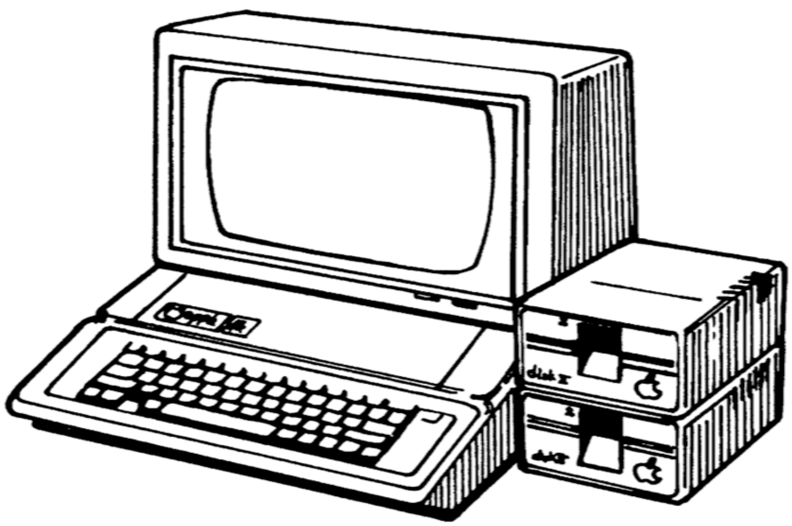
Apple][Computer Information

Apple][
ProDOS Operating System
Technical Information

Apple Assembly Line • Bob Sander-Cederlof • 1983-1988



Apple Assembly Line

SOURCE
http://www.txbobsc.com/aal -- 08 December 2008

TABLE OF CONTENTS

```
################################################################################
#  Commented Listing of ProDOS -- $F800-$F90B, $F996-FEBD
################################################################################
```

November 1983

ProDOS boots its bulk into the RAM card, from $D000 thru $FFFF. More is loaded into the alternate $D000-DFFF space, and all but 255 bytes are reserved out of the entire 16K space.

A system global page is maintained from $BF00-BFFF, for various variables and linkage routines. All communication between machine language programs and ProDOS is supposed to be through MLI (Machine Language Interface) calls and the system global page.

One of the first things I did with ProDOS was to start dis-assembling and commenting it. I want to know what is inside and how it works! Apple's 4-inch thick binder tells a lot, but not all.

Right away I ran into a roadblock: to disassemble out of the RAM card it has to be turned on. There is no monitor in the RAM card when ProDOS is loaded. Turning on the RAM card from the motherboard monitor causes a loud crash!

I overcame most of the problem by copying a monitor into the $F800-FFFF region of the RAM card like this:

```
*C089 C089 F800<F800.FFFFM
*C083 C083
```

The double C089 write-enables the RAM card, while memory reads are still from the motherboard. The rest of the line copies a monitor up. The two C083's get me into the RAM card monitor, ready to type things like "D000LLLLLLLLLLL"

But what about dis-assemblies of the space between $F800 and $FFFF? For this I had to write a little move program. My program turned on the RAM card and copied $F800-FFFF down to $6800-6FFF. Then I BSAVEd it, and later disassembled it.

The code from $F800-FFFF is mostly equivalent to what is in DOS 3.3 from $B800-BFFF. First I found a read/write block subroutine, which calls an RWTS-like subroutine twice per block. (All ProDOS works with 512-byte blocks, rather than sectors; this is like Apple Pascal, and the Apple ///.)

The listing which follows shows the RWB and RWTS subroutines, along with the READ.ADDRESS and READ.SECTOR subroutines. Next month I plan to lay out the SEEK.TRACK and WRITE.SECTOR subroutines, as well as the interrupt and reset handling code.

The outstanding difference between ProDOS and DOS 3.3 disk I/O is speed. ProDOS is considerably faster. Most of the speed increase is due to handling the conversion between memory-bytes and disk-bytes on the fly. DOS pre-converted a 256-byte block into 342 bytes in a special buffer, and then wrote the 342 bytes; ProDOS forms the first 86 bytes of the disk data in a special buffer, writes them, and then proceeds to write the rest of the data directly from the caller's buffer. When reading, DOS read the 342 disk-bytes into a buffer for later decoding into the caller's buffer. ProDOS reads and decodes simultaneously directly into the caller's buffer. This is achieved by extensive use of tables and self-modifying code.

Not only is direct time saved by doing a lot less copying of buffers, but also the sector interleaving can be arranged so that only two revolutions are required to read all 8 blocks on a track.

I believe Apple Pascal uses the same technique, at least for reading.

Whoever coded ProDOS decided to hard-code some parameters which DOS used to keep in tables specified by the user. For example, the number which tells how long to wait for a drive motor to rev up used to be kept in a Device Characteristics Table (DCT). That value is now inside a "LDA #$E8" instruction at $F84F. That means that if you are using a faster drive you have to figure out how to patch and unpatch ProDOS to take advantage of it.

Another hard-coded parameter is the maximum block number. This is no longer part of the data on an initialized disk. It is now locked into the four instructions at $F807-F80D, at a maximum of 279. If you have a 40- or 70-track drive, you can only use 35. Speaking of tracks, the delay tables for track seeking are still used, but they are of course buried in this same almost-unreachable area. If you have a drive with faster track-to-track stepping, the table to change is at $FB73-FB84.

Calls to RWTS in DOS 3.3 involved setting up two tables, an IOB and a DCT. The IOB contained all the data about slot, drive, track, sector, buffer address, etc. The DCT was a 5-byte table with data concerning the drive. ProDOS RWB is called in an entirely different way. A fixed-position table located at $42-47 in page zero is set up with the command, slot, buffer address, and block number.

There are three valid commands, which I call test, read, and write. Test (0) starts up the indicated drive. If it is successful, a normal return occurs; if not, you get an error return (carry set, and (A) non-zero). Read (1) and write (2) are what you expect them to be. RWB has a very simple job: validate the call parameters in $42-47, convert block number to track and sector, and call RWTS twice (once for each sector of the block).

ProDOS RWTS expects the sector number in the A-register, and the track in a variable at $FB56. RWTS handles turning on the drive motor and waiting for it to come up to speed. RWTS then calls SEEK.TRACK to find the desired track, READ.ADDRESS to find the selected sector, and branches to READ.SECTOR or WRITE.SECTOR depending on the command.

READ.ADDRESS is virtually the same in ProDOS as it was in DOS 3.3. READ.SECTOR is entirely different. I should point out here that ProDOS diskettes are entirely compatible with Apple /// diskettes. The file structures are exactly the same. Both ProDOS and Apple /// diskettes use the same basic recording techniques on the disk as DOS 3.3, so the diskettes are copyable using standard DOS 3.3 copiers such as the COPYA program on your old System Master Diskette.

READ.SECTOR begins by computing several addresses and plugging them into the code further down. (This enables the use of faster addressing modes, saving enough cycles to leave time for complete decoding of disk data on the fly.) First the disk slot number is merged into the instructions which read bytes from the drive. Next the caller's buffer address is put into the store instructions.

Note that the byte from the disk is loaded into the X-register, then used to index into BYTE.TABLE, at $F996, to get the equivalent 6-bit data value. Since a disk byte may only have certain values, there is some space within BYTE.TABLE that will never be accessed. Most of this unused space contains $FF bytes, but some of it is used for other small tables: BIT.PAIR.LEFT, .MIDDLE, and .RIGHT, and DATA.TRAILER. These are used by WRITE.SECTOR, which we'll look at next month.

Your buffer is divided into three parts: two 86-byte chunks, and one of 84 bytes. Data coming from the disk is in four chunks: three of 86 bytes, and one of 84.

The first chunk contains the lower two bits from every byte in the original data. READ.SECTOR reads this chunk into TBUF, so that the bits will be available later for merging with the upper six of each byte. ($FC53-FC68)

The second chunk contains the upper six bits from the first 86 bytes of the original data. $FC69-FC83 reads the chunk and merges in the lower two bits from TBUF, storing the completed bytes in the first 85 bytes of the caller's buffer. The last (86th) byte is saved on the stack (I am not sure why), and not stored in the caller's buffer until after all the rest of the data has been read.

A tricky manipulation is necessary to merge in those lower two bits. The data in TBUF has those bits in backward order, packed together with the bits from the other chunks. There was a good diagram of this on page 10 of the June 1981 issue of AAL. DOS merged them with a complex time-consuming shifting process. ProDOS does a swift table lookup, using the TBUF byte as an index to the BIT.PAIR.TABLE.

BIT.PAIR.TABLE has four bytes per row. The first three in each row supply the bit pairs; the fourth is used by SECTOR.WRITE to encode data, and will be covered next month.

When $FC69-FC83 is reading the first chunk, the first byte in each row is used to supply the lower two data bits. The byte in TBUF corresponding to the current position in the chunk selects a byte from BIT.PAIR.TABLE, and the two parts are merged together.

The next two chunks are handled just like the one I just described. After all the data has been read, READ.SECTOR expects to have accumulated a checksum of 00, and expects to find a trailing $EB after the data. Return with carry clear indicates all went well; carry set indicates a read error (bad checksum, missing header, or missing trailer).

I can't help wondering: can this fast read technique be fit into DOS 3.3? It takes a little more code and table space, but on the other hand it uses 256 bytes less of intermediate buffer. If we sacrificed the INIT command, could both the fast read and write be squeezed into DOS 3.3?

For more good information on ProDOS, be sure to take a look at Tom Weishaar's DOStalk column in the current issue of Softalk.

```
1000        .TI 76,PRODOS F800-FFFF.....COMMENTED BY RBS-C  11-8-83............
1010 *SAVE S.PRODOS F800-FFFF
1020 *---------------------------------
1030 RUNNING.SUM        .EQ $3A
1040 TBUF.0             .EQ $3A
1050 BYTE.AT.BUF00      .EQ $3B
1060 BYTE.AT.BUF01      .EQ $3C
1070 LAST.BYTE          .EQ $3D
1080 SLOT.X16           .EQ $3E
1090 INDEX.OF.LAST.BYTE .EQ $3F
1100 *---------------------------------
```

```
1110 RWB.COMMAND .EQ $42
1120 RWB.SLOT    .EQ $43   DSSSXXXX
1130 RWB.BUFFER  .EQ $44,45
1140 RWB.BLOCK   .EQ $46,47   0...279
1150 *--------------------------------
1160 BUFF.BASE .EQ $4700 DUMMY ADDRESS FOR ASSEMBLY ONLY
1170 *--------------------------------
1180 SAVE.LOC45 .EQ $BF56
1190 SAVE.D000  .EQ $BF57
1200 INTAREG    .EQ $BF88
1210 INTBANKID  .EQ $BF8D
1220 IRQXIT.3   .EQ $BFD3
1230 *--------------------------------
1240 DRV.PHASE  .EQ $C080
1250 DRV.MTROFF .EQ $C088
1260 DRV.MTRON  .EQ $C089
1270 DRV.ENBL.0 .EQ $C08A
1280 DRV.Q6L    .EQ $C08C
1290 DRV.Q6H    .EQ $C08D
1300 DRV.Q7L    .EQ $C08E
1310 DRV.Q7H    .EQ $C08F
1320 *--------------------------------
1330 *                 <<<COMPUTED >>>
1340 MODIFIER .EQ $60  <<<SLOT * 16>>>
1350 *--------------------------------
1360        .OR $F800
1370        .TA $800
1380 *--------------------------------
1390 *      READ/WRITE A BLOCK
1400 *
1410 *      1.  ASSURE VALID BLOCK NUMBER (0...279)
1420 *      2.  CONVERT BLOCK NUMBER TO TRACK/SECTOR
1430 *          TRACK = INT(BLOCK/8)
1440 *          BLOCK   SECTORS
1450 *          -----  ---------
1460 *            0     0 AND 2
1470 *            1     4 AND 6
1480 *            2     8 AND 10
1490 *            3    12 AND 14
1500 *            4     1 AND 3
1510 *            5     5 AND 7
1520 *            6     9 AND 11
1530 *            7    13 AND 15
1540 *      3.  CALL RWTS TWICE
1550 *      4.  RETURN WITH ERROR STATUS
1560 *--------------------------------
1570 RWB
1580        LDA RWB.BLOCK     BLOCK MUST BE 0...279
1590        LDX RWB.BLOCK+1
1600        STX RWTS.TRACK
1610        BEQ .1       ...BLOCK # LESS THAN 256
1620        DEX
1630        BNE .5       ...BLOCK # MORE THAN 511
1640        CMP #$18
1650        BCS .5       ...BLOCK # MORE THAN 279
1660 .1     LDY #5       SHIFT 5 BITS OF TRACK #
```

```
1670 .2     ASL               RWTS.TRACK   A-REG
1680        ROL RWTS.TRACK   ----------   --------
1690        DEY                00TTTTTT    ABC00000
1700        BNE .2
1710        ASL          TRANSFORM BLOCK # INTO SECTOR #
1720        BCC .3       ABC00000 --> 0000BC0A
1730        ORA #$10
1740 .3     LSR
1750        LSR
1760        LSR
1770        LSR
1780        PHA
1790        JSR RWTS     R/W FIRST SECTOR OF BLOCK
1800        PLA
1810        BCS .4       ...ERROR
1820        INC RWB.BUFFER+1
1830        ADC #2
1840        JSR RWTS     R/W SECOND SECTOR OF BLOCK
1850        DEC RWB.BUFFER+1
1860 .4     LDA RWTS.ERROR
1870        RTS
1880 *---BLOCK NUMBER > 279-----------
1890 .5     LDA #$27     I/O ERROR
1900        SEC
1910        RTS
1920 *--------------------------------
1930 *      READ/WRITE A GIVEN SECTOR
1940 *--------------------------------
1950 RWTS
1960        LDY #1       TRY SEEKING TWICE
1970        STY SEEK.COUNT
1980        STA RWTS.SECTOR
1990        LDA RWB.SLOT
2000        AND #$70     0SSS0000
2010        STA SLOT.X16
2020        JSR WAIT.FOR.OLD.MOTOR.TO.STOP
2030        JSR CHECK.IF.MOTOR.RUNNING
2040        PHP          SAVE ANSWER (.NE. IF RUNNING)
2050        LDA #$E8     MOTOR STARTING TIME
2060        STA MOTOR.TIME+1  ONLY HI-BYTE NECESSARY
2070        LDA RWB.SLOT      SAME SLOT AND DRIVE?
2080        CMP OLD.SLOT
2090        STA OLD.SLOT
2100        PHP               SAVE ANSWER
2110        ASL               DRIVE # TO C-BIT
2120        LDA DRV.MTRON,X   START MOTOR
2130        BCC .1            ...DRIVE 0
2140        INX               ...DRIVE 1
2150 .1     LDA DRV.ENBL.0,X  ENABLE DRIVE X
2160        PLP               SAME SLOT/DRIVE?
2170        BEQ .3            ...YES
2180        PLP          DISCARD ANSWER ABOUT MOTOR GOING
2190        LDY #7       DELAY 150-175 MILLISECS
2200 .2     JSR DELAY.100     DELAY 25 MILLISECS
2210        DEY
2220        BNE .2
```

```
       PHP           SAY MOTOR NOT ALREADY GOING
.3     LDA RWB.COMMAND   0=TEST, 1=READ, 2=WRITE
       BEQ .4            ...0, MERELY TEST
       LDA RWTS.TRACK
       JSR SEEK.TRACK
.4     PLP           WAS MOTOR ALREADY GOING?
       BNE .6        ...YES
.5     LDA #1        DELAY 100 USECS
       JSR DELAY.100
       LDA MOTOR.TIME+1
       BMI .5        ...WAIT TILL IT OUGHT TO BE
       JSR CHECK.IF.MOTOR.RUNNING
       BEQ .14       ...NOT RUNNING YET, ERROR
.6     LDA RWB.COMMAND
       BEQ .17       CHECK IF WRITE PROTECTED
       LSR           .CS. IF READ, .CC. IF WRITE
       BCS .7        ...READ
       JSR PRE.NYBBLE   ...WRITE
.7     LDY #64       TRY 64 TIMES TO FIND THE SECTOR
       STY SEARCH.COUNT
.8     LDX SLOT.X16
       JSR READ.ADDRESS
       BCC .10       ...FOUND IT
.9     DEC SEARCH.COUNT
       BPL .8        ...KEEP LOOKING
       LDA #$27      I/O ERROR CODE
       DEC SEEK.COUNT   ANY TRIES LEFT?
       BNE .14       ...NO, I/O ERROR
       LDA CURRENT.TRACK
       PHA
       ASL           SLIGHT RE-CALIBRATION
       ADC #$10
       LDY #64       ANOTHER 64 TRIES
       STY SEARCH.COUNT
       BNE .11       ...ALWAYS
.10    LDY HDR.TRACK     ACTUAL TRACK FOUND
       CPY CURRENT.TRACK
       BEQ .12           FOUND THE RIGHT ONE
       LDA CURRENT.TRACK WRONG ONE, TRY AGAIN
       PHA
       TYA           STARTING FROM TRACK FOUND
       ASL
.11    JSR UPDATE.TRACK.TABLE
       PLA
       JSR SEEK.TRACK
       BCC .8        ...ALWAYS
.12    LDA HDR.SECTOR
       CMP RWTS.SECTOR
       BNE .9
       LDA RWB.COMMAND
       LSR
       BCC .15           ...WRITE
       JSR READ.SECTOR   ...READ
       BCS .9            ...READ ERROR
.13    LDA #0        NO ERROR
       .HS D0        "BNE"...NEVER, JUST SKIPS "SEC"
```

```
 2790 .14    SEC          ERROR
 2800        STA RWTS.ERROR    SAVE ERROR CODE
 2810        LDA DRV.MTROFF,X  STOP MOTOR
 2820        RTS               RETURN
 2830 *--------------------------------
 2840 .15    JSR WRITE.SECTOR
 2850 .16    BCC .13      ...NO ERROR
 2860        LDA #$2B     WRITE PROTECTED ERROR CODE
 2870        BNE .14      ...ALWAYS
 2880 .17    LDX SLOT.X16 CHECK IF WRITE PROTECTED
 2890        LDA DRV.Q6H,X
 2900        LDA DRV.Q7L,X
 2910        ROL
 2920        LDA DRV.Q6L,X
 2930        JMP .16      GIVE ERROR IF PROTECTED
[ SEEK.TRACK is in this gap. It will be published next month. ]
[ The following tables start at $F996.                          ]

 3660 *--------------------------------
 3670 *      VALUE READ FROM DISK IS INDEX INTO THIS TABLE
 3680 *      TABLE ENTRY GIVES TOP 6 BITS OF ACTUAL DATA
 3690 *
 3700 *      OTHER DATA TABLES ARE IMBEDDED IN THE UNUSED
 3710 *      PORTIONS OF THE BYTE.TABLE
 3720 *--------------------------------
 3730 BYTE.TABLE .EQ *-$96
 3740        .HS 0004FFFF080CFF101418
 3750 BIT.PAIR.LEFT
 3760        .HS 008040C0
 3770        .HS FFFF1C20FFFFFF24282C
 3780        .HS 3034FFFF383C4044
 3790        .HS 484CFF5054585C606468
 3800 BIT.PAIR.MIDDLE
 3810        .HS 00201030
 3820 DATA.TRAILER
 3830        .HS DEAAEBFF
 3840        .HS FFFFFF6CFF70
 3850        .HS 7478FFFFFF7CFFFF
 3860        .HS 8084FF888C9094989CA0
 3870 BIT.PAIR.RIGHT
 3880        .HS 0008040C
 3890        .HS FFA4A8ACFFB0B4B8BCC0
 3900        .HS C4C8FFFFCCD0D4D8
 3910        .HS DCE0FFE4E8ECF0F4
 3920        .HS F8FC
 3930 *--------------------------------
 3940 BIT.PAIR.TABLE
 3950        .HS 00000096
 3960        .HS 02000097
 3970        .HS 0100009A
 3980        .HS 0300009B
 3990        .HS 0002009D
 4000        .HS 0202009E
 4010        .HS 0102009F
 4020        .HS 030200A6
 4030        .HS 000100A7
```

```
4040        .HS 020100AB
4050        .HS 010100AC
4060        .HS 030100AD
4070        .HS 000300AE
4080        .HS 020300AF
4090        .HS 010300B2
4100        .HS 030300B3
4110        .HS 000002B4
4120        .HS 020002B5
4130        .HS 010002B6
4140        .HS 030002B7
4150        .HS 000202B9
4160        .HS 020202BA
4170        .HS 010202BB
4180        .HS 030202BC
4190        .HS 000102BD
4200        .HS 020102BE
4210        .HS 010102BF
4220        .HS 030102CB
4230        .HS 000302CD
4240        .HS 020302CE
4250        .HS 010302CF
4260        .HS 030302D3
4270        .HS 000001D6
4280        .HS 020001D7
4290        .HS 010001D9
4300        .HS 030001DA
4310        .HS 000201DB
4320        .HS 020201DC
4330        .HS 010201DD
4340        .HS 030201DE
4350        .HS 000101DF
4360        .HS 020101E5
4370        .HS 010101E6
4380        .HS 030101E7
4390        .HS 000301E9
4400        .HS 020301EA
4410        .HS 010301EB
4420        .HS 030301EC
4430        .HS 000003ED
4440        .HS 020003EE
4450        .HS 010003EF
4460        .HS 030003F2
4470        .HS 000203F3
4480        .HS 020203F4
4490        .HS 010203F5
4500        .HS 030203F6
4510        .HS 000103F7
4520        .HS 020103F9
4530        .HS 010103FA
4540        .HS 030103FB
4550        .HS 000303FC
4560        .HS 020303FD
4570        .HS 010303FE
4580        .HS 030303FF
4590 *--------------------------------
```

```
4600 TBUF   .BS 86
4610 *-------------------------------
4620 RWTS.TRACK    .HS 07
4630 RWTS.SECTOR   .HS 0F
4640 RWTS.ERROR    .HS 00
4650 OLD.SLOT      .HS 60
4660 CURRENT.TRACK .HS 07
4670               .HS 00
4680 *-------------------------------
4690 OLD.TRACK.TABLE .EQ *-4
4700        .HS 0000     SLOT 2, DRIVE 0--DRIVE 1
4710        .HS 0000     SLOT 3
4720        .HS 0000     SLOT 4
4730        .HS 0000     SLOT 5
4740        .HS 0E00     SLOT 6
4750        .HS 0000     SLOT 7
4760 *-------------------------------
4770        .HS 00
4780 *-------------------------------
4790 SEARCH.COUNT      .BS 1
4800 SEEK.COUNT        .BS 1
4810 STEP.CNT      .EQ *
4820 SEEK.D5.CNT   .EQ *
4830 X1X1X1X1          .BS 1   ALSO STEP.CNT & SEEK.D5.CNT
4840 CHECK.SUM         .BS 1
4850 HDR.CHKSUM        .BS 1
4860 HDR.SECTOR        .BS 1
4870 HDR.TRACK     .EQ *
4880 MOTOR.TIME        .BS 2   ALSO HDR.TRACK & HDR.VOLUME
4890 CURRENT.TRACK.OLD .BS 1
4900 TARGET.TRACK      .BS 1
4910 *-------------------------------
4920 *      DELAY TIMES FOR ACCELERATION & DECELERATION
4930 *      OF TRACK STEPPING MOTOR
4940 *-------------------------------
4950 ONTBL  .HS 01302824201E1D1C1C
4960 OFFTBL .HS 702C26221F1E1D1C1C
4970 *-------------------------------
4980 *      DELAY ABOUT 100*A MICROSECONDS
4990 *      RUN DOWN MOTOR.TIME WHILE DELAYING
5000 *-------------------------------
5010 DELAY.100
5020 .1     LDX #17
5030 .2     DEX
5040        BNE .2
5050        INC MOTOR.TIME
5060        BNE .3
5070        INC MOTOR.TIME+1
5080 .3     SEC
5090        SBC #1
5100        BNE .1
5110        RTS
5120 *-------------------------------
5130 READ.ADDRESS
5140        LDY #$FC     TRY 772 TIMES TO FIND $D5
5150        STY SEEK.D5.CNT        (FROM $FCFC TO $10000)
```

```
5160 .1     INY
5170        BNE .2       ...KEEP TRYING
5180        INC SEEK.D5.CNT
5190        BEQ .11      ...THAT IS ENUF!
5200 .2     LDA DRV.Q6L,X     GET NEXT BYTE
5210        BPL .2
5220 .3     CMP #$D5     IS IT $D5?
5230        BNE .1       ...NO, TRY AGAIN
5240        NOP          ...YES, DELAY
5250 .4     LDA DRV.Q6L,X     GET NEXT BYTE
5260        BPL .4
5270        CMP #$AA     NOW NEED $AA AND $96
5280        BNE .3       ...NO, BACK TO $D5 SEARCH
5290        LDY #3       (READ 3 BYTES LATER)
5300 .5     LDA DRV.Q6L,X     GET NEXT BYTE
5310        BPL .5
5320        CMP #$96     BETTER BE...
5330        BNE .3       ...IT IS NOT
5340        SEI          ...NO INTERRUPTS NOW
5350        LDA #0       START CHECK SUM
5360 .6     STA CHECK.SUM
5370 .7     LDA DRV.Q6L,X     GET NEXT BYTE
5380        BPL .7       1X1X1X1X
5390        ROL          X1X1X1X1
5400        STA X1X1X1X1
5410 .8     LDA DRV.Q6L,X     GET NEXT BYTE
5420        BPL .8       1Y1Y1Y1Y
5430        AND X1X1X1X1   XYXYXYXY
5440        STA HDR.CHKSUM,Y
5450        EOR CHECK.SUM
5460        DEY
5470        BPL .6
5480        TAY          CHECK CHECKSUM
5490        BNE .11      NON-ZERO, ERROR
5500 .9     LDA DRV.Q6L,X     GET NEXT BYTE
5510        BPL .9
5520        CMP #$DE     TRAILER EXPECTED $DE.AA.EB
5530        BNE .11      NO, ERROR
5540        NOP
5550 .10    LDA DRV.Q6L,X
5560        BPL .10
5570        CMP #$AA
5580        BNE .11      NO, ERROR
5590        CLC
5600        RTS
5610 .11    SEC
5620        RTS
5630 *--------------------------------
5640 READ.SECTOR
5650        TXA          SLOT*16 ($60 FOR SLOT 6)
5660        ORA #$8C     BUILD Q6L ADDRESS FOR SLOT
5670        STA .9+1     STORE INTO READ-DISK OPS
5680        STA .12+1
5690        STA .13+1
5700        STA .15+1
5710        STA .18+1
```

```
5720        LDA RWB.BUFFER    PLUG CALLER'S BUFFER
5730        LDY RWB.BUFFER+1  ADDRESS INTO STORE'S
5740        STA .17+1    PNTR FOR LAST THIRD
5750        STY .17+2
5760        SEC          PNTR FOR MIDDLE THIRD
5770        SBC #84
5780        BCS .1
5790        DEY
5800 .1     STA .14+1
5810        STY .14+2
5820        SEC          PNTR FOR BOTTOM THIRD
5830        SBC #87
5840        BCS .2
5850        DEY
5860 .2     STA .11+1
5870        STY .11+2
5880 *---FIND $D5.AA.AD HEADER--------
5890        LDY #32      MUST FIND $D5 WITHIN 32 BYTES
5900 .3     DEY
5910        BEQ .10      ERROR RETURN
5920 .4     LDA DRV.Q6L,X
5930        BPL .4
5940 .5     EOR #$D5
5950        BNE .3
5960        NOP
5970 .6     LDA DRV.Q6L,X
5980        BPL .6
5990        CMP #$AA
6000        BNE .5
6010        NOP
6020 .7     LDA DRV.Q6L,X
6030        BPL .7
6040        CMP #$AD
6050        BNE .5
6060 *---READ 86 BYTES INTO TBUF...TBUF+85----------
6070 *---THESE ARE THE PACKED LOWER TWO BITS--------
6080 *---FROM EACH BYTE OF THE CALLER'S BUFFER.-----
6090        LDY #170
6100        LDA #0       INIT RUNNING EOR-SUM
6110 .8     STA RUNNING.SUM
6120 .9     LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
6130        BPL .9
6140        LDA BYTE.TABLE,X      DECODE DATA
6150        STA TBUF-170,Y
6160        EOR RUNNING.SUM
6170        INY
6180        BNE .8
6190 *---READ NEXT 86 BYTES-------------------------
6200 *---STORE 1ST 85 IN BUFFER...BUFFER+84---------
6210 *---SAVE THE 86TH BYTE ON THE STACK------------
6220        LDY #170
6230        BNE .12      ...ALWAYS
6240 *--
6250 .10    SEC          I/O ERROR EXIT
6260        RTS
6270 *--
```

```
6280 .11    STA BUFF.BASE-171,Y
6290 .12    LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
6300        BPL .12
6310        EOR BYTE.TABLE,X      DECODE DATA
6320        LDX TBUF-170,Y        MERGE LOWER 2 BITS
6330        EOR BIT.PAIR.TABLE,X
6340        INY
6350        BNE .11
6360        PHA          SAVE LAST BYTE (LATER BUFFER+85)
6370 *---READ NEXT 86 BYTES-----------
6380 *---STORE AT BUFFER+86...BUFFER+171------------
6390        AND #$FC              MASK FOR RUNNING EOR.SUM
6400        LDY #170
6410 .13    LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
6420        BPL .13
6430        EOR BYTE.TABLE,X      DECODE DATA
6440        LDX TBUF-170,Y        MERGE LOWER 2 BITS
6450        EOR BIT.PAIR.TABLE+1,X
6460 .14    STA BUFF.BASE-84,Y
6470        INY
6480        BNE .13
6490 *---READ NEXT 84 BYTES------------------------
6500 *---INTO BUFFER+172...BUFFER+255---------------
6510 .15    LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
6520        BPL .15
6530        AND #$FC
6540        LDY #172
6550 .16    EOR BYTE.TABLE,X      DECODE DATA
6560        LDX TBUF-172,Y        MERGE LOWER 2 BITS
6570        EOR BIT.PAIR.TABLE+2,X
6580 .17    STA BUFF.BASE,Y
6590 .18    LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
6600        BPL .18
6610        INY
6620        BNE .16
6630        AND #$FC
6640 *---END OF DATA------------------
6650        EOR BYTE.TABLE,X      DECODE DATA
6660        BNE .20           ...BAD CHECKSUM
6670        LDX SLOT.X16      CHECK FOR TRAILER $DE
6680 .19    LDA DRV.Q6L,X
6690        BPL .19
6700        CMP #$DE
6710        CLC
6720        BEQ .21           ...GOOD READ!
6730 .20    SEC               ...SIGNAL BAD READ
6740 .21    PLA               STORE BYTE AT BUFFER+85
6750        LDY #85
6760        STA (RWB.BUFFER),Y
6770        RTS
6780 *--------------------------------
6790 UPDATE.TRACK.TABLE
6800        JSR GET.SSSD.IN.X
6810        STA OLD.TRACK.TABLE,X
6820        RTS
6830 *--------------------------------
```

```
6840 CHECK.IF.MOTOR.RUNNING
6850        LDX SLOT.X16
6860 CHECK.IF.MOTOR.RUNNING.X
6870        LDY #0
6880 .1     LDA DRV.Q6L,X     READ CURRENT INPUT REGISTER
6890        JSR .2            ...12 CYCLES...
6900        PHA               ...7 MORE CYCLES...
6910        PLA
6920        CMP DRV.Q6L,X     BY NOW INPUT REGISTER
6930        BNE .2            SHOULD HAVE CHANGED
6940        LDA #$28     ERROR CODE: NO DEVICE CONNECTED
6950        DEY          BUT TRY 255 MORE TIMES
6960        BNE .1       ...RETURN .NE. IF MOVING...
6970 .2     RTS          ...RETURN .EQ. IF NOT MOVING...
6980 *--------------------------------
6990 GET.SSSD.IN.X
7000        PHA          SAVE A-REG
7010        LDA RWB.SLOT     DSSSXXXX
7020        LSR
7030        LSR
7040        LSR
7050        LSR          0000DSSS
7060        CMP #8       SET CARRY IF DRIVE 2
7070        AND #7       00000SSS
7080        ROL          0000SSSD
7090        TAX          INTO X-REG
7100        PLA          RESTORE A-REG
7110        RTS
7120 *--------------------------------
7130 WRITE.SECTOR
7140        SEC          IN CASE WRITE-PROTECTED
7150        LDA DRV.Q6H,X
7160        LDA DRV.Q7L,X
7170        BPL .1       ...NOT WRITE PROTECTED
7180        JMP WS.RET   ...PROTECTED, ERROR
7190 *--------------------------------
7200 .1     LDA TBUF
7210        STA TBUF.0
7220 *---WRITE 5 SYNC BYTES-----------
7230        LDA #$FF
7240        STA DRV.Q7H,X
7250        ORA DRV.Q6L,X
7260        LDY #4
7270        NOP          $FF AT 40-CYCLE INTERVALS LEAVES
7280        PHA          TWO ZERO-BITS AFTER EACH $FF
7290        PLA
7300 .2     PHA
7310        PLA
7320        JSR WRITE2
7330        DEY
7340        BNE .2
7350 *---WRITE $D5 AA AD HEADER-------
7360        LDA #$D5
7370        JSR WRITE1
7380        LDA #$AA
7390        JSR WRITE1
```

```
7400        LDA #$AD
7410        JSR WRITE1
7420 *---WRITE 86 BYTES FROM TBUF-------------------
7430 *---BACKWARDS:   TBUF+85...TBUF+1, TBUF.0------
7440        TYA          =0
7450        LDY #86
7460        BNE .4
7470 .3     LDA TBUF,Y
7480 .4     EOR TBUF-1,Y
7490        TAX
7500        LDA BIT.PAIR.TABLE+3,X
7510        LDX SLOT.X16
7520        STA DRV.Q6H,X
7530        LDA DRV.Q6L,X
7540        DEY
7550        BNE .3
7560        LDA TBUF.0
7570 *---WRITE PORTION OF BUFFER------
7580 *---UP TO A PAGE BOUNDARY--------
7590        LDY #*-*     FILLED IN WITH LO-BYTE OF BUFFER ADDRESS
7600 WS...5 EOR BUFF.BASE,Y   HI-BYTE FILLED IN
7610        AND #$FC
7620        TAX
7630        LDA BIT.PAIR.TABLE+3,X
7640 WS...6 LDX #MODIFIER
7650        STA DRV.Q6H,X
7660        LDA DRV.Q6L,X
7670 WS...7 LDA BUFF.BASE,Y   HI-BYTE FILLED IN
7680        INY
7690        BNE WS...5
7700 *---BRANCH ACCORDING TO BUFFER BOUNDARY CONDITIONS-----
7710        LDA BYTE.AT.BUF00
7720        BEQ WS..17   ...BUFFER ALL IN ONE PAGE
7730        LDA INDEX.OF.LAST.BYTE
7740        BEQ WS..16   ...ONLY ONE BYTE IN NEXT PAGE
7750 *---MORE THAN ONE BYTE IN NEXT PAGE--------------------
7760        LSR          ...DELAY TWO CYCLES
7770        LDA BYTE.AT.BUF00  PRE.NYBBLE ALREADY ENCODED
7780        STA DRV.Q6H,X      THIS BYTE
7790        LDA DRV.Q6L,X
7800        LDA BYTE.AT.BUF01
7810        NOP
7820        INY
7830        BCS WS..12
7840 WS...8 EOR BUFF.BASE+256,Y    HI-BYTE FILLED IN
7850        AND #$FC
7860        TAX
7870        LDA BIT.PAIR.TABLE+3,X
7880 WS...9 LDX #MODIFIER
7890        STA DRV.Q6H,X
7900        LDA DRV.Q6L,X
7910 WS..10 LDA BUFF.BASE+256,Y    HI-BYTE FILLED IN
7920        INY
7930 WS..11 EOR BUFF.BASE+256,Y    HI-BYTE FILLED IN
7940 WS..12 CPY INDEX.OF.LAST.BYTE
7950        AND #$FC
```

```
7960        TAX
7970        LDA BIT.PAIR.TABLE+3,X
7980 WS..13 LDX #MODIFIER
7990        STA DRV.Q6H,X
8000        LDA DRV.Q6L,X
8010 WS..14 LDA BUFF.BASE+256,Y    HI-BYTE FILLED IN
8020        INY
8030        BCC WS...8
8040        BCS .15       ...3 CYCLE NOP
8050 .15    BCS WS..17    ...ALWAYS
8060 *---WRITE BYTE AT BUFFER.00--------------------------
8070 WS..16 .DA #$AD,BYTE.AT.BUF00    4 CYCLES: LDA BYTE.AT.BUF00
8080        STA DRV.Q6H,X
8090        LDA DRV.Q6L,X
8100        PHA
8110        PLA
8120        PHA
8130        PLA
8140 WS..17 LDX LAST.BYTE
8150        LDA BIT.PAIR.TABLE+3,X
8160 WS..18 LDX #MODIFIER
8170        STA DRV.Q6H,X
8180        LDA DRV.Q6L,X
8190        LDY #0
8200        PHA
8210        PLA
8220 *---WRITE DATA TRAILER:  $DE AA EB FF----------
8230        NOP
8240        NOP
8250 .19    LDA DATA.TRAILER,Y
8260        JSR WRITE3
8270        INY
8280        CPY #4
8290        BNE .19
8300        CLC           SIGNAL NO ERROR
8310 WS.RET LDA DRV.Q7L,X     DRIVE TO SAFE MODE
8320        LDA DRV.Q6L,X
8330        RTS
8340 *-------------------------------
8350 WRITE1 CLC
8360 WRITE2 PHA
8370        PLA
8380 WRITE3 STA DRV.Q6H,X
8390        ORA DRV.Q6L,X
8400        RTS
8410 *-------------------------------
8420 PRE.NYBBLE
8430        LDA RWB.BUFFER    PLUG IN ADDRESS TO LOOP BELOW
8440        LDY RWB.BUFFER+1
8450        CLC
8460        ADC #2
8470        BCC .1
8480        INY
8490 .1     STA PN...6+1
8500        STY PN...6+2
8510        SEC
```

```
8520        SBC #$56
8530        BCS .2
8540        DEY
8550 .2     STA PN...5+1
8560        STY PN...5+2
8570        SEC
8580        SBC #$56
8590        BCS .3
8600        DEY
8610 .3     STA PN...4+1
8620        STY PN...4+2
8630 *---PACK THE LOWER TWO BITS INTO TBUF-------------
8640        LDY #170
8650 PN...4 LDA BUFF.BASE-170,Y   ADDRESS FILLED IN
8660        AND #3
8670        TAX
8680        LDA BIT.PAIR.RIGHT,X
8690        PHA
8700 PN...5 LDA BUFF.BASE-84,Y
8710        AND #3
8720        TAX
8730        PLA
8740        ORA BIT.PAIR.MIDDLE,X
8750        PHA
8760 PN...6 LDA BUFF.BASE+2,Y
8770        AND #3
8780        TAX
8790        PLA
8800        ORA BIT.PAIR.LEFT,X
8810        PHA
8820        TYA
8830        EOR #$FF
8840        TAX
8850        PLA
8860        STA TBUF,X
8870        INY
8880        BNE PN...4
8890 *---DETERMINE BUFFER BOUNDARY CONDITIONS----------
8900 *---AND SETUP WRITE.SECTOR ACCORDINGLY------------
8910        LDY RWB.BUFFER
8920        DEY
8930        STY INDEX.OF.LAST.BYTE
8940        LDA RWB.BUFFER
8950        STA WS...5-1
8960        BEQ .7
8970        EOR #$FF
8980        TAY
8990        LDA (RWB.BUFFER),Y
9000        INY
9010        EOR (RWB.BUFFER),Y
9020        AND #$FC
9030        TAX
9040        LDA BIT.PAIR.TABLE+3,X
9050 .7     STA BYTE.AT.BUF00    =0 IF BUFFER NOT SPLIT
9060        BEQ .9
9070        LDA INDEX.OF.LAST.BYTE
```

```
9080        LSR
9090        LDA (RWB.BUFFER),Y
9100        BCC .8
9110        INY
9120        EOR (RWB.BUFFER),Y
9130 .8     STA BYTE.AT.BUF01
9140 .9     LDY #$FF
9150        LDA (RWB.BUFFER),Y
9160        AND #$FC
9170        STA LAST.BYTE
9180 *---INSTALL BUFFER ADDRESSES IN WRITE.SECTOR------
9190        LDY RWB.BUFFER+1
9200        STY WS...5+2
9210        STY WS...7+2
9220        INY
9230        STY WS...8+2
9240        STY WS..10+2
9250        STY WS..11+2
9260        STY WS..14+2
9270 *---INSTALL SLOT*16 IN WRITE.SECTOR---------------
9280        LDX SLOT.X16
9290        STX WS...6+1
9300        STX WS...9+1
9310        STX WS..13+1
9320        STX WS..18+1
9330        RTS
9340 *--------------------------------
9350 WAIT.FOR.OLD.MOTOR.TO.STOP
9360        EOR OLD.SLOT      SAME SLOT AS BEFORE?
9370        ASL               (IGNORE DRIVE)
9380        BEQ .2            ...YES
9390        LDA #1       LONG MOTOR.TIME
9400        STA MOTOR.TIME+1  (COUNTS BACKWARDS)
9410 .1     LDA OLD.SLOT
9420        AND #$70
9430        TAX
9440        BEQ .2       ...NO PREVIOUS MOTOR RUNNING
9450        JSR CHECK.IF.MOTOR.RUNNING.X
9460        BEQ .2       ...NOT RUNNING YET
9470        LDA #1       DELAY ANOTHER 100 USECS
9480        JSR DELAY.100
9490        LDA MOTOR.TIME+1
9500        BNE .1       KEEP WAITING
9510 .2     RTS
9520 *--------------------------------
```

```
################################################################################
#  ProDOS and Clock Drivers, with a Commented Listing of ProDOS $F142-$F1BE
################################################################################
```

