 (MSLOT). MSLOT can then be used at the end of an interrupt routine to reset the expansion area if the interrupt has used the expansion ROM on other peripheral cards.

I have also recently spent some time figuring out what ProDOS checks during booting; this may be of interest to others. After checking the $F8 ROM identity bytes to see what machine it's in, testing for the amount of RAM present, and preparing the results for saving later in the machine ID byte ($BF98) in the global page, ProDOS scans the slots for disk devices and other cards.

The scan starts at slot 7 and works downwards. First ProDOS looks for a disk interface card. I'll describe this in more detail later. If a disk interface isn't found, ProDOS next looks for the Thunderclock signature ($08 at Cs00, $28 at Cs02, $58 at Cs04, $70 at Cs06). If a clock is found, the appropriate bit is set in the machine ID byte and the clock routine vector is enabled in the global page.

Failing that, the slot is tested for the Pascal 1.1 interface card identification bytes ($38 at Cs05, $18 at Cs07, $01 at Cs0B). If this much is found, byte $Cs0C, the Pascal 1.1 device signature byte, is checked for an 80-column card. Before ProDOS version 1.1 this byte had to equal $88, which is the ID for Apple's own 80-column "card". In version 1.1 and later, only the high nibble has to equal 8, the low nibble can be anything. Since the low nibble is supposed to identify the manufacturer, the newer versions of ProDOS will accept anyone's 80-column card.

If the three tests for a disk controller card, a clock card, and an 80-column card all fail, the slot is simply checked to see if a ROM is present. If so, the appropriate bit is set to 1 in the global page's SLTBYT ($BF99) flag. After all slots have been checked, a routine is called that does a checksum on the ROM to confirm that a genuine Apple II is in use. If the routine fails it hangs the system.

Now for the interesting part. If the disk interface card search finds the correct signature ($20 at $Cs01, $00 at $Cs03, $03 at $Cs05) in a slot, it then checks the last byte in that card's ROM space, $CsFF. If that byte is zero, the card is assumed to be a standard Apple II floppy controller. If the value is $FF the card is assumed to be an older DOS 3.2 13-sector controller and is rejected. Otherwise the value is saved for later placement in the global page, along with $Cs, as the entry point for access to the device.

Next the byte at $CsFE is checked. This is supposed to be a status byte that defines the characteristics of the device hooked to the disk controller. If the device doesn't support read and status calls (bits 0 and 1 equal 1), the card is rejected. Bit 4 of this byte is used to determine whether the device has two volumes (0 means 1 volume, 1 means 2 volumes). Both are assumed to be there if it's a floppy drive. Bit 5 is not tested. The *ProDOS Technical Reference Manual* says on page 110 that bits 5 and 4 of the $CsFE status byte indicate the number of volumes available on the device. Two bits indicates the possibility of four volumes, but the Apple manual says the possible range is zero to two—how many files does it take to fill zero volumes?

The top nibble of the status byte is shifted down and saved, in the global page's device list at $BF32-3F, as the low nibble of the device ID. The meaning of the device ID then becomes DSSSRIVV, where D is the drive number, SSS the slot number, R whether the device's storage medium is removable (1=yes), I whether the device is interruptable (1=yes), and VV the number of volumes on the device. ProDOS zeros out this lower nibble when using the device ID to call a disk driver.

Drives are added to the global page's device list as they are found and the device count (DEVCNT, $BF31) is incremented. If bit 4 of the controller card's status byte indicates two volumes are available, both are added to the device list. After the scan of all slots has been completed, this list is shuffled so that the boot drive is placed first, followed by the others with the highest slot first. This becomes the order in which drives are searched when a volume isn't found in the default drive.

Tom Vier
Reston, Va.

I've run into that bad-address-on-the-bus-with-indexed-instuctions problem before. **ProntoDOS** *goes crazy if you try to save the firmware on the card in slot 1 with a BSAVE FIRMWARE,A$C100,L$100 command.* **ProntoDOS** *uses an indexed instruction with the base back in page $C0 for this and all the Apple's softswitches get hit as the file is saved. What a mess. The newer 65C02 doesn't have this problem. Nonetheless, people writing firmware for cards that might end up in slot 1 should be careful not to use an indexed instruction with a base in page $C0.*

The information you've developed on the ProDOS initialization routines is very interesting, especially the part about the meaning of the lower nibble of the device ID. Note, however, that while the 4 that the ProFile and the 5 that the Sider put in this nibble make sense, floppies place a 0 there instead of the 9 you would expect, and /RAM shows up as $F instead of a more meaningful 4. The purpose of having two bits in the $CsFE status byte to indicate the number of volumes available rather than one escapes me completely. Since only one of the two bits is actually tested it would appear ProDOS expects only one or two volumes per device. Once a device is called, of course, it could place additional volumes into the device list using phantom slot assignments; but nothing limits this trick to four volumes. Bit 5 looks meaningless to me.

The hang-if-checksum-is-incorrect feature makes ProDOS inoperable on machines with custom Monitor ROMs. You can override this useless, arrogant feature by BLOADing ProDOS at $2000 and changing two bytes to $EA. The bytes to change are: versions 1.0.1 and 1.0.2, $265B-5C; version 1.1, $264D-4E; version 1.1.1, $269E-9F. (In general terms, search for the sequence 69 0B D0 03 and change D0 03 to EA EA. The enhanced IIe Monitor's search command (March, page 20: 0B69<2000.5A00S) works great for this.)