Bob Sander-Cederlof

November 1983

ProDOS is a new operating system which Apple expects to release to the public during the first quarter of 1984. I am told that new computers and disk drives will be shipped with ProDOS rather than DOS 3.3. Version 1.0 is already available to licensed developers (I have it).

Apple has released massive amounts of documentation to licensed developers, and has even been offering a full day class at $225 per seat in various cities around the country. I attended the Dallas class on October 21st. Even with all the help they are giving, there are still a lot of unclear details that can only be illuminated by well-commented assembly listings of the actual ProDOS code. Apple will never publish these, so we will do it ourselves.

My first serious foray into ProDOS began at the request of Dan Pote, Applied Engineering. Dan wanted me to modify the firmware of his Timemaster clock card so that it automatically had full compatibility with ProDOS. Dan wanted all programs, even protected ones, which boot under ProDOS, to be able to read the date and time from his card. Also, he wanted ProDOS to time/date stamp the files in the directory with his card, just as it does with Thunderclock. (No small task, it turned out.)

ProDOS, when booting, searches the slots for a Thunderclock. If it finds one, it marks a bit in the machine ID byte (MACHID, bit 0 of $BF98 = 1); it plugs two bytes at $F14D and F150 with $CN, where N is the slot number; and it stores a JMP opcode ($4C) at $BF06.

$BF06 is a standard vector to whatever clock routine is installed. If no Thunderclock was found, an RTS opcode will be stored there.

The ProDOS boot slot search looks for these Thunderclock ID bytes:

```
       $CN00 = $08
       $CN02 = $28
       $CN04 = $58
       $CN08 = $70
```

After booting, ProDOS loads and executes the program called STARTUP. The standard STARTUP program searches the slots for various cards and displays a list of what it finds. Unfortunately this list seldom agrees with the true configuration in any of my computers. For one thing, STARTUP examines different bytes than the boot search does. In looking for a clock card, STARTUP wants:

```
       $CN00 = $08
       $CN01 = $78
       $CN02 = $28
```

If you do not have a Thunderclock, but do have some other clock, you have several options. What I did for Dan was change the firmware of Timemaster so that it emulates Thunderclock. ProDOS is convinced it has a Thunderclock, but you are saved the extra expense, and you gain extra features.

Another approach is to write a program which installs your own clock driver inside ProDOS. Mike Owen, of Austin, Texas, did this for Dan. After ProDOS boots it loads the first type SYS file it can find in the directory whose name ends with ".SYSTEM". Normally this is "BASIC.SYSTEM", which then proceeds to execute STARTUP. However, you can set up your disk with CLOCK.SYSTEM before BASIC.SYSTEM in the directory.

Write CLOCK.SYSTEM so that it begins at $2000, because all type SYS files load there. The program should mark the clock ID bit in MACHID, punch a JMP opcode at $BF06, and look at the address in $BF07,BF08. That address is the beginning of the clock driver inside the language card. Right now that address is $F142, but it could change.

Your program should write enable the language card by two "LDA $C081" instructions in a row, and then copy your clock driver into the space starting at that address. You can use up to 124 bytes. If your driver has references to the clock slot, be sure to modify them to the actual slot you are using. If your driver has internal references, be sure to modify them to point to the actual addresses inside the new physical location.

It is standard practice in peripheral firmware to use the following code to find out which slot the card is in:

```
       JSR $FF58     A Guaranteed $60 (RTS opcode)
       TSX           Stack pointer
       LDA $100,X    Get $CN off stack
```

Many cards also use "BIT $FF58" as a means for setting the V-bit in the status register. BE AWARE THAT ProDOS DOES NOT HAVE $60 AT $FF58 in the language card!!!!

The Thunderclock has two entries, at $CN08 and $CN0B, which assume that $CN is already in the X-register. $CN0B allows setting the clock mode, and $CN08 reads the clock in the current mode. The ProDOS driver calls on these two entries, as the following listing shows.

ProDOS maintains a full page at $BF00 called the System Global Page. The definition of this page should not change, ever. They say. Locations $BF90-BF93 contain the current date and time in a packed format. A system call will read the clock, if a driver is installed, and format the year-month-day-hour-minute into these four bytes.

Now here is a listing of the current Thunderclock driver, as labelled and commented by me.

```
  1000 *SAVE S.PRODOS $F142...$F1BE
  1010 *---------------------------------
  1020 *  IF THE PRODOS BOOT RECOGNIZES A THUNDERCLOCK,
  1030 *  A "JMP $F142" IS INSTALLED AT $BF06 AND
  1040 *  THE SLOT ADDRESS IS PATCHED INTO THE FOLLOWING
  1050 *  CODE AT SLOT.A AND SLOT.B BELOW.
  1060 *---------------------------------
  1070 DATE   .EQ $BF90    $BF91 = YYYYYYYM
  1080 *                   $BF90 = MMMDDDDD
  1090 TIME   .EQ $BF92    $BF93 = 000HHHHH
  1100 *                   $BF92 = 00MMMMMM
  1110 MODE   .EQ $5F8-$C0 THUNDERCLOCK MODE IN SCREEN HOLE
  1120 *---------------------------------
  1130        .OR $F142
```

```
1140        .TA $800
1150 *--------------------------------
1160 PRODOS.THUNDERCLOCK.DRIVER
1170        LDX SLOT.B   $CN
1180        LDA MODE,X   SAVE CURRENT THUNDERCLOCK MODE
1190        PHA
1200        LDA #$A3     SEND "#" TO THUNDERCLOCK TO
1210        JSR $C20B       SELECT INTEGER MODE
1220 SLOT.A .EQ *-1
1230 *--------------------------------
1240 *      READ TIME & DATE INTO $200...$211 IN FORMAT:
1250 *--------------------------------
1260        JSR $C208
1270 SLOT.B .EQ *-1
1280 *--------------------------------
1290 *      CONVERT ASCII VALUES TO BINARY
1300 *      $3E -- MINUTE
1310 *      $3D -- HOUR
1320 *      $3C -- DAY OF MONTH
1330 *      $3B -- DAY OF WEEK
1340 *      $3A -- MONTH
1350 *--------------------------------
1360        CLC
1370        LDX #4
1380        LDY #12      POINT AT MINUTE
1390 .1     LDA $200,Y   TEN'S DIGIT
1400        AND #$07     IGNORE TOP BIT
1410        STA $3A      MULTIPLY DIGIT BY TEN
1420        ASL          *2
1430        ASL          *4
1440        ADC $3A      *5
1450        ASL          *10
1460        ADC $201,Y   ADD UNIT'S DIGIT
1470        SEC
1480        SBC #$B0     SUBTRACT ASCII ZERO
1490        STA $3A,X    STORE VALUE
1500        DEY          BACK UP TO PREVIOUS FIELD
1510        DEY
1520        DEY
1530        DEX          BACK UP TO PREVIOUS VALUE
1540        BPL .1       ...UNTIL ALL 5 FIELDS CONVERTED
1550 *--------------------------------
1560 *      PACK MONTH AND DAY OF MONTH,
1570 *--------------------------------
1580        TAY          MONTH (1...12)
1590        LSR          00000ABC--D
1600        ROR          D00000AB--C
1610        ROR          CD00000A--B
1620        ROR          BCD00000--A
1630        ORA $3C      MERGE DAY OF MONTH
1640        STA DATE     SAVE PACKED DAY AND MONTH
1650        PHP          SAVE TOP BIT OF MONTH
1660 *--------------------------------
1670 *      CONVERT MONTH, DAY OF MONTH,
1680 *      AND DAY OF WEEK INTO YEAR.
1690 *--------------------------------
```

```
1700        AND #$1F      ISOLATE DAY OF MONTH (1...31)
1710 *      CARRY SET FOR MONTHS 8...12
1720        ADC YEAR.DAY,Y    COMPUTE DAY OF YEAR
1730        BCC .2
1740        ADC #3    ADJUST REMAINDER FOR YEARDAY > 255
1750 .2     SEC           GET REMAINDER MODULO 7
1760 .3     SBC #7
1770        BCS .3        ...UNTIL ALL 7'S REMOVED
1780        ADC #7        RESTORE TO POSITIVE VALUE
1790        SBC $3B       SUBTRACT KNOWN DAY OF WEEK
1800        BCS .4        NO BORROW
1810        ADC #7        BORROWED, SO ADD 7 BACK
1820 .4     TAY           ADJUSTED DAY OW WEEK AS INDEX
1830        LDA YRTBL,Y  GET YEAR (82...87)
1840        PLP           GET HIGH BIT OF MONTH IN CARRY
1850        ROL           FORM YYYYYYYM
1860        STA DATE+1
1870        LDA $3D       GET HOUR
1880        STA TIME+1
1890        LDA $3E       GET MINUTE
1900        STA TIME
1910        PLA           RESTORE THUNDERCLOCK MODE
1920        LDX SLOT.B    GET $CN FOR INDEX
1930        STA MODE,X
1940        RTS
1950 *---------------------------------
1960 YEAR.DAY   .EQ *-1  OFFSET BECAUSE INDEX 1...12
1970    .DA #0,#31,#59,#90      JAN,FEB,MAR,APR
1980    .DA #120,#151,#181,#211 MAY,JUN,JUL,AUG
1990    .DA #242,#20,#51,#81    SEP,OCT,NOV,DEC
2000 *---------------------------------
2010 YRTBL  .DA #84,#84,#83,#82,#87,#86,#85
2020 *---------------------------------
```

```
################################################################################
#  Commented Listing of ProDOS -- $F90C-F995, $FD00-FE9A, $FEBE-FFFF
################################################################################
```

December 1983

Last month I printed the commented listing of the disk reading subroutines. This month's selection covers disk writing, track positioning, and interrupt handling. Together the two articles cover all the code between $F800 and $FFFF.

Several callers have wondered if this is all there is to ProDOS. No! It is only a small piece. In my opinion, this is the place to start in understanding ProDOS's features: A faster way of getting information to and from standard floppies. But remember that ProDOS also supports the ProFILE hard disk, and a RAM disk in the extended Apple //e memory.

Further, ProDOS has a file structure exactly like Apple /// SOS, with a hierarchical directory and file sizes up to 16 megabytes.

Further, ProDOS includes support for a clock/calendar card, 80-columns with Smarterm or //e, and interrupts.

ProDOS uses or reserves all but 255 bytes of the 16384 bytes in the language card area (both $D000-DFFF banks and all #E000-FFFF). The 255 bytes not reserved are from $D001 through $D0FF in one of the $D000 banks. The byte at $D000 is reserved, because ProDOS uses it to distinguish which $D000 bank is switched on when an interrupt occurs. The space at $BF00-BFFF is used by ProDOS for system linkages and variables (called the System Global Page).

In addition, if you are using Applesoft, ProDOS uses memory from $9600-BEFF. This space does not include any file buffers. When you OPEN files, buffers are allocated as needed. CLOSEing automatically de-allocates buffers. Each buffer is 1024 bytes long. As you can see, with ProDOS in place your Applesoft program has less room than ever.

Track Seeking: $F90C-F995

The SEEK.TRACK subroutine begins at $F90C. The very first instruction multiplies the track number by two, converting ProDOS logical track number to a physical track number. If you want to access a "half-track" position, you could either store a NOP opcode at $F90C, or enter the subroutine at $F90D.

A table is maintained of the current track position for each of up to 12 drives. I call it the OLD.TRACK.TABLE. The subroutine GET.SSSD.IN.X forms an index into OLD.TRACK.TABLE from slot# * 2 + drive#. There are no entries in the table for drives in slots 0 or 1, which is fine with me. ProDOS uses these slots as pseudo slots for the RAM-based pseudo-disk and for ProFILE, if I remember correctly.

The code in SEEK.TRACK.ABSOLUTE is similar but not identical to code in DOS 3.3. The differences do not seem to be significant.

Disk Writing: $FD00-FE9A

The overall process of writing a sector is handled by code in RWTS, which was listed last month. After the desired track is found, RWTS calls PRE.NYBBLE to build a block of 86 bytes containing the low-order two bits from each byte in the caller's buffer.

PRE.NYBBLE also stores a number of buffer addresses and slot*16 values inside the WRITE.SECTOR subroutine. Next RWTS calls READ.ADDRESS to find the sector, and then WRITE.SECTOR to put the data out.

WRITE.SECTOR is the real workhorse. And it is very critically timed. Once the write head in your drive is enabled, every machine cycle is closely counted until the last byte is written. First, five sync bytes are written (ten bits each, 1111111100). These are written by putting $FF in the write register at 40 cycle intervals. Following the sync bytes W.S writes a data header of D5 AA AD.

Second, the 86-byte block which PRE.NYBBLE built is written, followed by the coded form of the rest of your buffer. WRITE.SECTOR picks up bytes directly from your buffer, keeps a running checksum, encodes the high-order six bits into an 8-bit value, and writes it on the disk...one byte every 32 cycles, exactly. Since your buffer can be any arbitrary place in memory, and since the 6502 adds cycles for indexed instructions that cross page boundaries, WRITE.SECTOR splits the buffer in parts before and after a page boundary. All the overhead for the split is handled in PRE.NYBBLE, before the timed operations begin.

Finally, the checksum and a data trailer of DE AA EB FF are written.

Empty Space: $FEBE-FF9A

This space had no code in it. Nearly a whole page here.

Interrupt & RESET Handling: $FF9B-FFFF

If the RAM card is switched on when an interrupt or RESET occurs, the vectors at $FFFA-FFFF will be those ProDOS installed rather than the ones permanently coded in ROM. It turns out the non-maskable interrupt (NMI) is still vectored down into page 3. But the more interesting IRQ interrupt is now vectored to code at $FF9B inside ProDOS.

The ProDOS IRQ handler performs two functions beyond those built-in to the monitor ROM. First, the contents of location $45 are saved so that the monitor can safely clobber it. Second, a flag is set indicating which $D000 bank is currently switched on, so that it can be restored after the interrupt handler is finished. (The second step is omitted if the interrupt was caused by a BRK opcode.)

If the IRQ was not due to a BRK opcode, a fake "RTI" vector is pushed on the stack. This consists of a return address of $BF50 and a status of $04. The status keeps IRQ interrupts disabled, and $BF50 is a short routine which turns the ProDOS memory back on and jumps up to INT.SPLICE at $FFD8:

```
       BF50-  8D 8B C0  STA $C08B
       BF53-  4C D8 FF  JMP $FFD8
```

Of course, before coming back via the RTI, ProDOS tries to USE the interrupt. If you have set up one or more interrupt vectors with the ProDOS system call, they will be called.

INT.SPLICE restores the contents of $45 and switches the main $D000 bank on. Then it jumps back to $BFD3 with the information about which $D000 bank really should be on. $BFD3 turns on the other bank if necessary, and returns to the point at which the interrupt occured.

The instruction at $FFC8 is interesting. STA $C082 turns on the monitor ROM, so the next instruction to be executed is at $FFCB in ROM. This is an RTS opcode, so the address on the stack at that point is used. There are two possible values: $FA41 if an IRQ interrupt is being processed, or $FA61 if a RESET is being processed. This means the RTS will effectively branch to $FA42 or $FA62.

Uh Oh! At this point you had better hope that you are not running with the original Apple monitor ROM. The Apple II Plus ROM (called Autostart Monitor) and the Apple //e ROM are fine. $FA42 is the second instruction of the IRQ code, and $FA62 is the standard RESET handler. But the original ROM, like I have in my serial 219 machine, has entirely different code there.

I have an $FF at $FA42, followed by code for the monitor S (single step) command. And $FA62 is right in the middle of the S command. There is no telling what might happen, short of actually trying it out. No thanks. Just remember that RESET, BRK, and IRQ interrupts will not work correctly if they happen when the RAM area is switched on and you have the old original monitor in ROM.

There is another small empty space from $FFE9 through $FFF9, 17 bytes.

Perhaps I should point out that the listings this month and last are from the latest release of ProDOS, which may not be the final released version. However, I would expect any differences in the regions I have covered so far to be slight.

[In this web edition I have included the entire code, instead of just the pieces not printed in the November 1983 issue.]

```
1000        .TI 76,PRODOS F800-FFFF.....COMMENTED BY RBS-C  11-8-83...........
1010 *SAVE S.PRODOS F800-FFFF
1020 *--------------------------------
1030 RUNNING.SUM        .EQ $3A
1040 TBUF.0             .EQ $3A
1050 BYTE.AT.BUF00      .EQ $3B
1060 BYTE.AT.BUF01      .EQ $3C
1070 LAST.BYTE          .EQ $3D
1080 SLOT.X16           .EQ $3E
1090 INDEX.OF.LAST.BYTE .EQ $3F
1100 *--------------------------------
1110 RWB.COMMAND .EQ $42
1120 RWB.SLOT    .EQ $43   DSSSXXXX
1130 RWB.BUFFER  .EQ $44,45
1140 RWB.BLOCK   .EQ $46,47   0...279
1150 *--------------------------------
1160 BUFF.BASE .EQ $4700 DUMMY ADDRESS FOR ASSEMBLY ONLY
1170 *--------------------------------
1180 SAVE.LOC45 .EQ $BF56
1190 SAVE.D000  .EQ $BF57
1200 INTAREG    .EQ $BF88
1210 INTBANKID  .EQ $BF8D
1220 IRQXIT.3   .EQ $BFD3
1230 *--------------------------------
1240 DRV.PHASE  .EQ $C080
1250 DRV.MTROFF .EQ $C088
1260 DRV.MTRON  .EQ $C089
1270 DRV.ENBL.0 .EQ $C08A
1280 DRV.Q6L    .EQ $C08C
```

```
1290 DRV.Q6H    .EQ $C08D
1300 DRV.Q7L    .EQ $C08E
1310 DRV.Q7H    .EQ $C08F
1320 *--------------------------------
1330 *                 <<<COMPUTED >>>
1340 MODIFIER .EQ $60  <<<SLOT * 16>>>
1350 *--------------------------------
1360        .OR $F800
1370        .TA $800
1380 *--------------------------------
1390 *      READ/WRITE A BLOCK
1400 *
1410 *      1.  ASSURE VALID BLOCK NUMBER (0...279)
1420 *      2.  CONVERT BLOCK NUMBER TO TRACK/SECTOR
1430 *          TRACK = INT(BLOCK/8)
1440 *          BLOCK   SECTORS
1450 *          -----  ---------
1460 *            0     0 AND 2
1470 *            1     4 AND 6
1480 *            2     8 AND 10
1490 *            3    12 AND 14
1500 *            4     1 AND 3
1510 *            5     5 AND 7
1520 *            6     9 AND 11
1530 *            7    13 AND 15
1540 *      3.  CALL RWTS TWICE
1550 *      4.  RETURN WITH ERROR STATUS
1560 *--------------------------------
1570 RWB
1580        LDA RWB.BLOCK     BLOCK MUST BE 0...279
1590        LDX RWB.BLOCK+1
1600        STX RWTS.TRACK
1610        BEQ .1       ...BLOCK # LESS THAN 256
1620        DEX
1630        BNE .5       ...BLOCK # MORE THAN 511
1640        CMP #$18
1650        BCS .5       ...BLOCK # MORE THAN 279
1660 .1     LDY #5       SHIFT 5 BITS OF TRACK #
1670 .2     ASL              RWTS.TRACK   A-REG
1680        ROL RWTS.TRACK   ----------   --------
1690        DEY               00TTTTTT    ABC00000
1700        BNE .2
1710        ASL          TRANSFORM BLOCK # INTO SECTOR #
1720        BCC .3       ABC00000 --> 0000BC0A
1730        ORA #$10
1740 .3     LSR
1750        LSR
1760        LSR
1770        LSR
1780        PHA
1790        JSR RWTS     R/W FIRST SECTOR OF BLOCK
1800        PLA
1810        BCS .4       ...ERROR
1820        INC RWB.BUFFER+1
1830        ADC #2
1840        JSR RWTS     R/W SECOND SECTOR OF BLOCK
```

```
       DEC RWB.BUFFER+1
.4     LDA RWTS.ERROR
       RTS
*---BLOCK NUMBER > 279-----------
.5     LDA #$27     I/O ERROR
       SEC
       RTS
*--------------------------------
*      READ/WRITE A GIVEN SECTOR
*--------------------------------
RWTS
       LDY #1       TRY SEEKING TWICE
       STY SEEK.COUNT
       STA RWTS.SECTOR
       LDA RWB.SLOT
       AND #$70     0SSS0000
       STA SLOT.X16
       JSR WAIT.FOR.OLD.MOTOR.TO.STOP
       JSR CHECK.IF.MOTOR.RUNNING
       PHP          SAVE ANSWER (.NE. IF RUNNING)
       LDA #$E8     MOTOR STARTING TIME
       STA MOTOR.TIME+1  ONLY HI-BYTE NECESSARY
       LDA RWB.SLOT      SAME SLOT AND DRIVE?
       CMP OLD.SLOT
       STA OLD.SLOT
       PHP               SAVE ANSWER
       ASL               DRIVE # TO C-BIT
       LDA DRV.MTRON,X   START MOTOR
       BCC .1            ...DRIVE 0
       INX               ...DRIVE 1
.1     LDA DRV.ENBL.0,X  ENABLE DRIVE X
       PLP               SAME SLOT/DRIVE?
       BEQ .3            ...YES
       PLP          DISCARD ANSWER ABOUT MOTOR GOING
       LDY #7       DELAY 150-175 MILLISECS
.2     JSR DELAY.100     DELAY 25 MILLISECS
       DEY
       BNE .2
       PHP          SAY MOTOR NOT ALREADY GOING
.3     LDA RWB.COMMAND   0=TEST, 1=READ, 2=WRITE
       BEQ .4            ...0, MERELY TEST
       LDA RWTS.TRACK
       JSR SEEK.TRACK
.4     PLP          WAS MOTOR ALREADY GOING?
       BNE .6       ...YES
.5     LDA #1       DELAY 100 USECS
       JSR DELAY.100
       LDA MOTOR.TIME+1
       BMI .5       ...WAIT TILL IT OUGHT TO BE
       JSR CHECK.IF.MOTOR.RUNNING
       BEQ .14      ...NOT RUNNING YET, ERROR
.6     LDA RWB.COMMAND
       BEQ .17      CHECK IF WRITE PROTECTED
       LSR          .CS. IF READ, .CC. IF WRITE
       BCS .7       ...READ
       JSR PRE.NYBBLE   ...WRITE
```

```
2410 .7     LDY #64      TRY 64 TIMES TO FIND THE SECTOR
2420        STY SEARCH.COUNT
2430 .8     LDX SLOT.X16
2440        JSR READ.ADDRESS
2450        BCC .10      ...FOUND IT
2460 .9     DEC SEARCH.COUNT
2470        BPL .8       ...KEEP LOOKING
2480        LDA #$27     I/O ERROR CODE
2490        DEC SEEK.COUNT   ANY TRIES LEFT?
2500        BNE .14      ...NO, I/O ERROR
2510        LDA CURRENT.TRACK
2520        PHA
2530        ASL          SLIGHT RE-CALIBRATION
2540        ADC #$10
2550        LDY #64      ANOTHER 64 TRIES
2560        STY SEARCH.COUNT
2570        BNE .11      ...ALWAYS
2580 .10    LDY HDR.TRACK     ACTUAL TRACK FOUND
2590        CPY CURRENT.TRACK
2600        BEQ .12           FOUND THE RIGHT ONE
2610        LDA CURRENT.TRACK WRONG ONE, TRY AGAIN
2620        PHA
2630        TYA          STARTING FROM TRACK FOUND
2640        ASL
2650 .11    JSR UPDATE.TRACK.TABLE
2660        PLA
2670        JSR SEEK.TRACK
2680        BCC .8       ...ALWAYS
2690 .12    LDA HDR.SECTOR
2700        CMP RWTS.SECTOR
2710        BNE .9
2720        LDA RWB.COMMAND
2730        LSR
2740        BCC .15           ...WRITE
2750        JSR READ.SECTOR   ...READ
2760        BCS .9            ...READ ERROR
2770 .13    LDA #0       NO ERROR
2780        .HS D0       "BNE"...NEVER, JUST SKIPS "SEC"
2790 .14    SEC          ERROR
2800        STA RWTS.ERROR    SAVE ERROR CODE
2810        LDA DRV.MTROFF,X  STOP MOTOR
2820        RTS               RETURN
2830 *-------------------------------
2840 .15    JSR WRITE.SECTOR
2850 .16    BCC .13      ...NO ERROR
2860        LDA #$2B     WRITE PROTECTED ERROR CODE
2870        BNE .14      ...ALWAYS
2880 .17    LDX SLOT.X16 CHECK IF WRITE PROTECTED
2890        LDA DRV.Q6H,X
2900        LDA DRV.Q7L,X
2910        ROL
2920        LDA DRV.Q6L,X
2930        JMP .16      GIVE ERROR IF PROTECTED
2940 *-------------------------------
2950 SEEK.TRACK
2960        ASL          GET PHYSICAL TRACK #
```

```
2970        STA HDR.TRACK     ...SAVE HERE
2980        JSR CLEAR.PHASES  (CARRY WAS CLEAR)
2990        JSR GET.SSSD.IN.X
3000        LDA OLD.TRACK.TABLE,X
3010        STA CURRENT.TRACK
3020        LDA HDR.TRACK
3030        STA OLD.TRACK.TABLE,X
3040        JSR SEEK.TRACK.ABSOLUTE
3050 *--------------------------------
3060 CLEAR.PHASES
3070        LDY #3
3080 .1     TYA
3090        JSR PHASE.COMMANDER
3100        DEY
3110        BPL .1
3120        LSR CURRENT.TRACK    BACK TO LOGICAL TRACK #
3130        CLC          SIGNAL NO ERROR
3140        RTS
3150 *--------------------------------
3160 SEEK.TRACK.ABSOLUTE
3170        STA TARGET.TRACK  SAVE ACTUAL TRACK #
3180        CMP CURRENT.TRACK ALREADY THERE?
3190        BEQ .7            ...YES
3200        LDA #0
3210        STA STEP.CNT      # STEPS SO FAR
3220 .1     LDA CURRENT.TRACK
3230        STA CURRENT.TRACK.OLD
3240        SEC
3250        SBC TARGET.TRACK
3260        BEQ .6            ...WE HAVE ARRIVED
3270        BCS .2            CURRENT > DESIRED
3280        EOR #$FF          CURRENT < DESIRED
3290        INC CURRENT.TRACK
3300        BCC .3            ...ALWAYS
3310 .2     ADC #$FE          .CS., SO A=A-1
3320        DEC CURRENT.TRACK
3330 .3     CMP STEP.CNT GET MINIMUM OF:
3340        BCC .4         1. # OF TRACKS TO MOVE LESS 1
3350        LDA STEP.CNT   2. # OF STEPS SO FAR
3360 .4     CMP #9         3. EIGHT
3370        BCS .5
3380        TAY
3390        SEC          TURN NEW PHASE ON
3400 .5     JSR .7
3410        LDA ONTBL,Y  DELAY
3420        JSR DELAY.100
3430        LDA CURRENT.TRACK.OLD
3440        CLC          TURN OLD PHASE OFF
3450        JSR PHASE.COMMANDER
3460        LDA OFFTBL,Y DELAY
3470        JSR DELAY.100
3480        INC STEP.CNT # OF STEPS SO FAR
3490        BNE .1       ...ALWAYS
3500 .6     JSR DELAY.100
3510        CLC          TURN PHASE OFF
3520 .7     LDA CURRENT.TRACK
```

```
3530 *--------------------------------
3540 *       (A) = TRACK #
3550 *       .CC. THEN PHASE OFF
3560 *       .CS. THEN PHASE ON
3570 *--------------------------------
3580 PHASE.COMMANDER
3590        AND #3       ONLY KEEP LOWER TWO BITS
3600        ROL               00000XXC
3610        ORA SLOT.X16      0SSS0XXC
3620        TAX
3630        LDA DRV.PHASE,X
3640        LDX SLOT.X16      RESTORE SLOT*16
3650        RTS
3660 *--------------------------------
3670 *       VALUE READ FROM DISK IS INDEX INTO THIS TABLE
3680 *       TABLE ENTRY GIVES TOP 6 BITS OF ACTUAL DATA
3690 *
3700 *       OTHER DATA TABLES ARE IMBEDDED IN THE UNUSED
3710 *       PORTIONS OF THE BYTE.TABLE
3720 *--------------------------------
3730 BYTE.TABLE .EQ *-$96
3740        .HS 0004FFFF080CFF101418
3750 BIT.PAIR.LEFT
3760        .HS 008040C0
3770        .HS FFFF1C20FFFFFF24282C
3780        .HS 3034FFFF383C4044
3790        .HS 484CFF5054585C606468
3800 BIT.PAIR.MIDDLE
3810        .HS 00201030
3820 DATA.TRAILER
3830        .HS DEAAEBFF
3840        .HS FFFFFF6CFF70
3850        .HS 7478FFFFFF7CFFFF
3860        .HS 8084FF888C9094989CA0
3870 BIT.PAIR.RIGHT
3880        .HS 0008040C
3890        .HS FFA4A8ACFFB0B4B8BCC0
3900        .HS C4C8FFFFCCD0D4D8
3910        .HS DCE0FFE4E8ECF0F4
3920        .HS F8FC
3930 *--------------------------------
3940 BIT.PAIR.TABLE
3950        .HS 00000096
3960        .HS 02000097
3970        .HS 0100009A
3980        .HS 0300009B
3990        .HS 0002009D
4000        .HS 0202009E
4010        .HS 0102009F
4020        .HS 030200A6
4030        .HS 000100A7
4040        .HS 020100AB
4050        .HS 010100AC
4060        .HS 030100AD
4070        .HS 000300AE
4080        .HS 020300AF
```

```
4090        .HS 010300B2
4100        .HS 030300B3
4110        .HS 000002B4
4120        .HS 020002B5
4130        .HS 010002B6
4140        .HS 030002B7
4150        .HS 000202B9
4160        .HS 020202BA
4170        .HS 010202BB
4180        .HS 030202BC
4190        .HS 000102BD
4200        .HS 020102BE
4210        .HS 010102BF
4220        .HS 030102CB
4230        .HS 000302CD
4240        .HS 020302CE
4250        .HS 010302CF
4260        .HS 030302D3
4270        .HS 000001D6
4280        .HS 020001D7
4290        .HS 010001D9
4300        .HS 030001DA
4310        .HS 000201DB
4320        .HS 020201DC
4330        .HS 010201DD
4340        .HS 030201DE
4350        .HS 000101DF
4360        .HS 020101E5
4370        .HS 010101E6
4380        .HS 030101E7
4390        .HS 000301E9
4400        .HS 020301EA
4410        .HS 010301EB
4420        .HS 030301EC
4430        .HS 000003ED
4440        .HS 020003EE
4450        .HS 010003EF
4460        .HS 030003F2
4470        .HS 000203F3
4480        .HS 020203F4
4490        .HS 010203F5
4500        .HS 030203F6
4510        .HS 000103F7
4520        .HS 020103F9
4530        .HS 010103FA
4540        .HS 030103FB
4550        .HS 000303FC
4560        .HS 020303FD
4570        .HS 010303FE
4580        .HS 030303FF
4590 *--------------------------------
4600 TBUF   .BS 86
4610 *--------------------------------
4620 RWTS.TRACK    .HS 07
4630 RWTS.SECTOR   .HS 0F
4640 RWTS.ERROR    .HS 00
```

```
4650 OLD.SLOT      .HS 60
4660 CURRENT.TRACK .HS 07
4670               .HS 00
4680 *--------------------------------
4690 OLD.TRACK.TABLE .EQ *-4
4700        .HS 0000     SLOT 2, DRIVE 0--DRIVE 1
4710        .HS 0000     SLOT 3
4720        .HS 0000     SLOT 4
4730        .HS 0000     SLOT 5
4740        .HS 0E00     SLOT 6
4750        .HS 0000     SLOT 7
4760 *--------------------------------
4770        .HS 00
4780 *--------------------------------
4790 SEARCH.COUNT      .BS 1
4800 SEEK.COUNT        .BS 1
4810 STEP.CNT      .EQ *
4820 SEEK.D5.CNT   .EQ *
4830 X1X1X1X1          .BS 1   ALSO STEP.CNT & SEEK.D5.CNT
4840 CHECK.SUM         .BS 1
4850 HDR.CHKSUM        .BS 1
4860 HDR.SECTOR        .BS 1
4870 HDR.TRACK     .EQ *
4880 MOTOR.TIME        .BS 2   ALSO HDR.TRACK & HDR.VOLUME
4890 CURRENT.TRACK.OLD .BS 1
4900 TARGET.TRACK      .BS 1
4910 *--------------------------------
4920 *      DELAY TIMES FOR ACCELERATION & DECELERATION
4930 *      OF TRACK STEPPING MOTOR
4940 *--------------------------------
4950 ONTBL  .HS 01302824201E1D1C1C
4960 OFFTBL .HS 702C26221F1E1D1C1C
4970 *--------------------------------
4980 *      DELAY ABOUT 100*A MICROSECONDS
4990 *      RUN DOWN MOTOR.TIME WHILE DELAYING
5000 *--------------------------------
5010 DELAY.100
5020 .1     LDX #17
5030 .2     DEX
5040        BNE .2
5050        INC MOTOR.TIME
5060        BNE .3
5070        INC MOTOR.TIME+1
5080 .3     SEC
5090        SBC #1
5100        BNE .1
5110        RTS
5120 *--------------------------------
5130 READ.ADDRESS
5140        LDY #$FC     TRY 772 TIMES TO FIND $D5
5150        STY SEEK.D5.CNT        (FROM $FCFC TO $10000)
5160 .1     INY
5170        BNE .2       ...KEEP TRYING
5180        INC SEEK.D5.CNT
5190        BEQ .11      ...THAT IS ENUF!
5200 .2     LDA DRV.Q6L,X     GET NEXT BYTE
```

```
               BPL .2
        .3     CMP #$D5      IS IT $D5?
               BNE .1        ...NO, TRY AGAIN
               NOP           ...YES, DELAY
        .4     LDA DRV.Q6L,X      GET NEXT BYTE
               BPL .4
               CMP #$AA      NOW NEED $AA AND $96
               BNE .3        ...NO, BACK TO $D5 SEARCH
               LDY #3        (READ 3 BYTES LATER)
        .5     LDA DRV.Q6L,X      GET NEXT BYTE
               BPL .5
               CMP #$96      BETTER BE...
               BNE .3        ...IT IS NOT
               SEI           ...NO INTERRUPTS NOW
               LDA #0        START CHECK SUM
        .6     STA CHECK.SUM
        .7     LDA DRV.Q6L,X      GET NEXT BYTE
               BPL .7        1X1X1X1X
               ROL           X1X1X1X1
               STA X1X1X1X1
        .8     LDA DRV.Q6L,X      GET NEXT BYTE
               BPL .8        1Y1Y1Y1Y
               AND X1X1X1X1    XYXYXYXY
               STA HDR.CHKSUM,Y
               EOR CHECK.SUM
               DEY
               BPL .6
               TAY           CHECK CHECKSUM
               BNE .11       NON-ZERO, ERROR
        .9     LDA DRV.Q6L,X      GET NEXT BYTE
               BPL .9
               CMP #$DE      TRAILER EXPECTED $DE.AA.EB
               BNE .11       NO, ERROR
               NOP
        .10    LDA DRV.Q6L,X
               BPL .10
               CMP #$AA
               BNE .11       NO, ERROR
               CLC
               RTS
        .11    SEC
               RTS
        *--------------------------------
        READ.SECTOR
               TXA           SLOT*16 ($60 FOR SLOT 6)
               ORA #$8C      BUILD Q6L ADDRESS FOR SLOT
               STA .9+1      STORE INTO READ-DISK OPS
               STA .12+1
               STA .13+1
               STA .15+1
               STA .18+1
               LDA RWB.BUFFER     PLUG CALLER'S BUFFER
               LDY RWB.BUFFER+1   ADDRESS INTO STORE'S
               STA .17+1     PNTR FOR LAST THIRD
               STY .17+2
               SEC           PNTR FOR MIDDLE THIRD
```

```
5770        SBC #84
5780        BCS .1
5790        DEY
5800 .1     STA .14+1
5810        STY .14+2
5820        SEC          PNTR FOR BOTTOM THIRD
5830        SBC #87
5840        BCS .2
5850        DEY
5860 .2     STA .11+1
5870        STY .11+2
5880 *---FIND $D5.AA.AD HEADER--------
5890        LDY #32      MUST FIND $D5 WITHIN 32 BYTES
5900 .3     DEY
5910        BEQ .10      ERROR RETURN
5920 .4     LDA DRV.Q6L,X
5930        BPL .4
5940 .5     EOR #$D5
5950        BNE .3
5960        NOP
5970 .6     LDA DRV.Q6L,X
5980        BPL .6
5990        CMP #$AA
6000        BNE .5
6010        NOP
6020 .7     LDA DRV.Q6L,X
6030        BPL .7
6040        CMP #$AD
6050        BNE .5
6060 *---READ 86 BYTES INTO TBUF...TBUF+85----------
6070 *---THESE ARE THE PACKED LOWER TWO BITS--------
6080 *---FROM EACH BYTE OF THE CALLER'S BUFFER.-----
6090        LDY #170
6100        LDA #0       INIT RUNNING EOR-SUM
6110 .8     STA RUNNING.SUM
6120 .9     LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
6130        BPL .9
6140        LDA BYTE.TABLE,X      DECODE DATA
6150        STA TBUF-170,Y
6160        EOR RUNNING.SUM
6170        INY
6180        BNE .8
6190 *---READ NEXT 86 BYTES------------------------
6200 *---STORE 1ST 85 IN BUFFER...BUFFER+84---------
6210 *---SAVE THE 86TH BYTE ON THE STACK------------
6220        LDY #170
6230        BNE .12      ...ALWAYS
6240 *--
6250 .10    SEC          I/O ERROR EXIT
6260        RTS
6270 *--
6280 .11    STA BUFF.BASE-171,Y
6290 .12    LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
6300        BPL .12
6310        EOR BYTE.TABLE,X      DECODE DATA
6320        LDX TBUF-170,Y        MERGE LOWER 2 BITS
```

```
6330        EOR BIT.PAIR.TABLE,X
6340        INY
6350        BNE .11
6360        PHA          SAVE LAST BYTE (LATER BUFFER+85)
6370 *---READ NEXT 86 BYTES-----------
6380 *---STORE AT BUFFER+86...BUFFER+171------------
6390        AND #$FC              MASK FOR RUNNING EOR.SUM
6400        LDY #170
6410 .13    LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
6420        BPL .13
6430        EOR BYTE.TABLE,X      DECODE DATA
6440        LDX TBUF-170,Y        MERGE LOWER 2 BITS
6450        EOR BIT.PAIR.TABLE+1,X
6460 .14    STA BUFF.BASE-84,Y
6470        INY
6480        BNE .13
6490 *---READ NEXT 84 BYTES------------------------
6500 *---INTO BUFFER+172...BUFFER+255---------------
6510 .15    LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
6520        BPL .15
6530        AND #$FC
6540        LDY #172
6550 .16    EOR BYTE.TABLE,X      DECODE DATA
6560        LDX TBUF-172,Y        MERGE LOWER 2 BITS
6570        EOR BIT.PAIR.TABLE+2,X
6580 .17    STA BUFF.BASE,Y
6590 .18    LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
6600        BPL .18
6610        INY
6620        BNE .16
6630        AND #$FC
6640 *---END OF DATA------------------
6650        EOR BYTE.TABLE,X      DECODE DATA
6660        BNE .20             ...BAD CHECKSUM
6670        LDX SLOT.X16        CHECK FOR TRAILER $DE
6680 .19    LDA DRV.Q6L,X
6690        BPL .19
6700        CMP #$DE
6710        CLC
6720        BEQ .21             ...GOOD READ!
6730 .20    SEC                 ...SIGNAL BAD READ
6740 .21    PLA                 STORE BYTE AT BUFFER+85
6750        LDY #85
6760        STA (RWB.BUFFER),Y
6770        RTS
6780 *--------------------------------
6790 UPDATE.TRACK.TABLE
6800        JSR GET.SSSD.IN.X
6810        STA OLD.TRACK.TABLE,X
6820        RTS
6830 *--------------------------------
6840 CHECK.IF.MOTOR.RUNNING
6850        LDX SLOT.X16
6860 CHECK.IF.MOTOR.RUNNING.X
6870        LDY #0
6880 .1     LDA DRV.Q6L,X     READ CURRENT INPUT REGISTER
```

```
6890        JSR .2            ...12 CYCLES...
6900        PHA               ...7 MORE CYCLES...
6910        PLA
6920        CMP DRV.Q6L,X     BY NOW INPUT REGISTER
6930        BNE .2            SHOULD HAVE CHANGED
6940        LDA #$28     ERROR CODE: NO DEVICE CONNECTED
6950        DEY          BUT TRY 255 MORE TIMES
6960        BNE .1       ...RETURN .NE. IF MOVING...
6970 .2     RTS          ...RETURN .EQ. IF NOT MOVING...
6980 *--------------------------------
6990 GET.SSSD.IN.X
7000        PHA          SAVE A-REG
7010        LDA RWB.SLOT     DSSSXXXX
7020        LSR
7030        LSR
7040        LSR
7050        LSR          0000DSSS
7060        CMP #8       SET CARRY IF DRIVE 2
7070        AND #7       00000SSS
7080        ROL          0000SSSD
7090        TAX          INTO X-REG
7100        PLA          RESTORE A-REG
7110        RTS
7120 *--------------------------------
7130 WRITE.SECTOR
7140        SEC          IN CASE WRITE-PROTECTED
7150        LDA DRV.Q6H,X
7160        LDA DRV.Q7L,X
7170        BPL .1       ...NOT WRITE PROTECTED
7180        JMP WS.RET   ...PROTECTED, ERROR
7190 *--------------------------------
7200 .1     LDA TBUF
7210        STA TBUF.0
7220 *---WRITE 5 SYNC BYTES-----------
7230        LDA #$FF
7240        STA DRV.Q7H,X
7250        ORA DRV.Q6L,X
7260        LDY #4
7270        NOP          $FF AT 40-CYCLE INTERVALS LEAVES
7280        PHA          TWO ZERO-BITS AFTER EACH $FF
7290        PLA
7300 .2     PHA
7310        PLA
7320        JSR WRITE2
7330        DEY
7340        BNE .2
7350 *---WRITE $D5 AA AD HEADER-------
7360        LDA #$D5
7370        JSR WRITE1
7380        LDA #$AA
7390        JSR WRITE1
7400        LDA #$AD
7410        JSR WRITE1
7420 *---WRITE 86 BYTES FROM TBUF-------------------
7430 *---BACKWARDS:   TBUF+85...TBUF+1, TBUF.0------
7440        TYA          =0
```

```
7450        LDY #86
7460        BNE .4
7470 .3     LDA TBUF,Y
7480 .4     EOR TBUF-1,Y
7490        TAX
7500        LDA BIT.PAIR.TABLE+3,X
7510        LDX SLOT.X16
7520        STA DRV.Q6H,X
7530        LDA DRV.Q6L,X
7540        DEY
7550        BNE .3
7560        LDA TBUF.0
7570 *---WRITE PORTION OF BUFFER------
7580 *---UP TO A PAGE BOUNDARY--------
7590        LDY #*-*     FILLED IN WITH LO-BYTE OF BUFFER ADDRESS
7600 WS...5 EOR BUFF.BASE,Y   HI-BYTE FILLED IN
7610        AND #$FC
7620        TAX
7630        LDA BIT.PAIR.TABLE+3,X
7640 WS...6 LDX #MODIFIER
7650        STA DRV.Q6H,X
7660        LDA DRV.Q6L,X
7670 WS...7 LDA BUFF.BASE,Y   HI-BYTE FILLED IN
7680        INY
7690        BNE WS...5
7700 *---BRANCH ACCORDING TO BUFFER BOUNDARY CONDITIONS-----
7710        LDA BYTE.AT.BUF00
7720        BEQ WS..17   ...BUFFER ALL IN ONE PAGE
7730        LDA INDEX.OF.LAST.BYTE
7740        BEQ WS..16   ...ONLY ONE BYTE IN NEXT PAGE
7750 *---MORE THAN ONE BYTE IN NEXT PAGE--------------------
7760        LSR          ...DELAY TWO CYCLES
7770        LDA BYTE.AT.BUF00  PRE.NYBBLE ALREADY ENCODED
7780        STA DRV.Q6H,X      THIS BYTE
7790        LDA DRV.Q6L,X
7800        LDA BYTE.AT.BUF01
7810        NOP
7820        INY
7830        BCS WS..12
7840 WS...8 EOR BUFF.BASE+256,Y    HI-BYTE FILLED IN
7850        AND #$FC
7860        TAX
7870        LDA BIT.PAIR.TABLE+3,X
7880 WS...9 LDX #MODIFIER
7890        STA DRV.Q6H,X
7900        LDA DRV.Q6L,X
7910 WS..10 LDA BUFF.BASE+256,Y    HI-BYTE FILLED IN
7920        INY
7930 WS..11 EOR BUFF.BASE+256,Y    HI-BYTE FILLED IN
7940 WS..12 CPY INDEX.OF.LAST.BYTE
7950        AND #$FC
7960        TAX
7970        LDA BIT.PAIR.TABLE+3,X
7980 WS..13 LDX #MODIFIER
7990        STA DRV.Q6H,X
8000        LDA DRV.Q6L,X
```

```
WS..14 LDA BUFF.BASE+256,Y    HI-BYTE FILLED IN
       INY
       BCC WS...8
       BCS .15       ...3 CYCLE NOP
.15    BCS WS..17    ...ALWAYS
*---WRITE BYTE AT BUFFER.00--------------------------
WS..16 .DA #$AD,BYTE.AT.BUF00    4 CYCLES: LDA BYTE.AT.BUF00
       STA DRV.Q6H,X
       LDA DRV.Q6L,X
       PHA
       PLA
       PHA
       PLA
WS..17 LDX LAST.BYTE
       LDA BIT.PAIR.TABLE+3,X
WS..18 LDX #MODIFIER
       STA DRV.Q6H,X
       LDA DRV.Q6L,X
       LDY #0
       PHA
       PLA
*---WRITE DATA TRAILER:  $DE AA EB FF----------
       NOP
       NOP
.19    LDA DATA.TRAILER,Y
       JSR WRITE3
       INY
       CPY #4
       BNE .19
       CLC           SIGNAL NO ERROR
WS.RET LDA DRV.Q7L,X     DRIVE TO SAFE MODE
       LDA DRV.Q6L,X
       RTS
*--------------------------------
WRITE1 CLC
WRITE2 PHA
       PLA
WRITE3 STA DRV.Q6H,X
       ORA DRV.Q6L,X
       RTS
*--------------------------------
PRE.NYBBLE
       LDA RWB.BUFFER    PLUG IN ADDRESS TO LOOP BELOW
       LDY RWB.BUFFER+1
       CLC
       ADC #2
       BCC .1
       INY
.1     STA PN...6+1
       STY PN...6+2
       SEC
       SBC #$56
       BCS .2
       DEY
.2     STA PN...5+1
       STY PN...5+2
```

```
        SEC
        SBC #$56
        BCS .3
        DEY
.3      STA PN...4+1
        STY PN...4+2
*---PACK THE LOWER TWO BITS INTO TBUF-------------
        LDY #170
PN...4 LDA BUFF.BASE-170,Y   ADDRESS FILLED IN
        AND #3
        TAX
        LDA BIT.PAIR.RIGHT,X
        PHA
PN...5 LDA BUFF.BASE-84,Y
        AND #3
        TAX
        PLA
        ORA BIT.PAIR.MIDDLE,X
        PHA
PN...6 LDA BUFF.BASE+2,Y
        AND #3
        TAX
        PLA
        ORA BIT.PAIR.LEFT,X
        PHA
        TYA
        EOR #$FF
        TAX
        PLA
        STA TBUF,X
        INY
        BNE PN...4
*---DETERMINE BUFFER BOUNDARY CONDITIONS----------
*---AND SETUP WRITE.SECTOR ACCORDINGLY------------
        LDY RWB.BUFFER
        DEY
        STY INDEX.OF.LAST.BYTE
        LDA RWB.BUFFER
        STA WS...5-1
        BEQ .7
        EOR #$FF
        TAY
        LDA (RWB.BUFFER),Y
        INY
        EOR (RWB.BUFFER),Y
        AND #$FC
        TAX
        LDA BIT.PAIR.TABLE+3,X
.7      STA BYTE.AT.BUF00    =0 IF BUFFER NOT SPLIT
        BEQ .9
        LDA INDEX.OF.LAST.BYTE
        LSR
        LDA (RWB.BUFFER),Y
        BCC .8
        INY
        EOR (RWB.BUFFER),Y
```

```
9130 .8     STA BYTE.AT.BUF01
9140 .9     LDY #$FF
9150        LDA (RWB.BUFFER),Y
9160        AND #$FC
9170        STA LAST.BYTE
9180 *---INSTALL BUFFER ADDRESSES IN WRITE.SECTOR------
9190        LDY RWB.BUFFER+1
9200        STY WS...5+2
9210        STY WS...7+2
9220        INY
9230        STY WS...8+2
9240        STY WS..10+2
9250        STY WS..11+2
9260        STY WS..14+2
9270 *---INSTALL SLOT*16 IN WRITE.SECTOR---------------
9280        LDX SLOT.X16
9290        STX WS...6+1
9300        STX WS...9+1
9310        STX WS..13+1
9320        STX WS..18+1
9330        RTS
9340 *--------------------------------
9350 WAIT.FOR.OLD.MOTOR.TO.STOP
9360        EOR OLD.SLOT      SAME SLOT AS BEFORE?
9370        ASL               (IGNORE DRIVE)
9380        BEQ .2            ...YES
9390        LDA #1       LONG MOTOR.TIME
9400        STA MOTOR.TIME+1  (COUNTS BACKWARDS)
9410 .1     LDA OLD.SLOT
9420        AND #$70
9430        TAX
9440        BEQ .2       ...NO PREVIOUS MOTOR RUNNING
9450        JSR CHECK.IF.MOTOR.RUNNING.X
9460        BEQ .2       ...NOT RUNNING YET
9470        LDA #1       DELAY ANOTHER 100 USECS
9480        JSR DELAY.100
9490        LDA MOTOR.TIME+1
9500        BNE .1       KEEP WAITING
9510 .2     RTS
9520 *--------------------------------
9530        .BS $FF9B-*       <<<<EMPTY SPACE>>>>
9540 *--------------------------------
9550 IRQ
9560        PHA          SAVE A-REG
9570        LDA $45     SAVE LOC $45
9580        STA SAVE.LOC45
9590        PLA          SAVE A-REG AT LOC $45
9600        STA $45
9610        PLA          GET STATUS BEFORE IRQ
9620        PHA
9630        AND #$10     SEE IF "BRK"
9640        BNE .2       ...YES, LET MONITOR DO IT
9650        LDA $D000    SAVE $D000 BANK ID
9660        EOR #$D8
9670        BEQ .1
9680        LDA #$FF
```

```
9690 .1     STA INTBANKID
9700        STA SAVE.D000
9710        LDA #$BF     PUSH FAKE "RTI" VECTOR WITH
9720        PHA               IRQ DISABLED
9730        LDA #$50          AND SET TO RETURN TO $BF50
9740        PHA
9750        LDA #4
9760        PHA
9770 .2     LDA #$FA     PUSH "RTS" VECTOR FOR MONITOR
9780        PHA
9790        LDA #$41
9800        PHA
9810 CALL.MONITOR
9820        STA $C082    SWITCH TO MOTHERBOARD
9830 *--------------------------------
9840 RESET
9850        LDA RESET.VECTOR+1
9860        PHA          PUSH "RTS" VECTOR FOR MONITOR
9870        LDA RESET.VECTOR
9880        PHA
9890        JMP CALL.MONITOR
9900 *--------------------------------
9910 RESET.VECTOR
9920        .DA $FA61    MON.RESET-1
9930 *--------------------------------
9940 INT.SPLICE
9950        STA INTAREG
9960        LDA SAVE.LOC45
9970        STA $45
9980        LDA $C08B    SWITCH TO MAIN $D000 BANK
9990        LDA SAVE.D000
10000        JMP IRQXIT.3
10010 *--------------------------------
10020        .BS $FFFA-*       <<<<<EMPTY SPACE>>>>>
10030 *--------------------------------
10040 V.NMI      .DA $03FB
10050 V.RESET    .DA RESET
10060 V.IRQ      .DA IRQ
10070 *--------------------------------
```

```
################################################################################
#  Will ProDOS Work on a Franklin?
################################################################################
```

Bob Stout

March 1984

If you try to boot up ProDOS on a Franklin, it probably will fail. The ProDOS boot routine checks to see if you are in a genuine Apple monitor ROM. However, you can make it work.

Start the boot procedure; when meaningful action appears to have ceased, press the RESET switch. Get into the monitor and type 2647:EA EA and 2000G. Voila!

##
Will ProDOS Really Fly?
##

Bob Sander-Cederlof

March 1984

ProDOS appears to have been eclipsed by MacIntosh. The major software houses are probably putting their main effort into Mac.

ARTSCI has announced a ProDOS version of their MagiCalc spreadsheet program. Owners of the DOS 3.3 version may upgrade for $40, new customers pay $149.95. The only differences claimed are faster disk I/O and ability to edit the printer setup string. Nice, but $40 is a lot. And the spreadsheet files would no longer be accessible to DOS-based utilities.

ARTSCI will also send you their ProDOS catalog sorter program, complete with BASIC.SYSTEM, CONVERT, FILER, and the ProDOS image for only $24.95. Apple will reputedly be selling ProDOS with a user's manual and some tutorial files in addition to the files on ARTSCI's disk, but price and date are still unclear. (You get them free with a new disk drive.)

Practical Peripherals has announced a new clock card which is ProDOS compatible. Their design is apparently an upgrade of Superclock II (by Jeff Mazur, Westside Electronics). ProDOS was designed around Thunderclock, so other clocks must either emulate one of the Thunderclock modes or patch ProDOS during the boot process. Applied Engineering's new Timemaster II emulates Thunderclock and several others, so it is fully ProDOS compatible.

According to Don Lancaster, Applewriter //e has been written so that changing to ProDOS would be easy. Therefore we might expect a ProDOS-based version of this popular word processor to be announced soon. Or maybe they won't bother to announce it.

Meanwhile, I know of at least two people with plans to integrate the faster RWTS ProDOS uses into their enhanced DOS 3.3 packages. Have you seen the latest ads for David-DOS? Dave Weston compares the speeds of his fast DOS with DOS 3.3 and Pro-DOS. Guess what ... Pro-DOS doesn't win.

Unless you MUST have file compatibility with Apple /// SOS; or you MUST have hard hard-disk support for very large files; or you MUST have a hierarchical file directory; then stick with DOS 3.3, enhanced by Dave, or Bill Basham, or Art Schumer, or others. And if you MUST have at least 32K of program space with Applesoft; or you MUST have Integer BASIC support; or you MUST have compatibility with hundreds of existing software products; then stick with DOS 3.3.

```
################################################################################
#  More on ProDOS and Nonstandard Apples
################################################################################
```

June 1984

In the March issue we published Bob Stout's note on how to make ProDOS boot in a Franklin computer. The current issue, (No. 9) of Hardcore Computist points out that the address given in that note didn't work for the ProDOS version dated 1-JAN-84. Apparently Bob was referring to an earlier version. The correct address for the NOPs is $265B.

In a similar vein, inside this issue Jan Eugenides points out that ProDOS will also fail in an Apple with a modified Monitor ROM. He then gives a slightly different patch to defeat the check code.

```
################################################################################
#  Booting ProDOS with a Modified Monitor ROM
################################################################################
```

Jan Eugenides

June 1984

You may have already figured this out, but ProDOS won't boot if you have installed S. Knouse's modified ROM in your Apple. This can easily be fixed, as follows:

* On track 1, sector C, change bytes B4-B6 from AE B3 FB to A2 EA EA. This tells ProDOS your machine is a II+. If it's a //e, make B5 an A0 instead.
* On track 1, sector 9, change bytes 60-61 from A9 00 to A5 0C. This defeats the ROM check routine.

Ta daaa! Now ProDOS works just fine with your modified ROM.

```
################################################################################
#  Finding Memory Size in ProDOS
################################################################################
```

Bob Sander-Cederlof

March 1985

On page 6-63 of Beneath Apple ProDOS there is a small piece of code designed to determine how much memory there is:

```
       LDA $BF98
       ASL
       ASL
       BIT 0
       BPL SMLMEM      48K
       BVS MEM128      128K
       ...             otherwise 64K
```

The code will not work. The BIT 0 will test bits 7 and 6 of memory location $0000, which have nothing whatsoever to do with how much memory is in your machine. What was intended was to test bits 7 and 6 of the A-register, or in other words bits 5 and 4 of $BF98. Here is one way you can do that:

```
       LDA $BF98
       ASL
       ASL
       ASL
       BCC SMLMEM      48K
       BMI MEM128      128K
       ...             OTHERWISE 64K
```

Notice that not only does this perform the test correctly, it is also one byte shorter!

If you insist on using the same number of bytes, here is another way to test those bits:

```
       LDA $BF98
       AND #%00110000  ISOLATE BITS 5 AND 4
       CMP #%00100000
       BCC SMLMEM      48K
       BNE MEM128      128K
       ...             OTHERWISE 64K
```

If any of you have discovered any other problems with the sample code in this book, pass them along.

##
Shrinking Code Inside ProDOS
##

Bob Sander-Cederlof

April 1985

David Johnson challenged me a few days ago. We were talking about ProDOS: the need for a ProDOS version of the S-C Macro Assembler, the merits vs. enhanced DOS 3.3, and the rash of recent articles on shrinking various routines inside DOS to make room for more features.

I've been avoiding ProDOS as much as possible, trying not to notice its ever-increasing market-share. Dave's comment, "ProDOS is a fertile field for your shrinking talent," may have finally pushed me into action.

I am trying to make the ProDOS version of the S-C Macro Assembler, but is hard. I have Apple's manuals, Beneath Apple ProDOS, and the supplement to the latter book which explains almost every line of ProDOS code. Nevertheless, version 1.1.1 of ProDOS doesn't seem to conform to all these descriptions in every particular. I spent four hours last night chasing one little discrepancy. (Turned out to be my own bug, though.)

In the process, I ran across the subroutine ProDOS uses to convert binary numbers to decimal for printing. In version 1.1.1 it starts at $A62F, and with comments looks like this.

```
1000 *SAVE S.PRODOS NUMOUT
1010 *--------------------------------
1020        .OR $A62F
1030        .TA $800
1040 *--------------------------------
1050 *   CONVERT 00.XX.AA FROM BINARY TO DECIMAL
1060 *   STORE UNITS DIGIT AT $201,Y
1070 *   STORE OTHER DIGITS AT SUCCESSIVE LOWER ADDRESSES
1080 *
1090 *      Note:  it is assumed and required that
1100 *             ACCUM+2 already by zeroed!
1110 *             Either that, or already set to the
1120 *             highest byte of a 24-bit value.
1130 *--------------------------------
1140 CONVERT.TO.DECIMAL
1150        STX ACCUM+1
1160        STA ACCUM
1170 .1     JSR DIVIDE.ACCUM.BY.TEN
1180        LDA REMAINDER
1190        ORA #"0"
1200        STA BUFFER+1,Y
1210        DEY
1220        LDA ACCUM    CHECK IF QUOTIENT ZERO
1230        ORA ACCUM+1
1240        ORA ACCUM+2
1250        BNE .1
1260        RTS
1270 *--------------------------------
```

```
1280 DIVIDE.ACCUM.BY.TEN
1290        LDX #24      24 BITS IN DIVIDEND
1300        LDA #0       START WITH REM=0
1310        STA REMAINDER
1320 .1     JSR SHIFT.ACCUM.LEFT
1330        ROL REMAINDER
1340        SEC          REDUCE REMAINDER MOD 10
1350        LDA REMAINDER
1360        SBC #10
1370        BCC .2       STILL < 10
1380        STA REMAINDER
1390        INC ACCUM    QUOTIENT BIT
1400 .2     DEX          NEXT BIT
1410        BNE .1
1420        RTS
1430 *---------------------------------
1440 ACCUM      .EQ $BCAF,BCB0,BCB1
1450 REMAINDER  .EQ $BCB2
1460 BUFFER     .EQ $0200
1470 *---------------------------------
1480        .OR $AAD7
1490        .TA $900
1500 *---------------------------------
1510 SHIFT.ACCUM.LEFT
1520        ASL ACCUM
1530        ROL ACCUM+1
1540        ROL ACCUM+2
1550        RTS
1560 *---------------------------------
1570        .LIF
```

The conversion routine is designed to handle values between 0 and $FFFFFF. The heghest byte must already have been stored at ACCUM+2 before calling CONVERT.TO.DECIMAL. The middle byte must be in the X-register, and the low byte in the A-register. The decimal digits will be stored in ASCII in the $200 buffer, starting and $201+Y and working backwards.

One way of converting from binary to decimal is to perform a series of divide-by-ten operations. After each division, the remainder will be the next digit of the decimal value, working from right to left. That is the technique ProDOS uses, and the division is done by the subroutine in lines 1280-1420.

The dividend is in ACCUM, a 3-byte variable. The low byte is first, then the middle, and finally the high byte. One more byte is set aside for the remainder. A 24-step loop is set up to process all 24 bits of ACCUM. In the loop ACCUM and REMAINDER are shifted left. If REMAINDER is 10 or more, it is reduced by ten and the next quotient bit set to 1; otherwise the next quotient bit is 0.

The first possible improvement I noted was in the area of lines 1330-1360. the ROL REMAINDER will always leave carry status clear, because we never let REMAINDER get larger than 9. If we delete the SEC instruction, and change SBC #10 to SBC #9 (because carry clear means we need to borrow), we can save one byte. But that's not really worth the effort.

Next I realized that REMAINDER could be carried in the A-register within the 24-step loop, and not stored until the end of the loop. Here is that version, which saves seven bytes (original = 31 bytes, this one = 24 bytes):

```
1260 DIVIDE.ACCUM.BY.TEN
1270        LDX #24      24 BITS IN DIVIDEND
1280        LDA #0       START WITH REM=0
1290 .1     JSR SHIFT.ACCUM.LEFT
1300        ROL
1310        CMP #10
1320        BCC .2       STILL < 10
1330        SBC #10
1340        INC ACCUM    QUOTIENT BIT
1350 .2     DEX          NEXT BIT
1360        BNE .1
1370        STA REMAINDER
1380        RTS
```

To make sure my version really worked, I re-assembled the conversion program with an origin of $800, and appended a little test program. Here is my test program, which converts the value at $0000...0002 and prints it out.

```
1510 T      LDA 0
1520        STA ACCUM+2
1530        LDX 1
1540        LDA 2
1550        LDY #10
1560        JSR CONVERT.TO.DECIMAL
1570 .1     INY
1580        LDA BUFFER+1,Y
1590        JSR $FDED
1600        CPY #10
1610        BCC .1
1620        RTS
```

My best version is yet to come. I considered the fact that we could SHIFT the next quotient bit into the low end of ACCUM rather than using INC ACCUM to set a one-bit. I rearranged the loop so that the remainder reduction was done first, followed by the shift-left operation. I had to change the remainder reduction to work modulo 5 rather than 10, because the shifting operation came afterwards. I also had to inlcude my own three lines of code to ROL ACCUM, since the little subroutine in ProDOS started with ASL ACCUM. The result is still shorter than 31 bytes, but only four bytes shorter. Nevertheless, it is faster and neater, in my opinion.

```
1640 DIVIDE.ACCUM.BY.TEN.SHORTEST
1650        LDX #24      24 BITS IN DIVIDEND
1660        LDA #0       START WITH REM=0
1670 .1     CMP #5
1680        BCC .2       STILL < 10
1690        SBC #5
1700 .2     ROL ACCUM
1710        ROL ACCUM+1
1720        ROL ACCUM+2
1730        ROL
1740        DEX          NEXT BIT
1750        BNE .1
```

```
1760        STA REMAINDER
1770        RTS
```

##
Review: Apple ProDOS: Advanced Features for Programmers
##

Bill Morgan

May 1985

Gary Little, the prolific author of Inside the Apple //e and Inside the Apple //c, has yet another new book out. This one is called Apple ProDOS: Advanced Features for Programmers. In this volume Little covers just about all you need to know to write assembly language programs under ProDOS, from simply passing commands to BASIC.SYSTEM, through great detail on all the MLI calls, to writing your own interrupt handlers and device drivers.

Here's a quick summary of the book's contents:

1. An Introduction to ProDOS -- Little starts out with the history of Apple's DOS's, a comparison of ProDOS and DOS 3.3, and a summary of important features of ProDOS.
2. Files and File Management -- Here he covers the directory structures, file structures, disk formatting, and gives us a READ.BLOCK program.
3. Loading and Installing ProDOS -- This chapter goes into the boot process, ProDOS' memory usage, and the Global Page.
4. The Machine Language Interface -- This is the information on using the MLI, its error codes, and complete details of all MLI calls.
5. System Programming Featuring BASIC.SYSTEM -- Here we have a discussion of system programs, the structure and commands of BASIC.SYSTEM, and assembly language programming under BASIC.SYSTEM.
6. Interrupts -- In this chapter Little covers interrupts in general, ProDOS interrupt handling, and programming the Apple mouse.
7. Disk Drivers -- Nearing the end, we go into identifying and handling foreign disk drivers, driver commands, the /RAM driver, and adding your own driver.
8. ProDOS Clock Drivers -- And finally we find out about using the built-in clock support, adding a clock driver, and reading the date and time from Applesoft.

An important strength of this book is the wealth of examples. In the chapter on the Machine Language Interface there is an example of the correct use of EVERY MLI call. The BASIC.SYSTEM chapter includes an ONLINE command, to identify all disk volumes currently on line. The chapter on interrupts contains a couple of examples of mouse programming. The Disk driver section has a listing of a simple /RAM driver for main memory. And this is just a sample of the useful code provided in Little's new book. A companion disk containing all of the book's programs and more is available for $25.00 from the author.

I hear some of you asking: How does Apple ProDOS: Advanced Features (APAF) compare to Beneath Apple ProDOS (BAP)? Well, the two books complement each other quite nicely. With all its examples, treatment of interrupt handlers and device drivers, and overall clarity, I'd say that APAF is the better book on programming under ProDOS. BAP has useful examples as well, and better detail about the internals of diskette formatting and how ProDOS works, especially with its 120+ page supplement describing the code on a line-by-line basis. So if you're concerned with understanding the inner workings of the operating system, or with modifying its behavior, BAP is the book to have. Otherwise, get APAF for the best information on programming using ProDOS. Personally, I'm glad to have both books on the shelf here, along with Apple's ProDOS Technical Reference Manual.

Apple ProDOS: Advanced Features for Programmers, by Gary B. Little. Brady Communications Co., 1985. 266+iv pp., Reference Card. $17.95. Available from S-C Software for $17 + shipping.

```
################################################################################
#  DATE Command for ProDOS
################################################################################
```

Bill Morgan

May 1985

One of the nice new features in ProDOS is the way the diskette catalog shows the date of creation and last modification for each file, IF you have a clock/calendar card installed in your Apple. Well I don't have such a card in either of the Apples I use regularly, at work or at home. And no //c has a clock! (Yet, at least. I'll bet someone will come up with a way...)

Anyway, I got tired of always seeing <NO DATE> and started figuring out how to set a date without a clock to do it for me. A look at Beneath Apple ProDOS informed me that the current date is transformed into the format YYYYYYYMMMMDDDDD and stored (in the usual 6502 low byte/high byte sequence) at $BF90-BF91 in the ProDOS Global Pages (the fixed locations of all of the accessible system variables). The first thing I did was manually convert the current date into that format and poke it in from the Monitor. That went like this:

```
                         $BF90      $BF91
May =  $5 = 0101       MMM DDDDD  YYYYYYY M
 10 =  $A = 01010      101 01010  1010101 0
'85 = $55 = 1010101       $AA        $AA
```

So, the values to poke into $BF90-91 were $AA and $AA. What better time than a four-A day to start such a project!

That experiment worked just fine: the next file I saved on the disk showed creation and modification dates of 10-MAY-85, just as I had hoped. With that success under my belt the next step had to be to come up with a program to read and/or set those date bytes. And, while I'm at it, why not take advantage of ProDOS' built-in hooks for installing new commands and add a DATE command to the operating system?

How do I go about adding a command? The ProDOS Technical Reference Manual is pretty sketchy on the subject, but two other books, Beneath Apple ProDOS and the new Apple ProDOS: Advanced Features for Programmers, have good descriptions and examples of the procedure. If you're going to do much assembly language programming under ProDOS you should have one or both of those books.

When ProDOS fails to recognize a command it does a JSR EXTRNCMD ($BE06) to find out if an external command processor will claim this one. What I have to do is install the address of DATE in $BE07-08, after moving the address that was already there into a JMP instruction. This way, if DATE doesn't recognize the command it can pass it along to any other processor that might have been there before.

Processing of an external command is normally divided into two phases, a parser and a handler. The parser section will scan the command name at the beginning of the line. If the command is not recognized, the parser should set the carry bit and JMP on to the address found in EXTRNCMD to see if another external processor will claim it.

If the command is recognized, the parser can set certain bits in PBITS ($BE54-55) to signify which parameters are permitted or required on the command line, and store the address of the handler in EXTRNADDR ($BE50-51). After storing the command length minus one in XLEN ($BE52) and a zero in XCNUM ($BE53), to signify that an external processor did claim the command, the parser then returns control to ProDOS to scan the rest of the line. If the line was syntactically correct, ProDOS will return the values of the parameters in a set of standard locations ($BE58-6F) and pass control back to the handler address specified.

Since DATE is a simple processor that uses a nonstandard parameter, I just set PBITS to zero, to indicate no parsing necessary, and store the address of an RTS instruction in EXTRNADDR. I then proceed to do all my processing before returning to ProDOS.

There is one additional wrinkle to using an external command with ProDOS: where do I put my code so ProDOS, Applesoft, and others don't stomp all over it? In the interest of simplicity I have ignored that problem here. The best procedure, as shown in the books mentioned above, is to call ProDOS to assign me a buffer and then relocate my code into that buffer. The examples in the books provide details of this process.

Now, let's take a look at the code:

Lines 1310-1400 install DATE by moving the current External Command address to my exit JMP instruction and storing DATE's address in the vector.

Lines 1440-1540 check the input buffer to see if this is a DATE command. If not we branch on down to that JMP instruction where we earlier put the address found in the External Command vector. This passes control either on to the next external command in the chain, or back to ProDOS for a SYNTAX ERROR.

If the command matched we go on to lines 1560-1650 to do the necessary housekeeping. This involves storing the command length-1 in the Global Page, setting a couple of flags to tell ProDOS not to parse the rest of the command line, and that an external command has taken over. Then we supply a handler address for the second half of ProDOS' processing, which in this case is just an RTS instruction. Finally we reach lines 1670-1690, where we check to see if the character following DATE is a Carriage Return. If so we branch forward to RETURN.DATE to display the existing date.

If there is more than just DATE on the command line, we must want to set a new date, so we fall into SET.DATE at line 1710. This routine makes heavy use of ACCUMULATE.DIGITS at line 2400, so we'll examine that code first. The first step is to zero the byte where we'll be accumulating the value typed in. Then we scan forward in the input buffer, looking for a nonblank character. When we find one we first check to see if it is a slash, which marks the end of a number, or a Carriage Return, which marks the end of the line. If it was either of those we exit, setting the Carry bit to indicate which one we found.

If the character found was not a delimiter we next check to see if it is a number. If not, we have a SYNTAX ERROR. When we do get a number, we strip off the high bits to convert the ASCII code to a binary value, and save that value. We then multiply the previous value in ACCUM by 10 and add in the new value. Then it's back to line 2440 to get another character. Lines 2710-2730 load the A-register with the value found and branch to the error exit if that value was zero.

Now, back to SET.DATE. That routine begins at line 1720 with a DEY to get ready for the INY at the beginning of ACCUMULATE.DIGITS. We then get the month, check for a legal value, and store it. Next we get the day, save the status, and check and save that value. Then it's time to check the status to see if the day was followed by a slash, or by a Carriage Return. If it was a slash then a year was specified, so we go get that value. If it was a Return no year was present, so we use 1985. (I guess that means we'll have to reassemble or patch this program every year. I think I can handle that.)

The last step in SET.DATE is to fold the year, month, and day together as described above and store the results in the Global Page. The comments in the listing illustrate how the bits are shuffled around to the correct format. After setting the date we fall into RETURN.DATE to display the result.

RETURN.DATE, at lines 2080-2290, is quite straightforward. It just gets the bytes from the Global Page, unfolds them, and calls DEC.OUT to translate them to decimal numbers and display those numbers. Again, the comments illustrate the bit manipulations involved in the unfolding process.

The final section of code is DEC.OUT, at lines 2750-2910. In lines 2760-2810 we use the Y-register to count how many times we can subtract 10 from the number passed in the A-register. Then lines 2830-2910 restore and save the A-register, make sure the tens count is non-zero, convert it to a character and print it. We then recover the units value and print that out.

```
1000 *SAVE S.DATE
1010 *--------------------------------
1020 *
1030 *      Program to read or set the
1040 *      date bytes in the Global Page
1050 *
1060 *            by Bill Morgan
1070 *
1080 *--------------------------------
1090 POINTER    .EQ $40,41
1100 ACCUM      .EQ $42
1110 MONTH      .EQ $43
1120 DAY        .EQ $44
1130 TEMP       .EQ $45
1140
1150 WBUF       .EQ $200
1160
1170 EXTRNCMD   .EQ $BE07
1180 EXTRNADDR  .EQ $BE50,51
1190 XLEN       .EQ $BE52
1200 XCNUM      .EQ $BE53
1210 PBITS      .EQ $BE54
1220 GP.DATE    .EQ $BF90
1230
1240 PRAX       .EQ $F941
1250 CROUT      .EQ $FD8E
1260 COUT       .EQ $FDED
1270 *--------------------------------
1280        .OR $803
1290 *      .TF B.DATE
1300 *--------------------------------
1310 INSTALL
1320        LDA EXTRNCMD+1    exit to old
1330        STA EXIT+2        user command
1340        LDA EXTRNCMD
1350        STA EXIT+1
```

```
1360        LDA /DATE          become new
1370        STA EXTRNCMD+1     user command
1380        LDA #DATE
1390        STA EXTRNCMD
1400        RTS
1410 *--------------------------------
1420 COMMAND .AS /DATE/
1430 *--------------------------------
1440 DATE   LDY #0
1450        STY POINTER        point to input buffer
1460        LDA /WBUF
1470        STA POINTER+1
1480 .1     LDA (POINTER),Y    scan command
1490        AND #%01111111
1500        CMP COMMAND,Y
1510        BNE ERR.BRIDGE     not mine
1520        INY
1530        CPY #4
1540        BCC .1
1550 *--- ProDOS bookkeeping ---------
1560        DEY
1570        STY XLEN           command length - 1
1580        INY
1590        LDA #0
1600        STA PBITS          don't parse parms
1610        STA XCNUM          external command
1620        LDA #RTS1
1630        STA EXTRNADDR      no execution after
1640        LDA /RTS1          command parsing
1650        STA EXTRNADDR+1
1660 *--- set or display date? -------
1670        LDA (POINTER),Y
1680        CMP #$8D           DATE only?
1690        BEQ RETURN.DATE    yes, return old date
1700 *--------------------------------
1710 SET.DATE
1720        DEY
1730        JSR ACCUMULATE.DIGITS  get month
1740        CMP #13
1750        BCS ERROR          >12 no good
1760        STA MONTH
1770        JSR ACCUMULATE.DIGITS  get day
1780        PHP                save status
1790        CMP #32
1800        BCC GO.ON          <=31 ok
1810
1820        PLP
1830 ERR.BRIDGE
1840        BNE ERROR      ...always
1850
1860 GO.ON  STA DAY
1870        PLP                recover status
1880        BCC .1             .CC. if "/"
1890        LDA #85            year defaults to '85
1900        BNE .2         ...always
1910 .1     JSR ACCUMULATE.DIGITS  get year
```

```
1920        CMP #100
1930        BCS ERROR         >99 no good
1940 .2     PHA               save year
1950        LDA MONTH         X 0000MMMM
1960        LSR               M 00000MMM
1970        ROR               M M00000MM
1980        ROR               M MM00000M
1990        ROR               M MMM00000
2000        STA MONTH
2010        PLA               M 0YYYYYYY
2020        ROL               0 YYYYYYYM
2030        STA GP.DATE+1
2040        LDA MONTH           MMM00000
2050        ORA DAY             MMMDDDDD
2060        STA GP.DATE
2070 *--------------------------------
2080 RETURN.DATE
2090        JSR CROUT
2100        LDA GP.DATE+1     X YYYYYYYM
2110        LSR               M 0YYYYYYY
2120        PHA
2130        LDA GP.DATE       M MMMDDDDD
2140        PHA
2150        ROR               X MMMMDDDD
2160        LSR               X 0MMMMDDD
2170        LSR               X 00MMMMDD
2180        LSR               X 000MMMMD
2190        LSR               X 0000MMMM
2200        JSR DEC.OUT       display month
2210        LDA #"/"          /
2220        JSR COUT
2230        PLA               X MMMDDDDD
2240        AND #%00011111    X 000DDDDD
2250        JSR DEC.OUT       display day
2260        LDA #"/"          /
2270        JSR COUT
2280        PLA               X 0YYYYYYY
2290        JSR DEC.OUT       display year
2300 *--------------------------------
2310 GOOD.EXIT
2320        CLC               signal no error
2330 RTS1   RTS
2340 *--------------------------------
2350 ERROR1 PLA               clean up
2360        PLA               return addresses
2370 ERROR  SEC               signal error
2380 EXIT   JMP RTS1          INSTALL makes address
2390 *--------------------------------
2400 ACCUMULATE.DIGITS
2410        LDA #0
2420        STA ACCUM         zero accumulator
2430
2440 .1     INY               next character
2450        LDA (POINTER),Y
2460        AND #%01111111    hi-bit off
2470        CMP #' '          space?
```

```
       BEQ .1           back for another
       CMP #'/'         slash?
       BEQ .2           yes, exit .CC.
       CMP #$0D         <CR>?
       BEQ .3           yes, exit .CS.
       CMP #'0'         too small?
       BCC ERROR1       not digit
       CMP #'9'+1       too big?
       BCS ERROR1       not digit

       AND #%00001111   isolate value
       STA TEMP         stash it
       LDA ACCUM
       ASL              X 2
       ASL              X 4
       ADC ACCUM        X 5
       ASL              X 10
       ADC TEMP         add new digit
       BCS ERROR1       too big
       STA ACCUM
       BCC .1        ...always

.2     CLC              .CC. if /
.3     LDA ACCUM        return value
       BEQ ERROR1       0 no good
       RTS
*--------------------------------
DEC.OUT
       LDY #0           zero counter
       SEC              get ready
.1     SBC #10          subtract 10
       BCC .2           borrow?
       INY              count a 10
       BPL .1        ...always

.2     ADC #10          restore borrow
       PHA              save units
       TYA              print 10's count
       BEQ .3           no leading zero
       ORA #$B0         make character
       JSR COUT         print it
.3     PLA              recover units
       ORA #$B0         make character
       JMP COUT         return through COUT
```

```
################################################################################
#  Reading DOS 3.3 Disks With ProDOS
################################################################################
```

Bob Sander-Cederlof

July 1985

At the track and sector level, DOS 3.3 disks are identical to ProDOS disks. They both have 35 tracks, 16 sectors, and the sectors are laid out on the tracks the same way in both systems. You can use DOS's COPYA program to copy ProDOS disks, and you can use some ProDOS utilities on DOS disks.

The structure of the files is of course entirely different between the two systems. Hence the need for the CONVERT program found on ProDOS system master disks, and the System Utilities Disk that comes with the //c. Unfortunately both of the above programs have bugs that get in the way nearly every time I want to move a file from DOS to ProDOS. The one that bites me the most is the way CONVERT dies when it encounters a DOS filename which does not start with a letter. We routinely use such "illegal" filenames on our disks to separate and identify sections of long catalogs, but CONVERT goes absolutely crazy when it finds one.

Therefore, I decided to write a program which could "LOAD" assembler source files from a DOS 3.3 disk while I am running the ProDOS version of the S-C Macro Assembler. Even with error messages and other fancy features, the program turns out to be only a little over $280 bytes long, and it works.

It is based on the fact that the Block Read MLI call does not care whether the disk being read is a DOS or a ProDOS disk. The Block Read MLI call reads 512 bytes, or two sectors, at a time. The call looks like this:

```
       JSR $BF00       (MLI link in global page)
       .DA #$80        (block read code)
       .DA PARMLIST    (address of parameters)
```

MLI returns with carry clear if there was no error, or carry set if there was an error. The error code will be in the A-register if there was an error.

The PARMLIST for Block Read looks like this:

```
    PARMLIST .DA #3         (3 parameters)
             .DA #$60       (1-byte unit number)
             .DA BUFFER     (address of 512-byte buffer)
             .DA 2          (2-byte block number)
```

Page 3-17 of "Beneath Apple ProDOS" contains a table which converts block numbers to physical track/sector, and vice versa. The latest printing of the book also includes a line which correlates the physical sector values to the DOS 3.3 logical sector. Boiling it down, you can derive a ProDOS block number from the DOS 3.3 logical sector by multiplying the track number by 8 and adding a value according to the sector number from the following table:

```
    DOS sector #:  0 1 2 3 4 5 6 7 8 9 A B C D E F
                   0 7 6 6 5 5 4 4 3 3 2 2 1 1 0 F
```

For example, track 0 sector 2 is in ProDOS block 6. The only problem is, so is DOS track 0 sector 3. We also need to remember whether a given sector is in the upper or lower half of a 512-byte block.

I developed the following subroutine, which will translate the DOS logical track and sector numbers into the appropriate block number, read the block, and return with the address of the buffer page in which the sector data has been read. Call the routine with the track number in the A-register and the sector number in the X-register. The high-byte of the buffer address will return in the X-register. If MLI detects an error, the subroutine will return with carry set.

```
RTS     LDY #0  ASSUME BLOCK # < $100
        ASL     FORM TRACK*8
        ASL
        ASL
        BCC .1  ...BLOCK < $100
        INY     ...BLOCK > $0FF
.1      ASL     *2, MAKE ROOM FOR H/L FLAG BIT
        ORA BLKTBL,X   MERGE FROM SECTOR TRANSLATION
        ROR     H/L FLAG BIT TO CARRY
        STA BLOCK
        STY BLOCK+1
        LDX /BLOCK.BUFFER   HIGH BYTE OF BUFFER ADDRESS
        BCC .2  ...LOWER HALF OF BUFFER
        INX     ...UPPER HALF OF BUFFER
.2      JSR $BF00
        .DA #$80,PARMLIST
        RTS

BLKTBL .HS 00.0E.0D.0C.0B.0A.09.08
        .HS 07.06.05.04.03.02.01.0F

PARMLIST
        .DA #3
        .DA #$60        SLOT 6, DRIVE 1
        .DA BLOCK.BUFFER
BLOCK   .DA 0           <FILLED IN>
```

After playing with the subroutine a while, I proceeded to write the load program. Using a well-worn copy of "Beneath Apple DOS", I figured out once more how to work through a DOS catalog. I decided to display a menu of files on the screen, and allow a single keystroke to select a file to be loaded.

The program that follows is designed to work with the ProDOS version of the S-C Macro Assembler. Assuming it has been assembled and is in a ProDOS binary file as DOS.LOAD, and assuming you have booted the ProDOS version of the S-C Macro Assembler, you can start up the load program by typing "-DOS.LOAD". It will load source files from DOS disks, which are DOS type I files, and place them in the assembler's edit area. After selecting the slot and drive, the program reads the DOS catalog and displays 20 filenames at a time. Only type I filenames are displayed, any others are skipped over. If there are more than 20 files, you can page through them. If you change your mind about loading a file, you can abort. If you see the file you want to load, you type a single letter to select it. A few seconds later it has been loaded, and you are returned to the assembler.

The assembler's soft entry point is at $8003, and the load program jumps there after finishing a load or after encountering an error. Three pointer locations in page zero which the assembler uses are used by the load program: HIMEM ($73,74) points one byte higher than the program can be loaded; PP ($CA,CB) will point to the beginning of the program, if it is successfully loaded; LOMEM ($67,68) points to the lowest address the program can occupy. HIMEM is normally at $7400, and LOMEM at $1000, but these can be changed with the HIMEM and LOMEM commands. LOMEM could be set as low as $0800.

With these limitations on the program extent ($0800...73FF), you can see that the maximum size assembler source file that can be loaded from a DOS disk is $6C00 bytes, or 108 sectors. Or, if you prefer to leave LOMEM at $1000, you can load $6400 bytes or 100 sectors. Most likely you do not have any source files which are bigger than that anyway. If you do, you need to load the DOS version of the assembler and split the files before they can be transferred to ProDOS. The maximum size file of 108 data sectors would only have one track/sector list, so I did not include any logic to chain to a second track/sector list. You may be wondering where the load program itself loads....

The command interpreter I developed for the ProDOS version of the S-C Macro Assembler has three 1024-byte buffers permanently allocated between $7400 and $7FFF. None of them will be in use while the load program is executing, so I borrowed some of that space for the load program. The load program itself loads inside the buffer space allocated to the EXEC command, at $7400-77FF. The blocks read by MLI will be stored at $7C00-7DFF, and I will save a copy of the track/sector list for the file being loaded at $7E00-7EFF.

Now for a description of the actual code. Lines 1270-1410 ask you to type in the slot and drive numbers of the floppy drive the DOS disk is in. ProDOS uses a "unit number", which is a coded form of the slot and drive all in one byte. The slot number is in bits 4-6, and the drive number (0 or 1, corresponding to drives 1 or 2 respectively) in bit 7. My subroutine GETNUM prints a prompt message (selected by the Y-register), inputs a single character from the keyboard, and checks it for legal range. GETNUM is designed to accept only digits, starting with "1", and up to but not including the value in the A-register when GETNUM is called.

Once the unit number has been established, we fall into the LOAD.MENU code. This code is somewhat convoluted, enough to disgust even me. Interlocking loops? Multiple entries and exits? Ouch! Maybe it really IS structured code, but just not in Euclidean space. I think maybe it could be diagrammed on the surface of a Klein bottle (recursive torus?).

Anyway, let's walk through it. Line 1440-1500 set up a fresh menu display and read in the DOS VTOC page so we can start reading the catalog. The second and third bytes in the VTOC page give the track and sector of the first catalog sector. This is almost always track $11, sector $0F; however, by starting at VTOC, we are a little more general. We are still assuming we know where the VTOC is, which is track $11, sector 0. Some non-standard software sets up disks with the VTOC somewhere else, but you are very unlikely to find any S-C source code on such a disk. Each sector of the catalog also contains the track/sector of the next catalog sector in the 2nd and 3rd bytes.

Lines 1530-1550 read in the next catalog sector and set the pointer to the first file entry in that sector. Each file entry is 35 bytes long, and the first one starts at $0B within the sector. The subroutine READ.NEXT.CATALOG.SECTOR will return with carry set if there are no more catalog sectors. The first time through this code, when we fall in from the code above, we will read the first catalog sector.

Lines 1570-1960 pick up filenames out of the catalog sectors and write them on the screen. Not all file names are used: line 1610 filters out deleted files; lines 1660-1700 filter out files which are not type I. The track and sector of the active type-I files are saved in an array, indexed by the menu letter. These values are first picked up in lines 1620-1650, and added to the array in lines 1870-1940. Lines 1720-1770 print the menu letter and two dashes, and then lines 1780-1850 print the filename.

Lines 1950-1960 decrement the line count and test if the screen is full yet. I arbitrarily call a screen full if it has 20 filenames, leaving room for my three-line prompt message. We jump to MENU.SELECTION when we reach 20 lines or when we reach the end of the catalog, whichever comes first.

If we are not yet at the end of catalog and have not yet filled the screen, or if the file was one that got filtered out of the menu, we come to GET.NEXT.FILE at line 1980. Lines 1990-2040 update the pointer into the catalog sector so that it points at the next file, if there is another one. If so, we branch back to NEXT.FILE.NAME, to try the next one in the current sector. If no more names in this sector, we go back to NEXT.CAT.SECTOR to get the next catalog sector (if any).

When we reach the end of catalog, lines 2070,2080 set a flag. We need a flag to tell whether it was screen-full or catalog- end which caused us to come to MENU.SELECTION, so we can either continue through the catalog or wrap-around to the beginning should you wish to see another screenful of filenames.

The MENU.SELECTION section prints a three-line prompt message and waits for you to type a character. If you type a space, you seethe next screenful of filenames. (Of course, if there are fewer than 21 type I files on the disk you will see the same ones over again.) If you type the RETURN or ESCAPE keys, the load program will abort, returning directly to the assembler without loading a file. If you type a letter in the range of the menu, that file will be loaded. Any other key is ignored.

Lines 2260-2370 convert the menu letter you typed into an index to get the track and sector for the track/sector list of the selected file. The track/sector list contains the track and sector for every data sector in the file. Line 2310 reads the track/sector list, and lines 2330-2370 copy it into a special buffer.

The first two bytes of the first data sector of a type-I file contain the length of the file. We need to know the length so we can figure out where to read the data. Lines 2390-2510 read in the first data sector and get the file size.

Lines 2520-2630 figure out where PP should be set so that the file exactly fits between PP and HIMEM, and checks to make sure that it does not go below LOMEM.

Lines 2650-2670 copy the rest of that first sector into the load area, starting at PP. If the file is so short it doesn't fill the first data sector, the LOAD.FROM.SECTOR subroutine will return with carry set and we will return to the assembler, all finished. Otherwise, we fall into the code below, to load the succeeding data sectors. Eventually we will bump into HIMEM, and we are finished.

Now that this program is working I can see neat ways to extend it. Why restrict it to type-I files? It could also BLOAD type-B files, as long as an appropiate load address was set up. It could do the equivalent of a BLOAD on a type-T file, which then could be BSAVE as type TXT in ProDOS. Seems like we might be able to do away with the need for CONVERT, at least in the direction of moving from DOS to ProDOS.

```
1000 *SAVE S.DOS.LOAD
1010 *--------------------------------
1020        .OR $7400
1030        .TF DOS.LOAD
1040 *--------------------------------
1050 PNTR          .EQ $00,01
1060 CAT.INDEX     .EQ $02
1070 MENU.LETTER   .EQ $03
1080 LINE.COUNT    .EQ $04
1090 TRACK         .EQ $05
1100 SECTOR        .EQ $06
1110 DONE.FLAG     .EQ $07
1120 SIZE          .EQ $08,09
1130 LIMIT         .EQ $0A
1140 *--------------------------------
1150 LOMEM         .EQ $67,68
1160 HIMEM         .EQ $73,74
1170 PP            .EQ $CA,CB
1180 *--------------------------------
1190 BLOCK.BUFFER  .EQ $7C00
1200 TS.LIST       .EQ $7E00
1210 *--------------------------------
1220 MON.RDKEY     .EQ $FD0C
1230 MON.CROUT     .EQ $FD8E
1240 MON.PRHEX     .EQ $FDDA
1250 MON.COUT      .EQ $FDED
1260 *--------------------------------
1270 DOS.LOAD
1280        LDY #EM3     "SLOT:"
1290        LDA #"8"     1...7
1300        JSR GETNUM   00000SSS
1310        LSR          000000SS S
1320        ROR          S000000S S
1330        ROR          SS000000 S
1340        ROR          SSS00000
1350        STA UNIT
1360        LDY #EM4     "DRIVE:"
1370        LDA #"3"     1...2
1380        JSR GETNUM
1390        LSR
1400        LSR
1410        ROR UNIT     DSSS0000
1420 *--------------------------------
1430 LOAD.MENU
1440        JSR SETUP.SCREEN
1450        LDA #17           TRACK 17
1460        LDX #0            SECTOR 0
1470        STX DONE.FLAG
1480        STX PNTR
1490        JSR RTS           READ DOS 3.3 VTOC
1500        STX PNTR+1        SET POINTER
1510 *--------------------------------
1520 NEXT.CAT.SECTOR
1530        JSR READ.NEXT.CATALOG.SECTOR
1540        BCS END.OF.CATALOG
1550        LDY #$0B
```

```
*--------------------------------
NEXT.FILE.NAME
       STY CAT.INDEX
       LDA (PNTR),Y      TRACK
       BEQ END.OF.CATALOG
       BMI GET.NEXT.FILE ...DELETED FILE
       STA TRACK
       INY
       LDA (PNTR),Y
       STA SECTOR
       INY
       LDA (PNTR),Y      FILE TYPE
       ASL               INGORE LOCK BIT
       CMP #2            MUST BE TYPE I
       BNE GET.NEXT.FILE ...NOT I, SKIP OVER IT
*---DISPLAY MENU LINE------------
       LDA MENU.LETTER
       JSR MON.COUT      DISPLAY MENU LETTER,
       INC MENU.LETTER
       LDA #"-"
       JSR MON.COUT      ...TWO DASHES
       JSR MON.COUT
       LDX #30
.1     INY
       LDA (PNTR),Y
       ORA #$80
       JSR MON.COUT      ...AND FILENAME
       DEX
       BNE .1
       JSR MON.CROUT
*---SAVE T/S OF TS-LIST----------
       LDA MENU.LETTER
       AND #$1F          CONVERT TO INDEX
       TAX
       DEX               ...SINCE LETTER INC'ED ALREADY
       LDA TRACK
       STA TRACKS,X
       LDA SECTOR
       STA SECTORS,X
       DEC LINE.COUNT
       BEQ MENU.SELECTION  BRANCH IF SCREEN FULL
*--------------------------------
GET.NEXT.FILE
       CLC
       LDA CAT.INDEX
       ADC #35
       TAY               BUMP INDEX
       BCC NEXT.FILE.NAME
       BCS NEXT.CAT.SECTOR
*--------------------------------
END.OF.CATALOG
       LDA #1
       STA DONE.FLAG
MENU.SELECTION
       LDY #EM0          3-LINE PROMPT
       JSR PRINT.MSG
```

```
2120 .2     JSR MON.RDKEY
2130        CMP #$E0          LOWER CASE?
2140        BCC .3
2150        AND #$DF          STRIP CASE
2160 .3     CMP #" "          SPACE?
2170        BEQ MENU.NEXT.SCREEN
2180        CMP #$8D          RETURN?
2190        BEQ ABORT
2200        CMP #$9B          ESCAPE?
2210        BEQ ABORT
2220        CMP #"A"
2230        BCC .2            NOT A-Z, SO IGNORE
2240        CMP MENU.LETTER
2250        BCS .2            BEYOND VALID VALUES
2260 *---GET T/S LIST-----------------
2270        AND #$1F          CONVERT LETTER TO INDEX
2280        TAY
2290        LDX SECTORS,Y
2300        LDA TRACKS,Y
2310        JSR RTS           READ TRACK/SECTOR LIST
2320        STX PNTR+1        SET POINTER
2330        LDY #0
2340 .4     LDA (PNTR),Y      MOVE T/S LIST TO ITS BUFFER
2350        STA TS.LIST,Y
2360        INY
2370        BNE .4
2380 *---GET THE FILE SIZE------------
2390        LDY #$0C          POINT AT FIRST T/S
2400        STY CAT.INDEX
2410        LDA TS.LIST,Y     TRACK
2420        BEQ ERR.EMPTY.FILE
2430        LDX TS.LIST+1,Y   SECTOR
2440        JSR RTS           READ FIRST SECTOR
2450        STX PNTR+1
2460        LDY #0
2470        LDA (PNTR),Y      GET FILE SIZE
2480        STA SIZE
2490        INY
2500        LDA (PNTR),Y
2510        STA SIZE+1
2520 *---MAKE ROOM FOR FILE-----------
2530        SEC
2540        LDA HIMEM
2550        SBC SIZE
2560        STA PP            SET ASSEMBLER'S POINTER
2570        STA LPTR+1        AND OUR LOAD POINTER
2580        LDA HIMEM+1
2590        SBC SIZE+1
2600        STA PP+1
2610        STA LPTR+2
2620        CMP LOMEM+1
2630        BCC ERR.TOO.BIG   ...TOO LOW
2640 *---LOAD FROM 1ST SECTOR---------
2650        INY               POINT AT FIRST PROGRAM BYTE
2660 .5     JSR LOAD.FROM.SECTOR
2670        BCS ABORT         ...END OF LOAD
```

```
2680 *---LOAD REST OF FILE------------
2690        LDY CAT.INDEX
2700        INY
2710        INY
2720        BEQ ABORT
2730        STY CAT.INDEX     NEXT TRACK/SECTOR
2740        LDA TS.LIST,Y     TRACK
2750        BEQ ABORT         ...END OF FILE
2760        LDX TS.LIST+1,Y   SECTOR
2770        JSR RTS           READ IT
2780        STX PNTR+1        SET POINTER
2790        LDY #0
2800        BEQ .5            ...ALWAYS
2810 *--------------------------------
2820 ABORT  JMP $8003         WARMSTART ASSEMBLER
2830 *--------------------------------
2840 MENU.NEXT.SCREEN
2850        LDA DONE.FLAG
2860        BEQ .1
2870        JMP LOAD.MENU     START ALL OVER
2880 .1     JSR SETUP.SCREEN
2890        JMP GET.NEXT.FILE
2900 *--------------------------------
2910 ERR.EMPTY.FILE
2920        LDY #EM1
2930        .HS 2C
2940 ERR.TOO.BIG
2950        LDY #EM2
2960        JSR PRINT.MSG
2970        JMP $8003
2980 *--------------------------------
2990 PRINT.MSG
3000 .1     LDA EMS,Y
3010        BEQ .2            00 IS END OF MESSAGE
3020        JSR MON.COUT
3030        INY
3040        BNE .1            ...ALWAYS
3050 .2     RTS
3060 *--------------------------------
3070 GETNUM
3080        STA LIMIT
3090        JSR PRINT.MSG     PROMPT
3100 .1     JSR MON.RDKEY
3110        CMP #"1"
3120        BCC .1            GO BACK IF TOO SMALL
3130        CMP LIMIT
3140        BCS .1            ...OR TOO LARGE
3150        JSR MON.COUT      ECHO CHARACTER
3160        EOR #"0"          EXTRACT VALUE
3170        RTS
3180 *--------------------------------
3190 READ.NEXT.CATALOG.SECTOR
3200        LDA #$0B          RESTART INDEX
3210        STA CAT.INDEX
3220        SEC               IN CASE NO MORE SECTORS
3230        LDY #2
```

```
3240        LDA (PNTR),Y
3250        TAX                 SECTOR
3260        DEY
3270        LDA (PNTR),Y        TRACK
3280        BEQ .1              END OF CATALOG
3290        JSR RTS             READ IT
3300        STX PNTR+1          PAGE IN BUFFER
3310        CLC                 SIGNAL WE GOT A SECTOR
3320 .1     RTS
3330 *--------------------------------
3340 *   READ TRACK/SECTOR
3350 *      (A)=TRACK, (X)=SECTOR
3360 *      RETURNS (X)=PAGE OF BUFFER CONTAINING SECTOR
3370 *              CARRY SET IF ERROR
3380 *      CLOBBERS (A) AND (Y)
3390 *--------------------------------
3400 RTS
3410        LDY #0
3420        ASL                 TRACK*8
3430        ASL
3440        ASL
3450        BCC .1              BLOCK < $100
3460        INY                 BLOCK > $0FF
3470 .1     ASL                 *2, MAKE ROOM FOR H/L FLAG BIT
3480        ORA BLKTBL,X
3490        ROR                 H/L BIT TO CARRY
3500        STA BLOCK
3510        STY BLOCK+1
3520        LDX /BLOCK.BUFFER
3530        BCC .2              LOWER HALF OF BLOCK
3540        INX                 UPPER HALF OF BLOCK
3550 .2     JSR $BF00
3560        .DA #$80,PARMLIST
3570        BCS .3              ...ERROR
3580        RTS
3590 .3     PHA                 SAVE ERROR CODE
3600        LDY #EM5            "ERROR"
3610        JSR PRINT.MSG
3620        PLA
3630        JSR MON.PRHEX       DISPLAY CODE
3640        JMP $8003           SOFTLY BACK TO S-C MACRO
3650 *--------------------------------
3660 SETUP.SCREEN
3670        LDA #20             LINES PER SCREEN
3680        STA LINE.COUNT
3690        LDA #"A"            START MENU WITH LETTER "A"
3700        STA MENU.LETTER
3710        JSR MON.CROUT       THREE BLANK LINES
3720        JSR MON.CROUT
3730        JMP MON.CROUT       RETURN THROUGH CROUT
3740 *--------------------------------
3750 *      RETURN .CS. IF END OF LOAD
3760 *--------------------------------
3770 LOAD.FROM.SECTOR
3780        LDA LPTR+1          IS THERE ROOM FOR
3790        CMP HIMEM           ANOTHER BYTE?
```

```
3800        LDA LPTR+2
3810        SBC HIMEM+1
3820        BCS LFS2          NO, END OF LOAD
3830        LDA (PNTR),Y
3840 LPTR   STA $5555
3850        INC LPTR+1
3860        BNE .1
3870        INC LPTR+2
3880 .1     INY
3890        BNE LOAD.FROM.SECTOR
3900 LFS2   RTS
3910 *--------------------------------
3920 EMS
3930 EM0    .EQ *-EMS
3940        .HS 8D
3950        .AS -/TYPE LETTER TO LOAD A FILE,/
3960        .HS 8D
3970        .AS -/OR <SPACE> FOR MORE FILES,/
3980        .HS 8D
3990        .AS -/OR <RET> OR <ESC> TO ABORT:  /
4000        .HS 00
4010 EM1    .EQ *-EMS
4020        .HS 8D
4030        .AS -/FILE IS EMPTY/
4040        .HS 00
4050 EM2    .EQ *-EMS
4060        .HS 8D
4070        .AS -/FILE IS TOO BIG/
4080        .HS 00
4090 EM3    .EQ *-EMS
4100        .AS -/ SLOT: /
4110        .HS 00
4120 EM4    .EQ *-EMS
4130        .HS 8D
4140        .AS -/DRIVE: /
4150        .HS 00
4160 EM5    .EQ *-EMS
4170        .HS 8D
4180        .AS -/ERROR /
4190        .HS 00
4200 *--------------------------------
4210 BLKTBL .HS 00.0E.0D.0C.0B.0A.09.08
4220        .HS 07.06.05.04.03.02.01.0F
4230 *--------------------------------
4240 PARMLIST
4250        .DA #3
4260 UNIT   .HS 60            DRIVE-1*8+SLOT*16
4270        .DA BLOCK.BUFFER
4280 BLOCK  .DA 2
4290 *--------------------------------
4300 TRACKS  .BS 21
4310 SECTORS .BS 21
```

##
Multi-Level ProDOS Catalog
##

Bob Sander-Cederlof

July 1985

Last week I looked through some old piles of papers and came across a program by Greg Seitz, dated Dec 20, 1983. It was attached to a set of ProDOS Tech Notes, and Greg apparently worked at Apple at that time.

Greg's program lists the filenames of an entire ProDOS directory, showing the whole tree. It shows directory files by printing a slash in front of the filename, and shows the level by indenting. For example, a typical listing might look like this:

```
       PRODOS
       BASIC.SYSTEM
       /UTILITIES
        HELPER
        DOER
        /MORE
         WHATEVER
         AND.ANOTHER
        TEXT.FILE
       ANOTHER
```

A listing like this can be a big help in finding things on a large hard disk. The program can also be extended in many ways. One that comes to mind immediately is to print the rest of the CATALOG information as well as the file names. Another is to create a complete CATALOG MANAGER utility, which would permit re-arranging the filenames, promoting and demoting files, and so on.

I typed in Greg's program, and then I rewrote it. The listing that follows bears very little resemblance to his code, but I do thank him for the help in getting started.

The program assumes a prefix has been set. If there is no prefix, you will get a beep and no listing. If there is a prefix, and the directory named is online, the listing will begin with that directory. Another enhancement would be to display the current prefix, and allow accepting it or changing it before starting the filename listing.

If we were always starting with the volume directory, it would be a little easier. The volume directory always starts in block 2. However, since we are able to start with any directory, we do not know where it starts. ProDOS allows you to read a directory, and we can get the first block of any directory by using MLI to open the directory file.

Lines 1100-1120 read the current prefix into a buffer. The lines 1130-1150 open that file. Although I have never seen it in the books, apparently OPEN also reads the first block. After the OPEN call, BUFFER.ONE contains the first block of the directory file. Unless we are willing to do a complete search without ProDOS's help, this is the only way I know of to find the first block of a directory file (other than the volume directory).

Since the only reason to OPEN the directory file was to read the first block, lines 1180-1200 close it again. If any of these MLI calls don't go through, line 1210 will ring the alarm and stop.

Lines 1230-1260 start up the directory listing. The first block ONLY will be in BUFFER.ONE. All subsequent blocks will be read into BUFFER.TWO. In order to make the LIST.DIRECTORY program completely recursive, it is called with the buffer address in a zero-page pointer. SETUP.NEXT.BLOCK also gets the next block pointer from the buffer and saves it in NEXT.BLOCK.

LIST.DIRECTORY is really quite simple, in spite of its size. Its main function is to print a list of filenames. Each filename is preceded by a number of blanks, determined by NEST.LEVEL. NEST.LEVEL is incremented at line 1290, each time LIST.DIRECTORY is called. If a file listed happens to be a directory file, LIST.IDRECTORY saves all the pointers and counters on the stack and then calls itself. When the subdirectory's files have all been listed, that recursive call of LIST.DIRECTORY will return, the pointers and counters can be unstacked, and the listing can continue.

The format of the information in a directory is detailed quite well in both "Beneath Apple ProDOS" and "Apple ProDOS Advanced Features". (We recommend and sell both books.) The first four bytes of each block are two block numbers: that of the previous block, and that of the next block, in the same directory. This allows scanning in both forward and reverse directions through a directory. We will only use the next-block pointers in our program. After the block numbers there are 13 descriptors of 39 bytes each. The first descriptor in a directory describes the directory itself, and the rest describe files.

For some reason Apple was not quite sure that it would always use 13 39-byte descriptors, so they stored these two numbers in the directory descriptor. Anyone who access a directory is supposed to look up these two numbers and use them, just in case Apple decides to change them someday. The directory descriptor also contains an active file count. When a file is deleted this count is decremented, but the file descriptor remains. We use the active file count to determine when we reach the end of a directory. Lines 1300-1360 pick up the bytes per descriptor, descriptors per block, and active file count and save them.

Lines 1370-1450 set up PNTR to point at the first file descriptor, which follows the directory header. CURRENT.ENTRY.NUMBER will count up to 13, so we will know when it is time to read another block. We start at 2, because the first block uses the first descriptor for the header. We also clear the file count.

Lines 1460-1500 check for the special case of an empty directory. If there are no active files, we are finished.

Lines 1510-1750 print out the file name from the current file descriptor. The first byte of a descriptor contains a code for the type of file in the first nybble, and the length of the file name in the second nybble. If both are zero, the file has been deleted. The other legal values are $1x, $2x, and $3x to signify a seedling, sapling, or tree file, respectively; and $Dx to signify a directory file. All we care about is whether is a directory file or not, and how long the file name is.

If it is a directory file, lines 1760-2100 will be executed. Lines 1760-1860 push the counters and pointers on the stack. Lines 1870-1930 read in the first block of the sub-directory. Line 1950 calls LIST.DIRECTORY to list the subdirectory. After it is finished, line 1960 will decrement the nesting level. Lines 1970-2060 unstack the pointers and counters. If we were still in the first block of the highest level directory (where we started), we do not need to read the block again: it is still in BUFFER.ONE. Otherwise, lines 2070-2100 read the block back in. If we did not care how much memory we used, we could make this program a lot faster by using more buffers. We could have a different buffer for each level, so that blocks would never have to be re-read.

Lines 2110-2210 count the file just listed, and then check to see if our count is the same as the active file count from the directory header. If so, we are finished.

If we are not finished, lines 2220-2290 bump the pointer into the directory block by the size of a descriptor entry. If we are still in the same block, that is all that we need to do. If not, lines 2350-2420 read in the next block and set things up for it. Then it's back to the top again for the next file name!

We hope some time in the not-so-distant future to be able to write a complete catalog manager program like I started to describe back at the beginning of this article. Some of you are using Bill Morgan's CATALOG ARRANGER for DOS 3.3, and this would be an equivalent utility for ProDOS. We're not quite ready yet, but this program is a step in the right direction.

```
1000 *SAVE S.RECURCAT
1010 *--------------------------------
1020 MLI    .EQ $BF00
1030 DEVNUM .EQ $BF30
1040 BELL   .EQ $FBDD
1050 CROUT  .EQ $FD8E
1060 COUT   .EQ $FDED
1070 PNTR   .EQ $EB AND EC
1080 *--------------------------------
1090 CAT
1100        JSR MLI           GET CURRENT PREFIX
1110        .DA #$C7,P.PREFIX
1120        BCS .1            ...ERROR
1130        JSR MLI           OPEN THE DIRECTORY
1140        .DA #$C8,P.OPEN     AND READ FIRST BLOCK
1150        BCS .1            ...ERROR
1160        LDA DEVNUM        SET UP READ MLI BLOCK
1170        STA R.DEVNUM
1180        JSR MLI           CLOSE THE DIRECTORY
1190        .DA #$CC,P.CLOSE
1200        BCC .2            ...NO ERROR
1210 .1     JSR BELL          INDICATE ERROR
1220        RTS
1230 .2     LDA #0       BUFFERS ON PAGE BOUNDARIES
1240        STA NEST.LEVEL    START AT TOP LEVEL
1250        LDY /BUFFER.ONE   POINT TO NEXT BLOCK
1260        JSR SETUP.NEXT.BLOCK
1270 *--------------------------------
1280 LIST.DIRECTORY
1290        INC NEST.LEVEL    DROP TO NEXT LEVEL
1300 *---GET DIR DATA-----------------
1310        LDY #38
1320 .1     LDA (PNTR),Y           GET: BYTES.PER.ENTRY....35
1330        STA BYTES.PER.ENTRY-35,Y    ENTRIES.PER.BLOCK..36
1340        DEY                         FILE.COUNT......37,38
```

```
1350        CPY #35
1360        BCS .1
1370 *---POINT TO FIRST FILE NAME-----
1380        LDA #2            SKIP OVER DIR HEADER
1390        STA CURRENT.ENTRY.NUMBER
1400        ASL               A=4, CLEAR CARRY
1410        ADC BYTES.PER.ENTRY
1420        STA PNTR          POINT AT FIRST NAME
1430        LDA #0            START FILE COUNT
1440        STA CURRENT.FILE.COUNT
1450        STA CURRENT.FILE.COUNT+1
1460 *---STOP IF NO ACTIVE FILES------
1470        LDA ACTIVE.FILE.COUNT
1480        ORA ACTIVE.FILE.COUNT+1
1490        BNE .2       ...AT LEAST ONE FILE
1500        RTS          ...END OF DIRECTORY
1510 *---PRINT FILE NAME--------------
1520 .2     LDY #0            POINT TO TYPE/LENGTH
1530        LDA (PNTR),Y
1540        BEQ .8            0 = DELETED FILE
1550        AND #$0F          ISOLATE NAME LENGTH
1560        TAX               X = #CHARS IN NAME
1570        LDY NEST.LEVEL    NUMBER OF LEADING BLANKS
1580        LDA #" "
1590 .3     JSR COUT          INDENT BY DIRECTORY LEVEL
1600        DEY
1610        BNE .3
1620        LDA (PNTR),Y      GET TYPE/LENGTH
1630        PHA               1L, 2L, 3L, OR DL
1640        BPL .4            ...NOT DIR FILE
1650        LDA #"/"          DIR FILE, PRINT A SLASH
1660        JSR COUT
1670 .4     INY               PRINT THE FILE'S NAME
1680        LDA (PNTR),Y
1690        ORA #$80
1700        JSR COUT
1710        DEX
1720        BNE .4
1730        JSR CROUT
1740        PLA               GET TYPE/LENGTH AGAIN
1750        BPL .7            ...NOT DIR FILE
1760 *---PUSH DATA ON STACK-----------
1770        LDA PNTR+1        SAVE POINTER IN CURRENT BLOCK
1780        PHA
1790        LDA PNTR
1800        PHA          SAVE:  R.BLOCK
1810        LDX #0              BYTES.PER.ENTRY
1820 .5     LDA PUSH.VARS,X     ENTRIES.PER.BLOCK
1830        PHA                 ACTIVE.FILE.COUNT
1840        INX                 CURRENT.FILE.COUNT
1850        CPX #PUSH.COUNT     CURRENT.ENTRY.NUMBER
1860        BNE .5              NEXT.BLOCK
1870 *---READ HEADER OF SUBDIR--------
1880        LDY #$12          POINT AT KEYBLOCK POINTER
1890        LDA (PNTR),Y      GET HIGH BYTE
1900        TAX
```

```
1910        DEY
1920        LDA (PNTR),Y      GET LOW BYTE
1930        JSR READ.NEXT.BLOCK
1940 *---RECURSIVE CALL---------------
1950        JSR LIST.DIRECTORY
1960        DEC NEST.LEVEL    POP TO HIGHER LEVEL
1970 *---POP DATA OFF STACK-----------
1980        LDX #PUSH.COUNT   GET BLOCK OF VARS
1990 .6     PLA
2000        STA PUSH.VARS-1,X
2010        DEX
2020        BNE .6
2030        PLA
2040        STA PNTR          GET KEYBLOCK POINTER
2050        PLA
2060        STA PNTR+1
2070        CMP /BUFFER.TWO   IS BLOCK IN BUFFER.TWO?
2080        BCC .7            ...NO, DON'T NEED TO READ
2090        JSR MLI           ...YES, MUST READ THE BLOCK
2100        .DA #$80,P.READ
2110 *---COUNT THE FILE---------------
2120 .7     INC CURRENT.FILE.COUNT
2130        BNE .8
2140        INC CURRENT.FILE.COUNT+1
2150 *---SEE IF THAT WAS LAST FILE----
2160 .8     LDA CURRENT.FILE.COUNT
2170        CMP ACTIVE.FILE.COUNT
2180        LDA CURRENT.FILE.COUNT+1
2190        SBC ACTIVE.FILE.COUNT+1
2200        BCC .9            ...NOT LAST FILE
2210        RTS               ...END OF DIRECTORY
2220 *---ADVANCE PNTR TO NEXT ENTRY---
2230 .9     CLC
2240        LDA PNTR          GET RESULT IN Y,X
2250        ADC BYTES.PER.ENTRY
2260        TAX
2270        LDA PNTR+1
2280        ADC #0
2290        TAY
2300 *---ARE WE STILL INSIDE BLOCK?---
2310        LDA CURRENT.ENTRY.NUMBER
2320        INC CURRENT.ENTRY.NUMBER
2330        CMP ENTRIES.PER.BLOCK
2340        BCC .10           ...INSIDE SAME BLOCK
2350 *---READ NEXT BLOCK--------------
2360        LDA NEXT.BLOCK
2370        LDX NEXT.BLOCK+1
2380        JSR READ.NEXT.BLOCK
2390        LDA #1            START WITH FIRST ENTRY
2400        STA CURRENT.ENTRY.NUMBER    IN NEW BLOCK
2410        LDX #4            SKIP OVER BLOCK NUMBERS
2420        LDY /BUFFER.TWO
2430 .10    STX PNTR          NEW PNTR VALUE
2440        STY PNTR+1
2450        JMP .2            ...TO LIST NEXT FILENAME
2460 *--------------------------------
```

```
2470 READ.NEXT.BLOCK
2480        STA R.BLOCK       BLOCK # IN X,A
2490        STX R.BLOCK+1
2500        JSR MLI           READ THE BLOCK
2510        .DA #$80,P.READ
2520        LDA #BUFFER.TWO   WE USED BUFFER.TWO
2530        LDY /BUFFER.TWO
2540 SETUP.NEXT.BLOCK
2550        STA PNTR          PNTR FROM Y,A
2560        STY PNTR+1
2570        LDY #2            GET NEXT BLOCK #
2580        LDA (PNTR),Y
2590        STA NEXT.BLOCK
2600        INY
2610        LDA (PNTR),Y
2620        STA NEXT.BLOCK+1
2630        RTS               RETURN
2640 *--------------------------------
2650 P.PREFIX   .DA #1
2660            .DA BUFFER.TWO
2670 *--------------------------------
2680 P.OPEN     .DA #3
2690            .DA BUFFER.TWO
2700 OPENBUF    .DA BUFFER.ONE
2710            .DA #0
2720 *--------------------------------
2730 P.CLOSE    .DA #1
2740            .DA #0
2750 *--------------------------------
2760 P.READ     .DA #3
2770 R.DEVNUM   .DA #$60
2780            .DA BUFFER.TWO
2790 PUSH.VARS .EQ *
2800 R.BLOCK    .DA 0
2810 *--------------------------------
2820 BYTES.PER.ENTRY          .BS 1
2830 ENTRIES.PER.BLOCK        .BS 1
2840 ACTIVE.FILE.COUNT        .BS 2
2850 CURRENT.FILE.COUNT       .BS 2
2860 CURRENT.ENTRY.NUMBER     .BS 1
2870 NEXT.BLOCK               .BS 2
2880 PUSH.COUNT               .EQ *-PUSH.VARS
2890 *--------------------------------
2900 NEST.LEVEL               .BS 1
2910 *--------------------------------
2920 WASTED .EQ *+255/256*256-*
2930        .BS WASTED
2940 *--------------------------------
2950 BUFFER.ONE  .BS 512
2960 BUFFER.TWO  .BS 512
2970 *--------------------------------
```

##
Put DOS and ProDOS Files on Same Disk
##

Bob Sander-Cederlof

September 1985

In the February 1985 issue of AAL I showed how to create a DOS-less DOS 3.3 data disk. Tracks 1 and 2, normally full of the DOS image, were instead made available for files. Booting the disk gets you a message that such a disk cannot be booted.

Now that we are publishing more and more programs intended for use under ProDOS, we foresee the need to publish Quarterly Disks that contain both DOS and ProDOS programs. Believe it or not, this is really possible.

The DOS operating system keeps its Volume Table of Contents (VTOC) and catalog in track $11. The VTOC is in sector 0 of that track, and the catalog normally fills the rest of the track. A major part of the VTOC is the bit map, which shows which sectors are as yet unused by any files. If we want to reserve some sectors for use by a ProDOS directory on the same disk, we merely mark those sectors as already being in use in the DOS bit map.

ProDOS keeps its directory and bit map all in track 0. This track is not available to DOS for file storage anyway, so we can be comfortable stealing it for a ProDOS setup on the same diskette.

I decided to keep things fairly simple, by splitting the disk into two parts purely on a track basis. ProDOS gets some number of tracks starting with track 0, and DOS gets all the tracks from just after ProDOS to track 34. If ProDOS gets more than 17 tracks, it will hop over track $11 (since DOS's catalog is there). Normally I will split the disk in half, giving tracks 0-16 to ProDOS and tracks 17-34 to DOS. With this arrangement, ProDOS thus starts with 129 free blocks, and DOS starts with 272 free sectors.

The program I wrote does not interact with the user; instead, you set all the options by changing the source code and re-assembling. It would be nice to have an interactive front end to get slot, drive, volume number for the DOS half, volume name for the ProDOS half, and how many tracks to put in each half. Maybe we'll add this stuff later, or maybe you would like to try your hand at it.

The parameters you might want to change are found in lines 1020-1050. You can see that I started the DOS allocation at track $12, just after the catalog track. I also chose volume 1, drive 1, slot 6. You can use any volume number from 1 to 254. Since these numbers were under my control, I did not bother to check for legal values. If we add an interactive front end, we will have to validate them. We might also want to display the number of ProDOS blocks and DOS sectors that result from the DOS.LOW.TRACK selection, maybe in a graphic format. You might even use a joystick or mouse....

You might also want to change the ProDOS volume name. I am calling it "DATA". The name is in line 2850. It can be up to 15 characters long, and the number of characters must be stored in the right nybble of the byte just before the name. This is automatically inserted for you, by the assembler. If you should try to assemble a name larger than 15 characters, line 2870 will cause a RANGE ERROR. Another way of changing the ProDOS volume name is to do so after initialization using the ProDOS FILER program.

Lines 1090 and 1100 compute the number of free DOS sectors and ProDOS blocks. The values are not used anywhere in the program, but are nice to know.

Line 1300 sets the program origin at $803. Why $803, and not $800? If we load and run an assembly language program at $800, and then later try to load and run an Applesoft program, Applesoft can get confused. Applesoft requires that $800 contain a $00 value, but it does not make sure it happens when you LOAD an Applesoft program from the disk. By putting our program at $803 we make sure we don't kill the $00 and $800. Well, then why not start at $801? I don't know, we just always did it that way. (It would make good sense if our program started by putting $00 in $801 and $802, indicating to Applesoft that it had no program in memory.)

DOUBLE.INIT is written to run under DOS 3.3, and makes calls on the RWTS subroutine to format and write information on the disk. The entire DOUBLE.INIT program is driven by lines 1320-1490. The flow is very straightforward:

1. Format the disk as 35 empty tracks.
2. Write DOS VTOC and Catalog in track 17.
3. Write ProDOS Directory and bit map in track 0.
4. Write "YOU CANNOT BOOT" code in boot sector.

Formatting a blank disk is simple, unless you have a modified DOS with the INIT capability removed. Lines 1510-1590 set up a format call to RWTS, and fall into my RWTS caller.

Lines 1600-1800 call RWTS and return, unless there was an error condition. If there was an error, I will print out "RWTS ERROR" and the error code in hex. The error code values you might see are:

$08 -- Error during formatting
$10 -- Trying to write on write protected disk
$40 -- Drive error

I don't think you can get $20 (volume mismatch) or $80 (read error) from DOUBLE.INIT. After printing the error message, DOS will be warm started, aborting DOUBLE.INIT.

Building the DOS VTOC and Catalog is handled by lines 1820- 2310. The beginning section of the VTOC contains information about the number of tracks and sectors, where to find the catalog, etc. This is all assembled in at lines 2260-2310, and is copied into my buffer by lines 1880-1930. Since the volume number is a parameter, I specially load it in with lines 1940 and 1950. The rest of the VTOC is a bit map showing which sectors are not yet used. Lines 1960-2090 build this bit map. Lines 1840-1870 and 2100-2120 cause the VTOC image to be written on track 17 ($11) sector 0.

There are some unused bytes in the beginning part of the VTOC, so I decided to put some private information in there. See line 2270 and line 2290.

The rest of track 17 is a series of empty linked sectors comprising the catalog. The chain starts with sector $0F, and works backward to sector 1. Lines 2130-2240 build each sector in turn and write it on the disk.

The ProDOS directory and bit map are installed in track 0 by lines 2330-2900. This gets a little tricky, because we are trying to write ProDOS blocks with DOS 3.3 RWTS. Here is a correspondence table, showing the blocks and sectors in track 0:

ProDOS Block:	0	1	2	3	4	5	6	7
DOS 3.3 Sectors:	0,E	D,C	B,A	9,8	7,6	5,4	3,2	F,1

The first sector of each pair contains the first part of each block, and so on.

The ProDOS bit map goes in block 6, which is sectors 3 and 2. Even if we had an entire diskette allocated to ProDOS the bit map would occupy very little of the first of these two sectors. Since formatting the disk wrote 256 zeroes into every sector, we can leave sector 2 unchanged. Lines 2700-2820 build the bit map data for sector 3 and write it out. Note that block 7 is available, all blocks in track 17 are unavailable.

The ProDOS Directory starts in block 2. The first two bytes of a directory sector point to the previous block in the directory chain, and the next two bytes point to the following block in the chain. We follow the standard ProDOS convention of linking blocks 3, 4, and 5 into the directory. Those three blocks contain no other information, since there are as yet no filenames in the directory. Here's how the chain links together:

	Previous Block	Next Block	
Block 2:	0	3	(zero means the beginning)
Block 3:	2	4	
Block 4:	3	5	
Block 5:	4	0	(zero means the end)

Block 2 gets some extra information, the volume header. Lines 2840-2900 contain the header data, which is copied into my buffer by lines 2590-2630.

The no-booting boot program is shown in lines 3000-3190. This is coded as a .PHase at $800 (see lines 3010 and 3190), since the disk controller boot ROM will load it at that address. All the program does is turn off the disk motor and print out a little message. Lines 1410-1490 write this program on track 0 sector 0.

I think if you really wanted to you could put a copy of the ProDOS boot program in block 0 (sectors 0 and E). Then if you copied the file named PRODOS into the ProDOS half of the disk, you could boot ProDOS.

There is one thing to look out for if you start cranking out DOUBLE DISKS. There are some utility programs in existence which are designed to "correct" the DOS bitmap in the VTOC sector. Since these programs have never heard of ProDOS, let alone of DOUBLE DISKS, they are going to tell DOS that all those tracks we carefully gave to ProDOS belong to DOS. If you let that happen to a disk on which you have already stored some ProDOS files, zzzaaaapppp!

```
1000 *SAVE S.INIT DOS & PRODOS
1010 *--------------------------------
1020 DOS.LOW.TRACK .EQ $12    DOS $12...$22
1030 DOS.VOLUME    .EQ 1
1040 SLOT          .EQ 6
1050 DRIVE         .EQ 1
1060 *--------------------------------
```

```
1070 PRODOS.MAX.BLOCKS .EQ DOS.LOW.TRACK*8
1080 *--------------------------------
1090 ACTUAL.DOS.SECTORS   .EQ DOS.LOW.TRACK>$11+34-DOS.LOW.TRACK*16
1100 ACTUAL.PRODOS.BLOCKS .EQ DOS.LOW.TRACK<$12+DOS.LOW.TRACK-2*8+1
1110 *--------------------------------
1120 DOS.WARM.START .EQ $03D0
1130 RWTS        .EQ $03D9
1140 GETIOB      .EQ $03E3
1150 *--------------------------------
1160 R.PARMS     .EQ $B7E8
1170 R.SLOT16    .EQ $B7E9
1180 R.DRIVE     .EQ $B7EA
1190 R.VOLUME    .EQ $B7EB
1200 R.TRACK     .EQ $B7EC
1210 R.SECTOR    .EQ $B7ED
1220 R.BUFFER    .EQ $B7F0,B7F1
1230 R.OPCODE    .EQ $B7F4
1240 R.ERROR     .EQ $B7F5
1250 *--------------------------------
1260 MON.CROUT   .EQ $FD8E
1270 MON.PRBYTE  .EQ $FDDA
1280 MON.COUT    .EQ $FDED
1290 *--------------------------------
1300        .OR $803
1310 *--------------------------------
1320 DOUBLE.INIT
1330        JSR FORMAT.35.TRACKS
1340        LDA #INIT.BUFFER
1350        STA R.BUFFER
1360        LDA /INIT.BUFFER
1370        STA R.BUFFER+1
1380        JSR BUILD.DOS.CATALOG
1390        JSR BUILD.PRODOS.CATALOG
1400 *---WRITE BOOT PROGRAM-----------
1410        LDA #BOOTER
1420        STA R.BUFFER
1430        LDA /BOOTER
1440        STA R.BUFFER+1
1450        JSR CLEAR.INIT.BUFFER
1460        LDA #0
1470        STA R.TRACK
1480        STA R.SECTOR
1490        JMP CALL.RWTS
1500 *--------------------------------
1510 FORMAT.35.TRACKS
1520        LDA #SLOT*16
1530        STA R.SLOT16
1540        LDA #DRIVE
1550        STA R.DRIVE
1560        LDA #DOS.VOLUME
1570        STA R.VOLUME
1580        STA V.VOLUME
1590        LDA #$04      INIT OPCODE FOR RWTS
1600 CALL.RWTS.OP.IN.A
1610        STA R.OPCODE
1620 CALL.RWTS
```

```
1630        JSR GETIOB
1640        JSR RWTS
1650        BCS .1       ERROR
1660        RTS
1670 .1     LDY #0       PRINT "ERROR"
1680 .2     LDA ERMSG,Y
1690        BEQ .3
1700        JSR MON.COUT
1710        INY
1720        BNE .2       ...ALWAYS
1730 .3     LDA R.ERROR  GET ERROR CODE
1740        JSR MON.PRBYTE
1750        JSR MON.CROUT
1760        JMP DOS.WARM.START
1770 *--------------------------------
1780 ERMSG  .HS 8D87
1790        .AS -/RWTS ERROR /
1800        .HS 00
1810 *--------------------------------
1820 BUILD.DOS.CATALOG
1830        JSR CLEAR.INIT.BUFFER
1840        LDA #17
1850        STA R.TRACK
1860        LDA #0
1870        STA R.SECTOR
1880 *---BUILD GENERIC VTOC-----------
1890        LDY #VTOC.SZ-1
1900 .0     LDA VTOC,Y
1910        STA INIT.BUFFER,Y
1920        DEY
1930        BPL .0
1940        LDA #DOS.VOLUME
1950        STA V.VOLUME
1960 *---PREPARE BITMAP---------------
1970        LDY #4*34
1980        LDA #$FF
1990 .1     CPY #4*17    ARE WE ON CATALOG TRACK?
2000        BEQ .2
2010        CPY #4*DOS.LOW.TRACK
2020        BCC .3            IN PRODOS ARENA
2030        STA V.BITMAP+1,Y
2040        STA V.BITMAP,Y
2050 .2     DEY
2060        DEY
2070        DEY
2080        DEY
2090        BNE .1
2100 *---WRITE VTOC ON NEW DISK-------
2110 .3     LDA #2            RWTS WRITE OPCODE
2120        JSR CALL.RWTS.OP.IN.A
2130 *---WRITE CATALOG CHAIN----------
2140        JSR CLEAR.INIT.BUFFER
2150        LDA #17      TRACK 17
2160        LDY #15      START IN SECTOR 15
2170        STA C.TRACK
2180 .4     STY R.SECTOR
```

```
2190        DEY
2200        STY C.SECTOR
2202        BNE .5
2203        STY C.TRACK  TERMINATE THE CHAIN
2210 .5     JSR CALL.RWTS
2220        LDY C.SECTOR
2230        BNE .4
2240        RTS
2250 *--------------------------------
2260 VTOC   .HS 04.11.0F.03.00.00.01
2270        .AS "COMBINATION DOS/PRODOS DATA DISK"
2280        .HS 7A
2290        .AS /07-25-85/
2300        .HS 11.01.00.00.23.10.00.01
2310 VTOC.SZ .EQ *-VTOC
2320 *--------------------------------
2330 BUILD.PRODOS.CATALOG
2340        LDA #0
2350        STA R.TRACK
2360        JSR CLEAR.INIT.BUFFER
2370 *--------------------------------
2380        LDA #5            SECTOR 5 = BLOCK 5
2390        STA R.SECTOR      BACK LINK = 0004
2400        LDA #4             FWD LINK = 0000
2410        STA INIT.BUFFER
2420        JSR CALL.RWTS
2430 *--------------------------------
2440        LDA #7            SECTOR 7 = BLOCK 4
2450        STA R.SECTOR      BACK LINK = 0003
2460        DEC INIT.BUFFER    FWD LINK = 0005
2470        LDA #5
2480        STA INIT.BUFFER+2
2490        JSR CALL.RWTS
2500 *--------------------------------
2510        LDA #9            SECTOR 9 = BLOCK 3
2520        STA R.SECTOR      BACK LINK = 0002
2530        DEC INIT.BUFFER    FWD LINK = 0004
2540        DEC INIT.BUFFER+2
2550        JSR CALL.RWTS
2560 *--------------------------------
2570        LDA #11           SECTOR 11 = BLOCK 2
2580        STA R.SECTOR      BACK LINK = 0000
2590        LDY #HDR.SZ-1      FWD LINK = 0003
2600 .1     LDA HEADER,Y
2610        STA INIT.BUFFER,Y GET VOLUME HEADER
2620        DEY
2630        BPL .1
2640        LDA #PRODOS.MAX.BLOCKS
2650        STA INIT.BUFFER+$29
2660        LDA /PRODOS.MAX.BLOCKS
2670        STA INIT.BUFFER+$2A
2680        JSR CALL.RWTS
2690 *--------------------------------
2700        LDA #3
2710        STA R.SECTOR
2720        JSR CLEAR.INIT.BUFFER
```

```
2730        LDA #$FF
2740        LDY #DOS.LOW.TRACK-1
2750 .2     CPY #17      SKIP OVER DOS CATALOG TRACK
2760        BEQ .3
2770        STA INIT.BUFFER,Y
2780 .3     DEY
2790        BPL .2
2800        LDA #1       MAKE ONLY BLOCK 7 AVAILABLE
2810        STA INIT.BUFFER   IN TRACK 0
2820        JMP CALL.RWTS
2830 *---------------------------------
2840 HEADER .DA 0,3,#$F0+VNSZ
2850 VN     .AS /DATA/
2860 VNSZ   .EQ *-VN
2870        .BS 15-VNSZ
2880        .HS 00.00.00.00.00.00.00.00.00.00.00.00
2890        .HS 00.00.C3.27.0D.00.00.06.00.08.00
2900 HDR.SZ .EQ *-HEADER
2910 *---------------------------------
2920 CLEAR.INIT.BUFFER
2930        LDY #0
2940        TYA
2950 .1     STA INIT.BUFFER,Y
2960        INY
2970        BNE .1
2980        RTS
2990 *---------------------------------
3000 BOOTER
3010        .PH $800
3020 BOOTER.PHASE
3030        .HS 01
3040        LDA $C088,X  MOTOR OFF
3050        LDY #0
3060 .1     LDA MESSAGE,Y
3070        BEQ .2
3080        JSR $FDF0
3090        INY
3100        BNE .1
3110 .2     JMP $FF59
3120 *---------------------------------
3130 MESSAGE
3140        .HS 8D8D8787
3150        .AS -"COMBINATION DOS/PRODOS DATA DISK"
3160        .HS 8D8D8787
3170        .AS -/NO DOS IMAGE ON THIS DISK/
3180        .HS 8D8D00
3190        .EP
3200 *---------------------------------
3210 INIT.BUFFER .BS 256
3220 *---------------------------------
3230 V.VOLUME   .EQ INIT.BUFFER-$BB+$C1
3240 V.BITMAP   .EQ INIT.BUFFER-$BB+$F3
3250 *---------------------------------
3260 C.TRACK        .EQ INIT.BUFFER+1
3270 C.SECTOR       .EQ INIT.BUFFER+2
3280 *---------------------------------
```

```
################################################################################
#  ProDOS Snooper
################################################################################
```

Bob Sander-Cederlof

October 1985

This past week I have been working on a project which involved creating a new device driver for a disk-like device. In the process of debugging my driver, I had to write a "snooper" program.

By "snooper", I mean a program which will make a list of all calls to the driver, recording the origin of the call and the parameters of the call.

ProDOS keeps a table of the addresses of the device drivers assigned to each slot and drive between $BF10 and $BF2F. There are two bytes for each slot and drive. $BF10-1F is for drive 1, and $BF20-2F is for drive 2. For example, the address of the device driver for slot 6 drive 1 is at $BF1C,1D. (Normally this address is $D000.)

I have a Sider drive in slot 7. The device driver address for the Sider is $C753, and is kept at $BF1E,1F and $BF2E,2F.

By patching the device driver address to point to my own code, I can get control whenever ProDOS tries to read or write or whatever. If I save and restore all the registers, and jump to the REAL device driver after I am finished, ProDOS will never be the wiser. But I will!

While my program has control, I can capture all the information I am interested in. Unfortunately I cannot print it out at this time, because if I try to ProDOS will get stuck in a loop. Instead I will save the data in a buffer so I can look at it later.

The program which follows has three distinct parts. Lines 1140-1290 are an installation and removal tool. If the program has just been BLOADed or LOADed and ASMed, running INSTALL.SNOOPER will (you guessed it!) install the snooper. The actual device driver address for the slot (which you specified in line 1060 before assembling the program) will be saved in my two-byte variable DRIVER. The previous contents of DRIVER, which is the address of my snoop routine, will be copied into ProDOS's table. The value of DRIVES, which you specified before assembling the program at line 1070, will determine whether SNOOPER is connected to drive 2 or not. It will always be connected to drive 1.

If SNOOPER has already been installed, running INSTALL.SNOOPER will reverse the installation process, returning ProDOS to its original state. INSTALL.SNOOPER also resets the buffer I use to keep the captured information. To make it easy to run INSTALL.SNOOPER, I put a JMP to it at $300. After assembly you can type "$300G" to install the snooper, and type the same again to dis-install it.

The JMP at $303 (line 1120) goes to the display program. After SNOOPER has been installed, all disk accesses on the installed slot will cause information to be accumulated in BUFFER. Typing "$303G" will cause the contents of BUFFER to be displayed in an easy-to-read format.

I set up SNOOPER to capture eight bytes of information each time it is activated. You might decide to save more or less. I save the return address from the stack, to get some idea of which routine inside ProDOS is trying to access the disk. I also save the six bytes at $42-47, which are the calling parameters for the device driver. Page 6-8 of Beneath Apple ProDOS describes these parameters; you can also find out about them in Apple's ProDOS Technical Reference Manual and in Gary Little's "Apple ProDOS--Advanced Features".

$42 contains the command code: 00=status, 01=read, 02=write, and 03=format. $43 contains the unit number, in the format DSSS0000 (where SSS=slot and D=0 for drive 1, D=1 for drive 2). $44-45 contain the address of the memory buffer, lo-byte first; the buffer is 512 bytes long. $46-47 contain the block number to be read or written.

My DISPLAY program displays each group of eight bytes on a separate line, in the following format:

```
hhll:cc.uu.buff.blok
```

where hhll is the return address from the stack, hi-byte first; cc is the command code; uu is the unit number; buff is the buffer address, hi-byte first; blok is the block number, hi-byte first.

If you get into figuring out more of what ProDOS is doing, you might want to save more information from the stack. You can look behind the immediate return address to get more return addresses and other data which have been saved on the stack before calling the device driver.

A word of explanation about lines 1040, 1360, 1370, 1490, and 1500. Line 1040 tells the S-C Macro Assembler that it is OK to assemble opcodes legal in the 65C02. The PHX, PHY, PLX and PLY opcodes are in the 65C02, 65802, and 65816; however, they are not in the 6502. If you have only the 6502 in your Apple, you will need to substitute the longer code shown in the comments. Leave out line 1040, and use the following:

```
1360    TYA
1365    PHA
1370    TXA
1375    PHA
.
.
.
1490    PLA
1495    TAX
1500    PLA
1505    TAY
```

In the process of "snooping" I was able to debug my new device drivers for the project I was developing. I also discovered what appear to be some gross in-efficiencies in ProDOS. In the course of even simple CATALOGs, LOADs, and SAVEs the same blocks are read into the same buffers over and over, at times when it would appear to be totally unnecessary. If there was some mechanism inside MLI to keep track of the fact that a complete un-spoiled copy of a particular block was already in RAM, it could save a lot of time. On the other hand, it could be that the current approach is safer. I think it is a potentially fruitful area for further investigation. Any takers?

```
1010 *SAVE PRODOS.SNOOPER
1020 *--------------------------------
1030        .OR $300
1040        .OP 65C02    (If you have one)
```

```
1050 *--------------------------------
1060 SLOT   .EQ 6
1070 DRIVES .EQ 2
1080 *--------------------------------
1090 BUFFER .EQ $800
1100 *--------------------------------
1110 A300   JMP INSTALL.SNOOPER
1120 A303   JMP DISPLAY
1130 *--------------------------------
1140 INSTALL.SNOOPER
1150        LDX #1
1160 .1     LDA 2*SLOT+$BF10,X
1170        PHA          SAVE CURRENT DRIVER ADDRESS
1180        LDA DRIVER,X INSTALL NEW DRIVER ADDRESS
1190        STA 2*SLOT+$BF10,X
1200   .DO DRIVES=2
1210        STA 2*SLOT+$BF20,X
1220   .FIN
1230        PLA          REMEMBER OLD DRIVER
1240        STA DRIVER,X
1250        LDA BUFFER.ADDR,X
1260        STA A+1,X
1270        DEX
1280        BPL .1       NOW THE OTHER BYTE
1290        RTS
1300 *--------------------------------
1310 DRIVER      .DA SNOOPER
1320 BUFFER.ADDR .DA BUFFER
1330 *--------------------------------
1340 SNOOPER
1350        PHA
1360        PHY      (If no 65C02 use TYA, PHA)
1370        PHX      (If no 65C02 use TXA, PHA)
1380        TSX
1390        LDA $104,X   LO-BYTE OF RETURN ADDR
1400        JSR STORE.BYTE
1410        LDA $105,X   HI-BYTE OF RETURN ADDR
1420        JSR STORE.BYTE
1430        LDX #0       $42...47
1440 .1     LDA $42,X    WHICH ARE THE PARAMETERS
1450        JSR STORE.BYTE     FOR THE CALL
1460        INX
1470        CPX #6
1480        BCC .1
1490        PLX      (If no 65C02 use PLA, TAX)
1500        PLY      (If no 65C02 use PLA, TAY)
1510        PLA
1520        JMP (DRIVER) CONTINUE IN DRIVER
1530 *--------------------------------
1540 STORE.BYTE
1550 A      STA BUFFER   THIS ADDRESS IS MODIFIED
1560        INC A+1      BUMP PNTR TO NEXT ADDRESS
1570        BNE .1
1580        INC A+2
1590 .1     RTS
1600 *--------------------------------
```

```
1610 COUT   .EQ $FDED
1620 CROUT  .EQ $FD8E
1630 PRBYTE .EQ $FDDA
1640 PNTR   .EQ $00,01
1650 *--------------------------------
1660 DISPLAY
1670        LDA #BUFFER  SET UP PNTR INTO BUFFER
1680        STA PNTR
1690        LDA /BUFFER
1700        STA PNTR+1
1710 *---CHECK IF FINISHED------------
1720 .1     LDA PNTR
1730        CMP A+1
1740        LDA PNTR+1
1750        SBC A+2
1760        BCC .2
1770        RTS
1780 *---DISPLAY NEXT 8 BYTES---------
1790 .2     LDY #1
1800        JSR WORD     DISPLAY RETURN ADDRESS
1810        LDA #":"     "XXXX:"
1820        JSR COUT
1830        JSR BYTE     DISPLAY ($42)=OPCODE
1840        JSR BYTE     DISPLAY ($43)=UNIT NUMBER
1850        INY
1860        JSR WORD     DISPLAY ($44,45)=BUFFER ADDR
1870        JSR DOT
1880        JSR WORD     DISPLAY ($46,47)=BLOCK NUMBER
1890        JSR CROUT    CARRIAGE RETURN
1900        LDA PNTR     ADVANCE PNTR TO NEXT
1910        CLC               GROUP OF 8 BYTES
1920        ADC #8
1930        STA PNTR
1940        BCC .1
1950        INC PNTR+1
1960        BNE .1       ...ALWAYS
1970 *--------------------------------
1980 WORD   LDA (PNTR),Y      DISPLAY HI-BYTE
1990        JSR PRBYTE
2000        DEY               DISPLAY LO-BYTE
2010        LDA (PNTR),Y
2020        INY
2030        INY               ADVANCE INDEX
2040        JMP PRBYTE
2060 *--------------------------------
2070 BYTE   LDA (PNTR),Y      DISPLAY BYTE
2080        JSR PRBYTE
2090 DOT    LDA #"."          PRINT "."
2100        INY               ADVANCE INDEX
2110        JMP COUT
2120 *--------------------------------
```

```
################################################################################
#  An Easier QUIT from ProDOS
################################################################################
```

Mark Jackson
Chicago, IL

November 1985

When using a hard disk with ProDOS it is often useful to use the MLI QUIT call to go from one application to another. However, if you are deep within a subdirectory the QUIT code makes you retype the entire Prefix if you want to shorten it. To allow the use of the right arrow during the QUIT call do the following:

```
UNLOCK PRODOS
BLOAD PRODOS,A$2000,TSYS
CALL-151
5764:75      (for ProDOS 1.1.1 -- use 5964 for 1.0.1)
BSAVE PRODOS,A$2000,TSYS
LOCK PRODOS
```

This changes the input call to $FD75 which allows right arrow input. There is one drawback: now to restore the prompted prefix you must press ESCape when asked for the Pathname of the next application.

```
################################################################################
#  Commented Listing of ProDOS QUIT Code
################################################################################
```

Bob Sander-Cederlof

November 1985

After reading Mark Jackson's article on improving the ProDOS QUIT code, I though it would be nice to have a commented listing of that program. The listing which follows is just that.

The ProDOS QUIT code is booted into $D100-D3FF in the alternate $D000 bank (the one you get by diddling $C083). Normally ProDOS MLI stays in the $C08B side. When a program issues the QUIT call (MLI code $65), the contents of $D100-D3FF are copied to $1000-12FF; then ProDOS jumps to $1000.

If you BLOAD the SYS file named PRODOS from a bootable ProDOS 1.1.1 disk, and examine it, you will find that it is laid out in eight parts. The first part is a relocator, which copies the other seven parts into their normal homes. Like this:

Position as loaded	Position copied to	
2000-29FF	---	Relocator
2A00-2BFF	Aux 200-3FF	/RAM/ driver
2C00-2C7F	FF00-FF7F	/RAM/ driver
2C80-2CFF	nowhere	All zeroes
2D00-4DFF	D000-F0FF	MLI Kernel
4E00-4EFF	BF00-BFFF	System Global Page
4F00-4F7F	D742-D7BD	Thunderclock driver
4F80-4FFF	FF80-FFFF	Interrupt Code
5000-56FF	F800-FEFF	Device Drivers
5700-59FF	D100-D3FF(alt)	QUIT Code
zeroes	F100-F7FF	

The part I am interested in right now is the QUIT code, which is at $5700-$59FF in the PRODOS file.

The QUIT code is not written very efficiently. For some reason, there are two completely separate editing programs: one for the prefix, and another for the pathname. (And as Mark points out, neither one is very handy.) Even the code that initializes the BITMAP is inefficient.

```
1000 *SAVE S.PRODOS.QUIT
1010 *--------------------------------
1020 CH     .EQ $24
1030 CV     .EQ $25
1040 ERRCOD .EQ $DE
1050 *--------------------------------
1060 BUF    .EQ $0280
1070 *--------------------------------
1080 SYSTEM .EQ $2000
1090 *--------------------------------
1100 MLI    .EQ $BF00
1110 BITMAP .EQ $BF58
```

```
1120 *--------------------------------
1130 KEY    .EQ $C000
1140 S80STOREOFF .EQ $C000
1150 S80OFF .EQ $C00C
1160 SALTON .EQ $C00F
1170 STROBE .EQ $C010
1180 ROM    .EQ $C082
1190 *--------------------------------
1200 HOME   .EQ $FC58
1210 CLREOL .EQ $FC9C
1220 RDKEY  .EQ $FD0C
1230 CROUT  .EQ $FD8E
1240 COUT   .EQ $FDED
1250 SETKBD .EQ $FE89
1260 SETVID .EQ $FE93
1270 BELL   .EQ $FF3A
1280 *--------------------------------
1290        .MA MLI
1300        JSR MLI
1310        .DA #$]1,]2
1320        .EM
1330 *--------------------------------
1340        .OR $1000
1350        .TA $5700
1360 *--------------------------------
1370 PRODOS.QUIT
1380        LDA ROM      TURN ON THE MONITOR ROM
1390        JSR SETVID   GET BACK TO GOOD OLD-FASHIONED
1400        JSR SETKBD        DOWN-HOME 40 COLUMN DISPLAY
1410        STA S80OFF
1420        STA SALTON   Know what I mean, Vern?
1430        STA S80STOREOFF
1440 *---PREPARE BITMAP---------------
1450        LDX #$17
1460        LDA #1     Mark $BFxx in use
1470        STA BITMAP,X
1480        DEX
1490        LDA #0       Most pages are free
1500 .1     STA BITMAP,X
1510        DEX
1520        BPL .1
1530        LDA #$CF     $0000-01FF, $0400-07FF in use
1540        STA BITMAP
1550 *---DISPLAY PREFIX---------------
1560 GET.PREFIX
1570        JSR HOME
1580        JSR CROUT
1590        LDA #Q.PRFX
1600        STA MSG.ADDR
1610        LDA /Q.PRFX
1620        STA MSG.ADDR+1
1630        JSR PRINT.MESSAGE
1640        LDA #3       VTAB 4
1650        STA CV
1660        JSR CROUT    MAKE IT 5
1670        >MLI C7,PREFIX.PARM
```

```
1680        LDX BUF      # CHARS IN PREFIX
1690        LDA #0       MARK END OF PREFIX WITH 00
1700        STA BUF+1,X  SO OUR MESSAGE PRINTER WILL
1710        LDA #BUF+1        PRINT IT.
1720        STA MSG.ADDR
1730        LDA /BUF+1
1740        STA MSG.ADDR+1
1750        JSR PRINT.MESSAGE
1760 *---GET NEW PREFIX---------------
1770        LDX #0
1780        DEC CV       MOVE CURSOR TO BEGINNING OF LINE
1790        JSR CROUT
1800 NEXT.PREFIX.CHAR
1810        JSR RDKEY
1820        CMP #$8D
1830        BEQ SET.NEW.PREFIX   ...ACCEPT WHAT IS ON SCREEN
1840        PHA          ERASE PREFIX FROM SCREEN
1850        JSR CLREOL
1860        PLA
1870        CMP #$9B     IS CHAR <ESCAPE>?
1880        BEQ GET.PREFIX   ...YES, START ALL OVER
1890        CMP #$98     IS CHAR CTRL-X?
1900 START.PREFIX.OVER
1910        BEQ GET.PREFIX    ...START ALL OVER
1920        CMP #$89     IS CHAR <TAB>?
1930        BEQ .3       ...YES, RING BELL
1940        CMP #$88     IS CHAR BACKSPACE?
1950        BNE .2       ...NO, APPEND TO LINE
1960        CPX #0       ...BACKSPACE, UNLESS AT BEGINNING
1970        BEQ .1       AT BEGINNING ALREADY
1980        DEC CH       BACK UP
1990        DEX
2000 .1     JSR CLREOL   CHOP OFF AFTER CURSOR
2010        JMP NEXT.PREFIX.CHAR
2020 .2     BCS .4       OTHER CONTROL CHAR < $88
2030 .3     JSR BELL
2040        JMP NEXT.PREFIX.CHAR
2050 .4     CMP #"Z"+1
2060        BCC .5       ...NOT LOWER CASE
2070        AND #$DF     CONVERT LOWER CASE TO UPPER
2080 .5     CMP #"."     ALLOW PERIOD, SLASH, DIGITS
2090        BCC .3       ...TOO SMALL
2100        CMP #"Z"+1   ALLOW LETTERS
2110        BCS .3       ...TOO LARGE
2120        CMP #"9"+1
2130        BCC .6       ...PERIOD, SLASH, OR DIGIT
2140        CMP #"A"
2150        BCC .3       ...NOT A LEGAL CHARACTER
2160 .6     INX
2170        CPX #$27
2180        BCS START.PREFIX.OVER   ...TOO LONG
2190        STA BUF,X
2200        JSR COUT     ECHO THE CHARACTER
2210        JMP NEXT.PREFIX.CHAR
2220 *--------------------------------
2230 SET.NEW.PREFIX
```

```
2240        CPX #0       DID WE CHANGE IT?
2250        BEQ GET.PATHNAME   ...NO
2260        STX BUF      ...YES, SO TELL SYSTEM
2270        >MLI C6,PREFIX.PARM
2280        BCC GET.PATHNAME   ...NO ERRORS
2290        JSR BELL     DING, DONG!
2300        LDA #0       SET .EQ. STATUS
2310 PFXOVR BEQ START.PREFIX.OVER    ...ALWAYS
2320 *--------------------------------
2330 GET.PATHNAME
2340        JSR HOME
2350 START.PATHNAME.OVER
2360        JSR CROUT
2370        LDA #Q.PATH
2380        STA MSG.ADDR
2390        LDA /Q.PATH
2400        STA MSG.ADDR+1
2410        JSR PRINT.MESSAGE
2420        LDA #3       VTAB 4
2430        STA CV
2440        JSR CROUT    MAKE IT 5
2450        LDX #0
2460 NEXT.PATHNAME.CHAR
2470        LDA #$FF     CURSOR CHARACTER
2480        JSR COUT
2490        DEC CH       BACK UP OVER CURSOR
2500 .1     LDA KEY
2510        BPL .1       ...WAIT TILL KEY PRESSED
2520        STA STROBE
2530        CMP #$9B     <ESCAPE>?
2540        BNE .2       ...NO
2550        LDA CH       IF AT BEGINNING, GET PREFIX OVER
2560        BNE GET.PATHNAME      ...ELSE GET PATHNAME OVER
2570        BEQ PFXOVR
2580 .2     CMP #$98     CONTROL-X?
2590 .3     BEQ GET.PATHNAME
2600        CMP #$89     TAB KEY?
2610        BEQ .5       ...YES
2620        CMP #$88     BACKSPACE?
2630        BNE .4       ...NO
2640        JMP BACKSPACE.IN.PATHNAME
2650 *--------------------------------
2660 .4     BCS .6
2670 .5     JSR BELL     ...INVALID CHAR, RING BELL
2680        JMP NEXT.PATHNAME.CHAR
2690 *--------------------------------
2700 .6     CMP #$8D
2710        BEQ SET.NEW.PATHNAME
2720        CMP #"Z"+1
2730        BCC .7
2740        AND #$DF     CHANGE LOWER CASE TO UPPER
2750 .7     CMP #"."     ACCEPT DOT, SLASH, OR DIGIT
2760        BCC .5       ...TOO SMALL
2770        CMP #"Z"+1   ACCEPT LETTERS
2780        BCS .5       ...TOO LARGE
2790        CMP #"9"+1
```

```
2800        BCC .8       ...DOT, SLASH, OR DIGIT
2810        CMP #"A"
2820        BCC .5       ...NOT A VALID CHARACTER
2830 .8     PHA          CLEAR BEYOND THIS POINT
2840        JSR CLREOL
2850        PLA
2860        JSR COUT     ECHO THE NEW CHARACTER
2870        INX
2880        CPX #$27
2890        BCS .3       ...NAME TOO LONG
2900        STA BUF,X    APPEND CHAR TO NAME
2910        JMP NEXT.PATHNAME.CHAR
2920 *--------------------------------
2930 SET.NEW.PATHNAME
2940        LDA #" "
2950        JSR COUT
2960        STX BUF
2970        >MLI C4,FILE.INFO.PARM
2980        BCC .1       ...NO ERRORS
2990        JMP PROCESS.ERROR
3000 *--------------------------------
3010 .1     LDA FILTYP   FILE.INFO.PARM+4
3020        CMP #$FF
3030        BEQ .2       "SYS" FILE
3040        LDA #1
3050        JMP PROCESS.ERROR
3060 *--------------------------------
3070 .2     LDA #0
3080        STA CL.REF   CLOSE.PARM+1, REF NO.
3090        >MLI CC,CLOSE.PARM
3100        BCC .3       ...NO ERROR
3110        JMP PROCESS.ERROR
3120 *--------------------------------
3130 .3     LDA ACBITS   FILE.INFO.PARM+3
3140        AND #1
3150        BNE .4       ...OKAY TO READ IT
3160        LDA #$27
3170        JMP PROCESS.ERROR
3180 *--------------------------------
3190 .4     >MLI C8,OPEN.PARM
3200        BCC .5       ...NO ERRORS
3210        JMP PROCESS.ERROR
3220 *--------------------------------
3230 .5     LDA OP.REF   OPEN.PARM+5, REF NO.
3240        STA RD.REF   READ.PARM+1, REF NO.
3250        STA EF.REF   EOF.PARM+1, REF NO.
3260        >MLI D1,EOF.PARM
3270        BCC .6       ...NO ERRORS
3280        JMP PROCESS.ERROR
3290 *--------------------------------
3300 .6     LDA FIL.SZ+2    EOF.PARM+4
3310        BEQ .7       ...NOT TOO LONG
3320        LDA #$27
3330        JMP PROCESS.ERROR
3340 *--------------------------------
3350 .7     LDA FIL.SZ      EOF.PARM+2
```

```
3360        STA READ.PARM+4
3370        LDA FIL.SZ+1    EOF.PARM+3
3380        STA READ.PARM+5
3390        >MLI CA,READ.PARM
3400        PHP
3410        >MLI CC,CLOSE.PARM
3420        BCC .9
3430        PLP
3440 .8     JMP PROCESS.ERROR
3450 *--------------------------------
3460 .9     PLP
3470        BCS .8
3480        JMP SYSTEM
3490 *--------------------------------
3500 BACKSPACE.IN.PATHNAME
3510        LDA CH       UNLESS ALREADY AT BEGINNING
3520        BEQ .1       ...WE WERE
3530        DEX
3540        LDA #" "
3550        JSR COUT
3560        DEC CH
3570        DEC CH
3580        JSR COUT
3590        DEC CH
3600 .1     JMP NEXT.PATHNAME.CHAR
3610 *--------------------------------
3620 PRINT.MESSAGE
3630        LDX #0
3640 MSG.LP LDA MSG.LP,X
3650 MSG.ADDR   .EQ *-2
3660        BEQ .1
3670        ORA #$80
3680        JSR COUT
3690        INX
3700        BNE MSG.LP
3710 .1     RTS
3720 *--------------------------------
3730 PROCESS.ERROR
3740        STA ERRCOD
3750        LDA #12      VTAB 13
3760        STA CV
3770        JSR CROUT    MAKE IT 14
3780        LDA ERRCOD
3790        CMP #1
3800        BNE .1
3810        LDA #ERQT.1
3820        STA MSG.ADDR
3830        LDA /ERQT.1
3840        STA MSG.ADDR+1
3850        BNE .3
3860 .1     CMP #$40
3870        BEQ .2
3880        CMP #$44
3890        BEQ .2
3900        CMP #$45
3910        BEQ .2
```

```
3920        CMP #$46
3930        BEQ .2
3940        LDA #ERQT.2
3950        STA MSG.ADDR
3960        LDA /ERQT.2
3970        STA MSG.ADDR+1
3980        BNE .3        ...ALWAYS
3990 .2     LDA #ERQT.3
4000        STA MSG.ADDR
4010        LDA /ERQT.3
4020        STA MSG.ADDR+1
4030 .3     JSR PRINT.MESSAGE
4040        LDA #0       VTAB 1
4050        STA CV
4060        JMP START.PATHNAME.OVER
4070 *--------------------------------
4080 Q.PRFX .AS -/ENTER PREFIX (PRESS "RETURN" TO ACCEPT)/
4090        .HS 00
4100 Q.PATH .AS -/ENTER PATHNAME OF NEXT APPLICATION/
4110        .HS 00
4120 ERQT.1 .HS 87
4130        .AS -/NOT A TYPE "SYS" FILE/
4140        .HS 00
4150 ERQT.2 .HS 87
4160        .AS -"I/O ERROR            "
4170        .HS 00
4180 ERQT.3 .HS 87
4190        .AS -"FILE/PATH NOT FOUND  "
4200        .HS 00
4210 *--------------------------------
4220 FILE.INFO.PARM
4230        .DA #10
4240        .DA BUF
4250 ACBITS .HS 00
4260 FILTYP .HS 00
4270        .BS 13
4280 *--------------------------------
4290 OPEN.PARM
4300        .DA #3
4310        .DA BUF
4320        .DA $1800    BUFFER ADDR
4330 OP.REF .BS 1        REF NO.
4340 *--------------------------------
4350 CLOSE.PARM
4360        .DA #1
4370 CL.REF .BS 1        REF NO.
4380 *--------------------------------
4390 READ.PARM
4400        .DA #4
4410 RD.REF .BS 1        REF NO.
4420        .DA $2000    BUFFER ADDR
4430        .BS 2        # BYTES TO READ
4440        .BS 2        # ACTUALLY READ
4450 *--------------------------------
4460 EOF.PARM
4470        .DA #2
```

```
4480 EF.REF .BS 1        REF NO.
4490 FIL.SZ .BS 3        EOF POSITION
4500 *--------------------------------
4510 PREFIX.PARM
4520        .DA #1
4530        .DA BUF
4540 *--------------------------------
4550        .LIF
```

```
################################################################################
#  ProDOS MLI Tracing
################################################################################
```

Ken Kashmarek
Eldridge, Iowa

December 1985

I took Bob S-C's work with ProDOS Snooper (October 1985 AAL) one step further: I added MLI calls to the information that is collected in the trace table. By combining the MLI call data with the device driver data, we get a better idea of what is happening.

The entries below all come from slot 6 drive 1. MLI calls are tagged with an "M" after the hex data. To support both the MLI calls and device driver calls, the hex output provides the data as it exists in memory without taking into account whether a set of bytes is a two byte memory pointer or a single data byte.

For all calls, the return address is still shown as hi-byte first before the colon. Data for the device driver parameter is still from $42-$47. For MLI calls, the return address is to the program that called the routine in the BASIC.SYSTEM global page. All BASIC.SYSTEM calls go to the $BE00 global page and then to the $BF00 ProDOS global page. MLI data is the MLI call number followed by the first five bytes of the parameter list (some bytes do not apply if the list is shorter).

The volume in question is labeled /TEST and has one file, ABC, in the root directory.

First of all, issue: CAT,S6

```
     A6E9:C7 BC BC 02 BC BC M GET PREFIX
     A85F:C5 60 01 02 00 03 M ON LINE CALL   + Not used when
     EC0C:01 50 00 DC 02 00   READ BLOCK 2   + CAT /TEST entered
     A825:C4 BC BC C3 0F 00 M GET FILE INFO
     EC0C:01 60 00 DC 02 00   READ BLOCK 2
     EC0C:01 60 00 DC 06 00   READ Bit Map
     B1B9:C8 BC BC 00 8A 01 M OPEN FILE
     EC0C:01 60 00 DC 02 00   READ BLOCK 2
     EE85:01 60 00 8A 02 00   READ BLOCK 2
     B175:CA 01 59 02 2B 00 M READ FILE
     B201:CE 01 2B 00 00 03 M SET FILE MARK  + Appears for each
     B208:CA 01 59 02 27 00 M READ FILE      + file in directory
     B0A5:CC 01 00 C3 CF D0 M CLOSE FILE
     B0FB:C5 60 BD BC 00 03 M ON LINE CALL
     EC0C:01 50 00 DC 02 00   READ BLOCK 2
     B10F:C4 BC BC C3 0F 18 M GET FILE INFO
     EC0C:01 60 00 DC 02 00   READ BLOCK 2
     EC0C:01 60 00 DC 06 00   READ Bit Map
```

For this simple operation, there are ten MLI calls and eight device driver calls (disk I/O operations). I do not understand the reason for the Get Prefix call at the beginning. It would appear that the On Line call and the Get File Info call at the end are unnecessary (we will be checking this out as we go). On Line returns the volume name, but this should already be available through the prefix or pathname of the directory. Get File Info information should already be available from the previous call, and the bit map was already read in once. However, this is a simple catalog operation and may be indicative of some of the steps necessary for more complex catalog operations.

Carrying this one step further, I issued CAT /TEST/DIR. In this case, the first read of the bit map is not performed. Next, the former apparently duplicate read of block 2 now turns into a read of block 7, the key block for subdirectory DIR (in /TEXT/DIR; the device driver return address is $EE85, the buffer address is $8A00). Note: block 2 is the key block of the root directory.

A Get File Info call for a volume name (/TEST) always reads the bit map. Therefore, this call is repeated when cataloging a volume, but not when cataloging a subdirectory. As to the On Line call, it is used to get volume name for the Get File Info call for the free space information for the volume, since the initial catalog command may have been for a subdirectory. This explains (only partially) what appeared to be duplicate reads of the same information.

Now, let's try loading an Applesoft file: LOAD ABC,S6

```
     A85F:C5 60 01 02 00 03 M ON LINE CALL   + Not used for
     EC0C:01 60 00 DC 02 00   READ BLOCK 2   + LOAD /TEST/ABC
     A825:C4 BC BC E3 FC 01 M GET FILE INFO
     EC0C:01 60 00 DC 02 00   READ BLOCK 2
     AC00:CC 00 00 C3 CF D0 M CLOSE ALL FILES
     B1B9:C8 BC BC 00 8A 01 M OPEN FILE
     EC0C:01 60 00 DC 02 00   READ BLOCK 2
     EE85:01 60 00 8A 07 00   READ BLOCK 7
     AC22:D1 01 01 02 00 03 M GET FILE EOF
     AC4B:CA 01 01 08 09 00 M READ FILE
     AC50:CC 01 00 C3 CF D0 M CLOSE FILE
```

The loaded program is less than 512 bytes in length, so the key block read is the only data I/O operation. As with the catalog operation, the Get File Info call is used to verify the file type. Close All Files is used in case the previous program left any open. Note the Get File EOF call which is used to get the length for the Read File call (which performs the entire load operation). This example is relatively simple. Let's check what happens when we create an Applesoft file that is just over 512 bytes in length (changing our seedling file into a sapling file, which requires an index block and two data blocks).

We'll lengthen the program, and then type: SAVE /TEST/ABC.3

```
     A825:C4 BC BC C3 0F 18 M GET FILE INFO
     EC0C:01 60 00 DC 02 00   READ BLOCK 2
     ACDC:C0 BC BC C3 FC 01 M CREATE FILE
     EC0C:01 60 00 DC 02 00   READ BLOCK 2
     F477:00 60 00 DC 00 00   STATUS S6,D1
     EC0C:01 60 00 DA 06 00   READ BIT MAP
     EC0C:02 60 00 DC 07 00   WRITE BLOCK 7
     EC0C:01 60 00 DC 02 00   READ BLOCK 2
     EC0C:02 60 00 DC 02 00   WRITE BLOCK 2
     EC0C:02 60 00 DA 06 00   WRITE BIT MAP
     B1B9:C8 BC BC 00 8A 01 M OPEN FILE CALL
     EC0C:01 60 00 DC 02 00   READ BLOCK 2
     EE85:01 60 00 8A 07 00   READ BLOCK 7
     AD0A:CB 01 01 08 5B 02 M WRITE FILE CALL
     F477:00 60 01 08 00 00   STATUS S6,D1
```

```
EE85:02 60 00 8A 07 00   WRITE BLOCK 7
EC0C:01 60 00 DA 06 00   READ BIT MAP
EC0C:02 60 00 DA 06 00   WRITE BIT MAP
EE85:02 60 00 8C 08 00   WRITE BLOCK 8
EC0C:01 60 00 DA 06 00   READ BIT MAP
AD11:D0 01 5B 02 00 03 M SET FILE E0F CALL
AD16:CC 01 00 C3 CF D0 M CLOSE FILE CALL
EE85:02 60 00 8A 09 00   WRITE BLOCK 9
EC0C:02 60 00 DA 06 00   WRITE BIT MAP
EE85:02 60 00 8C 08 00   WRITE BLOCK 8
EC0C:01 60 00 DC 02 00   READ BLOCK 2
EC0C:01 60 00 DC 02 00   READ BLOCK 2
EC0C:02 60 00 DC 02 00   WRITE BLOCK 2
```

This sequence has the same number of MLI calls for a seedling or a sapling file. The big difference is allocating the index block (block number 8) and additional data blocks. This also generates additional calls to read and write the bit map.

If the file already exists, and the SAVE command does not change the length, then the Create File call is not executed, there are no accesses to the bit map (block 6), and the index block does not change. If the file length changes sufficiently to add or delete blocks, then the bit map is updated and the index block is rewritten (this is forced by the Set File EOF call which adjusts the file length).

Interesting note: whenever a file is opened, the first data block is always read in, even if the file will subsequently be written to. Likewise, when a new file is allocated, the first data block is allocated and written, even if no data is placed in the block.

In the above sequence, what appears to be a duplicate read of block 2 (return address $EC0C) is actually a read to separate blocks if the SAVE command was to a subdirectory. It turns out to be duplicate reads to the subdirectory block, write to the subdirectory, then read and write the root directory. Sigh.

LOAD /TEST/ABC.3 is similar to the previous load operation, except that we must also read the index block before reading the data blocks, and there are two data blocks rather than one.

Finally, let's try deleting this file: DELETE /TEST/ABC.3

```
A825:C4 BC BC E3 04 00 M GET FILE INFO CALL
EC0C:01 60 00 DC 02 00   READ BLOCK 2
9AD7:C1 BC BC 02 BC BC M DESTROY FILE CALL
EC0C:01 60 00 DC 02 00   READ BLOCK 2
F477:00 60 00 DC 00 00   STATUS S6,D1
EC0C:01 60 00 DC 08 00   READ BLOCK 8 (index block)
EC0C:01 60 00 DA 06 00   READ BIT MAP
EC0C:02 60 00 DC 08 00   WRITE INDEX BLOCK (zeroed)
EC0C:01 60 00 DC 07 00   READ BLOCK 7
EC0C:02 60 00 DA 06 00   WRITE BIT MAP
EC0C:01 60 00 DC 02 00   READ BLOCK 2
EC0C:02 60 00 DC 02 00   WRITE BLOCK 2
```

Again, use Get File Info for file type and status call to see if the disk can be written to. The bit map is read and written to reflect the freed blocks. Block 8, the former file index block, is trashed. I don't know why block 7 is read in. Trashing the index block makes it very hard to reconstruct a DELETEd file.

At this point, we get a feel for what is happening between the MLI calls and the device driver calls. Consider how extensive these simple examples become on a hard disk if working down three or four directory levels and at the second, third, or fourth block in each directroy, and the hard disk has five blocks for the bit map (and we need the fifth block because the disk is almost full). Ouch!

I performed one more test case, far too long to list here. It involved adding a record to a new sparse random access file. The new record caused the file to grow to a tree file. The program used was:

```
10 D$ - CHR$(4)
20 PRINT D$"OPEN /TEST/NAMES,L140"
30 PRINT D$"WRITE/TEST/NAMES,R936"
40 PRINT "XXX ... XXX": REM 120 X's
50 PRINT D$"CLOSE/TEST/NAMES"
```

This sequence produced eight MLI calls and 29 device driver calls to perform I/O (there were three status calls). The file ended up with six blocks (master index block, two index blocks, and three data blocks) which generated 12 accesses to read and write the bit map.

A 32 megabyte hard disk, the maximum size supported by ProDOS, requires 16 blocks for the free space bit map. Obviously, such a disk would suffer quite a performance impact when allocating new files, or adding space to existing files, if the hard disk were more than half full.

```
################################################################################
#  Correction to DOS/ProDOS Double Init
################################################################################
```

Bob Sander-Cederlof

January 1986

The Sep 85 (V5N12) issue of AAL included an article and program to initialize a disk with both DOS and ProDOS catalogs in separate halves of the disk. After trying to use Catalog Arranger on a disk we made with DOUBLE.INIT, we discovered that program has a bug.

The DOS catalog is written in track $11, starting with sector 15 and going backwards to sector 1. The second and third bytes in each catalog sector are supposed to point to the next catalog sector, with the exception of those bytes in the LAST catalog sector. In the last catalog sector, the link bytes should both be $00, to signal to anyone who tries to read the catalog that this is indeed the last sector. DOUBLE.INIT stored $11 in the first link byte, and so some catalog reading programs such as Catalog Arranger get very confused.

The fix is to add the following lines to the program, where the line numbers correspond to those in the printed listing in AAL:

```
       2201      BNE .5
       2202      STY C.TRACK   (Y=0)
```

Add the label ".5" to line 2210, so that it reads:

```
       2210 .5   JSR CALL.RWTS
```

If you have already created some disks with DOUBLE.INIT, we suggest you use a program such as Bag of Tricks, CIA, or some other disk zap program to clear the second byte of track $11, sector $01 on those disks.

```
################################################################################
#  Modifying ProDOS for Non-Standard ROMs
################################################################################
```

Bob Sander-Cederlof

March 1986

We have published several times ways to defeat the ROM Checksummer that is executed during a ProDOS boot, so that owners of Franklin clones (or even real Apples with modified monitor ROMs) could use ProDOS-based software. See AALs of March and June, 1984.

Both of these previous articles are out of date now, because they apply to older versions of ProDOS than are current. What follows applies to Version 1.1.1 of ProDOS.

There are two problems with getting ProDOS to boot on a non-standard machine. The first is the ROM Checksummer. This subroutine starts at $267C in Version 1.1.1, and is only called from $25EE. The code is purposely weird, designed to look like it is NOT checking the ROMs. It also has apparently purposeful side effects. Here is a listing of the subroutine:

```
  1000 *SAVE CHECKSUMMER
  1010 *--------------------------------
  1020        .OR $267C    POSITION IN PRODOS SYSTEM FILE
  1030 *--------------------------------
  1040 CHECKSUMMER
  1050        CLC
  1060        LDY $2674    (GETS A VALUE 0)
  1070 .1     LDA ($0A),Y  GETS (FB09...FB10)
  1080        AND #$DF     STRIP OFF LOWER CASE BIT
  1090        ADC $2674    ACCUMULATE SHIFTED SUM
  1100        STA $2674
  1110        ROL $2674    SHIFT RESULT, CARRY INTO BIT 0
  1120        INY
  1130        CPY $2677    DO IT 8 TIMES
  1140        BNE .1
  1150        TYA          A = Y = 8
  1160        ASL          FORM $80 BY SHIFTING
  1170        ASL
  1180        ASL
  1190        ASL
  1200        TAY          $80 TO Y FOR LATER TRICK
  1210        EOR $2674    MERGE WITH PREVIOUS "SUM"
  1220        ADC #11      FORM $00 FOR VALID ROMS
  1230        BNE .2       ...NOT A VALID ROM
  1240        LDA $0C      GET MACHINE ID BYTE
  1250        RTS
  1260 .2     LDA #0       SIGNAL INVALIDITY
  1270        RTS
```

The pointer at $0A,0B was set up to point to $FB09 using very sneaky code at $248A. Location $2674 initially contains a 0, and $2677 contains an 8. Only the bytes from $FB09 through $FB10 are checksummed. Truthfully, "checksummed" is not the correct word.

The wizards who put ProDOS together figured out a fancy function which changes the 64 bits from $FB09 through $FB10 into the value $75. Their function does this whether your ROMs are the original monitor ROM from 1977-78, the Autostart ROM, the original //e ROM, or any other standard Apple ROM. The values in $FB09-FB10 are not the same in all cases, but the function result is always $75. However, a Franklin ROM does not produce $75. Probably a BASIS also gives a different result, and other clones. Once $75 is obtained, further slippery code changes the value to $00.

The original Apple II ROM has executable code at $FB09, and in hex it is this: B0 A2 20 4A FF 38 B0 9E. All other Apple monitor ROMs have an ASCII string at $FB09. The string is either "APPLE][" or "Apple][". Notice that the "AND #$DF" in the checksummer strips out the upper/lower case bit, making both ASCII strings the same.

I wrote a test program to print out all the intermediate values during the "Checksummer's" operation. Here are the results, for both kinds of ROMs.

Original ROM					Later ROMs				
LDA	AND	ADC	STA	ROL	LDA	AND	ADC	STA	ROL
B0	90	00	90	20	C1	C1	00	C1	82
A2	82	20	A2	44	D0/F0	D0	82	52	A5
20	00	44	44	88	D0/F0	D0	A5	75	EB
4A	4A	88	D2	A4	CC/EC	CC	EB	B7	6F
FF	DF	A4	83	07	C5/E5	C5	6F	34	69
38	18	07	1F	3E	A0	80	69	E9	D2
B0	90	3E	C3	9C	DD	DD	D2	AF	5F
9E	9E	9C	3A	75	DB	DB	5F	3A	75

I don't understand why this code gives the same result, but I see it does. Now, dear readers, tell me how anyone ever figured out what sequence of operations would produce the same result using these two different sets of eight bytes, and yet produce a different result for clones! If you understand it, please explain it to me!

By the way, here is a listing of my test program:

```
1000  .LIF
1010 *SAVE TEST.CKSUMMER
1020 *--------------------------------
1030 *   SIMULATE PRODOS $FB09-FB10 CHECK-SUMMER
1040 *      (AT $267C IN PRODOS 1.1.1)
1050 *--------------------------------
1060 T
1070        LDA #S1
1080        STA $0A
1090        LDA /S1
1100        STA $0B
1110        JSR CS
1120        LDA #S2
1130        STA $0A
1140        LDA /S2
1150        STA $0B
1160 CS
1170        JSR PT
1180        CLC
1190        LDY #0
1200        STY X
1210 .1     LDA ($0A),Y
```

```
1220        JSR B
1230        AND #$DF
1240        JSR B
1250        LDA X
1260        JSR B
1270        LDA ($0A),Y
1280        AND #$DF
1290        ADC X
1300        STA X
1310        JSR B
1320        ROL X
1330        LDA X
1340        JSR B
1350        JSR $FD8E
1360        INY
1370        CPY #8
1380        BCC .1
1390        TYA
1400        ASL
1410        ASL
1420        ASL
1430        ASL
1440        ORA X
1450        JSR B
1460        ADC #$0B
1470 *--------------------------------
1480 B      PHA
1490        PHP
1500        JSR $FDDA
1510        LDA #" "
1520        JSR $FDED
1530        JSR $FDED
1540        PLP
1550        PLA
1560        RTS
1570 *--------------------------------
1580 X      .BS 1
1590 *--------------------------------
1600 S1     .AS -/APPLE ][/
1610 S2     .HS B0.A2.20.4A.FF.38.B0.9E
1620 *--------------------------------
1630 TITLE  .HS 8D8D
1640        .AS -/LDA AND ADC STA ROL/
1650        .HS 8D00
1660 *--------------------------------
1670 PT
1680        LDY #0
1690 .1     LDA TITLE,Y
1700        BEQ .2
1710        JSR $FDED
1720        INY
1730        BNE .1
1740 .2     RTS
1750 *--------------------------------
```

The checksummer can be defeated. The best way, preserving the various side effects, is to change the byte at $269F from $03 to $00. This changes the BNE to an effective no-operation, because it will branch to the next instruction regardless of the status. Another way to get the same result is to store $EA at both $269E and $269F. Still another way is to change the "LDA #0" at $26A3,4 to "LDA $0C" (A5 0C), so that either case gives the same result.

If it thinks it is in a valid Apple computer, the checksummer returns a value in the A-register which is non-zero, obtained from location $0C. The value at $0C has been previously set by looking at other locations in the ROM, trying to tell which version is there. Part of this code is at $2402 and following, and part is at $2047 and following. The byte at $0C will eventually become the Machine ID byte at $BF98 in the System Global Page, so it also gets some bits telling how much RAM is available, and whether an 80-column card and a clock card are found.

If you have a non-standard Apple or a clone the bytes which are checked to determine which kind of ROM you have may give an illegal result. The following table shows the bytes checked, and the resulting values for $0C. The values in parentheses are not ever checked, but I included them for completeness. The value in $0C will be further modified to indicate the amount of RAM found and the presence of a clock card.

Version	FBB3	FB1E	FBC0	FBBF	$0C
Original Apple II	38	(AD)	(60)	(2F)	00
Autostart, II Plus	EA	AD	(EA)	(EA)	40
Original //e	06	(AD)	EA	(C1)	80
Enhanced //e	06	(AD)	E0	(00)	80
DEBUG //e	06	(AD)	E1	(00)	80
Original //c	06	(4C)	00	FF	88
//c Unidisk 3.5	06	(4C)	00	00	88
/// Emulating II	EA	8A	(??)	(??)	C0

By the way, ProDOS 1.1.1 will not allow booting by an Apple /// emulating a II Plus, possibly because the standard emulator only emulates a 48K machine.

I have no idea what a clone would have in those four locations, but chances are it would be different. You should probably try to fool ProDOS into thinking you are in a II Plus, because most clones are II Plus clones. This means you should somehow change the ID procedures so that the result in $0C is a value of $40. One way to do this is change the code at $2402 and following like this:

```
         Standard                     Change to

   2402- A9 00    LDA #0        2402- A9 40    LDA #$40
   2404- 85 0C    STA $0C       2404- 4C 2E 24 JMP $242E
   2406- A3 B3 FB LDX $FBB3
```

If your clone or modified ROM is a //e, change $2402 to LDA #$80 instead.

You may also need to modify the code at $2047 and following. If you are trying to fool ProDOS into thinking you are an Apple II Plus or //e, and have already made the change described above, change $2047-9 like this:

```
         Standard                    Change to

   2047- AE B3 FB LDX $FBB3    2047- 4C 6D 20 JMP $206D
```

No doubt future versions of ProDOS will make provision for clones and modified ROMs even more difficult. And there are always the further problems encountered by usage of the ROMs from BASIC.SYSTEM and the ProDOS Kernel and whatever application program is running.

I am intrigued about seeing what the minimum amount of code is that can distinguish between the four legal varieties of ROM for ProDOS. I notice from the table above that I can identify the four types and weed out the ///emulator by the following simple code at $2402:

```
            LDA $FBB3
            ORA $FB1E
            LDX #3
       .1   CMP TABLE.1,X
            BEQ .2
            DEX
            BPL .1
            SEC
            RTS
     *
     TABLE.1  .HS BD.EF.AF.4E
     TABLE.2  .HS 00.40.80.88
     *
       .2   LDA TABLE.2,X
            JMP $242E
```

With this code installed, all the code from $2047-$206C is not needed, and the JMP $206E should be installed at $2047. The new code at $2402 fits in the existing space with room to spare. Can you do it with even shorter code?

```
################################################################################
#  New ProDOS Program Selector
################################################################################
```

Bob Sander-Cederlof

July 1986

In the November 1985 issue of Apple Assembly Line I printed a complete commented listing of the ProDOS QUIT code. This code resides at $D100-D3FF inside ProDOS, and is downloaded to $1000 and executed by the $65 MLI call.

The BYE command in BASIC.SYSTEM and SCASM.SYSTEM both call ProDOS MLI with call number $65, and so do many other system programs. For some reason FILER has its own quit code, which operates slightly differently from MLI-$65, but not really better. No one seems to particularly like MLI-$65, but they usually learn to live with it. That is, unless they purchase Catalyst, MouseFiler, or one of the other commercially-available ProDOS program selectors.

Not wanting to buy three or four different program selectors until I found one I liked, I decided to try writing my own. It replaces the standard QUIT code inside ProDOS, so that MLI-$65 downloads and executes my new code. My program first lists all of the on-line volume names, so that you can select a volume. You perform the selection by moving the cursor-bar with the arrow keys, and pressing RETURN. ESCAPE makes the program re-do the list of volume names, in case you want to change diskettes. Once a volume is selected, all of the system (SYS) and directory (DIR) filenames in that volume will be listed. Again, you use the arrow keys and RETURN to select either a system program to be executed, or a sub-directory to display. Just a few quick keystrokes and you are in a new application!

Here is an example of the volume name display:

```
       S/D  VOLUME NAME

       3/2  RAM
       7/1  HARD1
       7/2  HARD2
       6/1  UTILITIES

       USE ARROWS AND <RETURN> TO SELECT
       USE <ESCAPE> TO TRY AGAIN
```

And here is an example of a filename display:

```
       /HARD1

       SYS -- PRODOS
       SYS -- SCASM.SYSTEM
       SYS -- BASIC.SYSTEM
       SYS -- CONVERT
       SYS -- UTIL.SYSTEM
       DIR -- ASM1
       DIR -- ASM2
       DIR -- SCI
       DIR -- FSE
       DIR -- XREF
```

```
DIR -- SCWP
DIR -- TIMEMASTER
DIR -- THUNDERCLOCK
DIR -- PHASOR
DIR -- MINTERMS
DIR -- DP18
<<<MORE>>>

USE ARROWS AND <RETURN> TO SELECT
USE <ESCAPE> TO TRY AGAIN
```

All of the SYS files are listed first, and then all of the DIR files, regardless of the order within the directory. This makes it easier to find the file you are looking for. If there are more than 16 filenames to display, the first 16 will be listed, followed by the word "<<<MORE>>>". When you use the arrow keys to move beyond the bottom of the list, if there are more filenames, the list will scroll up to make room for the next name on the screen. When the top name listed is not the first name in the list, the word "<<<MORE>>>" will be displayed above the list. Actually, it is easier to use than it is to describe.

I have gotten so used to an 80-column display now that I decided to make the menu in that mode. Lines 1425-1430 initialize the 80-column display for an enhanced Apple //e or //c. If you want to use some other configuration, or just like 40-columns better, replace those two lines with the following:

```
1421    JSR $FE93
1422    JSR $FE89
1423    STA $C00C
1424    STA $C00F
1425    STA $C000
1426 .2 JSR HOME
```

The six lines above make QUITTER a little too long to fit in three pages, so you need to make room for it somehow. I suggest putting the variables from lines 4870-4950 into page zero, say at $06-$0E. This will make the code assemble shorter, so it still fits between $D100 and $D3FF inside ProDOS.

An alternative is to make a further modification to ProDOS. The subroutine which downloads the QUIT code is at $FCE5-FD3A inside ProDOS. It is very inefficient, so there is ample room for adding features. However, by merely changing the LDX #3 at $FD06 to LDX #4, you can make it download four pages instead of three. When you BLOAD PRODOS at $2000, the LDX #3 is found at $4C06. Since the QUIT code is at the end of the PRODOS file, you can write a longer QUIT program if you wish. You also need to change the $03 at $2233 to $04, so that the boot code will install QUIT where it belongs.

Walking through the New QUITTER

The comments in lines 1010-1090 explain how to install the new QUITTER inside the PRODOS system file. Just in case there is an error, I recommend you try this first on a disk you can afford to lose. It all works here, but there's many a slip 'twixt the cup and the lip!

Line 1320 switches on the motherboard ROM code, so that we can use Apple monitor routines. Lines 1330-1410 clear out the memory bitmap in the ProDOS System Global Page. We have to do that so we can load another system file. Once the bitmap has been cleared, it is not safe to try to return to whatever system program was operating before QUITTER was entered. Anyway, the RESET vector has already been pointed at QUITTER, so it is pretty difficult to get out of QUITTER. If you wish, you could add a feature that allows aborting the QUIT call, but be aware that the memory bitmap will have been messed up.

Lines 1420-1590 display a list of all the volumes currently on-line, and allow you to move the cursor bar up and down the list. The subroutine DISPLAY.VOLUMES lists the volume names, displaying the one under the cursor in inverse mode. The subroutine GET.KEY accepts the four arrow keys, RETURN, and ESCAPE. The left and up arrows move the cursor bar up, while the right and down arrows move the cursor bar down. GET.KEY is a little complicated, since it also handles windowing for long lists of filenames.

The subroutine READ.THE.FILE, called from line 1610, reads in an entire volume directory or sub-directory. ProDOS has the built-in ability to read directories just as though they were regular files, so READ.THE.FILE is pretty simple: it merely OPENs the file, READs it, and CLOSEs it. Lines 2030-2150 perform the additional task of appending the current volume or filename to the previous prefix.

Lines 1640-1710 clear the screen and display the pathname of the selected directory, in preparation for display a file menu. Lines 1720-1800 collect a list of pointers to all of the SYS and DIR files in the directory, using the SCAN.DIRECTORY subroutine. SCAN.DIRECTORY appends a pointer to a list of pointers in DIRBUF for each file it finds of the specified type.

Lines 1810-1900 display the SYS and DIR files found in the directory. If there are more than 16 files, the word "<<<MORE>>>" will be displayed after the 16th name. Moving the cursor bar down will scroll the list up, so that you can see the rest of the filenames. If you press ESCAPE or RESET, it all starts over collecting volume names. If you press RETURN when the cursor bar is on a DIR file, the directory name will be added to the current prefix and a new filename list will appear.

If you press RETURN when the cursor bar is on a SYS file, lines 1950-1990 will load the system file and start it running. Lines 1950-1960 set the system prefix to the directory the system file is in. Lines 1970-1980 read the file into RAM starting at $2000, and if there are no errors we blast-off with a JMP $2000. If there ARE errors, the program just starts over.

```
1000 *SAVE NEW.QUIT.CODE
1010 *--------------------------------
1020 *    Installation:
1030 *      1.  BLOAD PRODOS,TSYS,A$2000
1040 *      2.  BLOAD B.NEW.QUITTER,A$5700
1050 *      3.  BSAVE PRODOS,TSYS,A$2000,L$3A00
1060 *    Location:
1070 *      In PRODOS file:   $5700-59FF
1080 *      In ProDOS image:  $D100-D3FF
1090 *      For execution:    $1000-12FF
1100 *--------------------------------
1110 *    Code which downloads the QUIT code resides at
1120 *      $FCE5-FD3A.  This is loaded from $4BE5-4C3A.
1130 *--------------------------------
1140 BPNTR  .EQ $00,01
1150 SPNTR  .EQ $02,03
1160 DPNTR  .EQ $04,05
1170 CV     .EQ $25
```

```
1180 INVFLG .EQ $32
1190 *--------------------------------
1200 HOME   .EQ $FC58
1210 CLREOL .EQ $FC9C
1220 COUT   .EQ $FDED
1230 CROUT  .EQ $FD8E
1240 *--------------------------------
1250 MLI    .EQ $BF00
1260 BITMAP .EQ $BF58
1270 *--------------------------------
1280        .OR $1000
1290        .TF B.NEW.QUITTER
1300 *--------------------------------
1310 QUITTER
1320        LDA $C082    MOTHERBOARD ROMS
1330        LDX #$16
1340        LDA #0       PREPARE VIRGIN BITMAP
1350 .1     STA BITMAP,X
1360        DEX
1370        BNE .1
1380        INX          X=1, LOCKOUT $BF00 PAGE
1390        STX BITMAP+$17
1400        LDA #$CF
1410        STA BITMAP
1420 *---LIST VOLUME NAMES------------
1425 .2     LDA #$99     CTRL-Y
1430        JSR $C300    SET I/O HOOKS, 80-COL MODE, CLEAR SCREEN
1440        LDY #Q.SDV
1450        JSR MSG
1460        JSR CLOSE.ALL.FILES
1470        JSR MLI
1480        .DA #$C5,ONLINE
1490        LDY #0
1500        STY MAX.DIRPNT
1510        STY DIR.START
1520        STY PATHNAME
1530 .3     STY SEL.LINE
1540        JSR DISPLAY.VOLUMES
1550        LDY #Q.VHELP
1560        JSR MSG
1570        JSR GET.KEY
1580        BCC .3       ...ARROW KEYS
1590        BNE .2       ...ESCAPE KEY
1600 *---READ DIRECTORY---------------
1610 .4     JSR READ.THE.FILE
1620        BCS .7
1630 *---PRINT PATHNAME---------------
1640        JSR HOME
1650        LDY #0
1660 .5     LDA PATHNAME+1,Y
1670        ORA #$80
1680        JSR COUT
1690        INY
1700        CPY PATHNAME
1710        BCC .5
1720 *---COLLECT FILENAMES------------
```

```
1730        LDX #0
1740        LDA #$FF          FIRST JUST "SYS" FILES
1750        JSR SCAN.DIRECTORY
1760        LDA #$0F          THEN JUST "DIR" FILES
1770        JSR SCAN.DIRECTORY
1771        TXA          SEE IF ANY FILES FOUND
1772        BEQ .2       ...NO, BACK TO THE TOP
1780        LDA #0       MARK END OF LIST
1790        STA DIRBUF+256,X
1800        STX MAX.DIRPNT
1810 *---LIST THE FILENAMES-----------
1820        TAY          Y=0
1830        STY DIR.START
1840 .6     STY SEL.LINE
1850        JSR DISPLAY.FILES
1860        LDY #Q.VHELP
1870        JSR MSG
1880        JSR GET.KEY
1890        BCC .6       ...ARROW KEYS
1900        BNE .2       ...ESCAPE KEY
1910        LDY #$10
1920        LDA (SPNTR),Y     GET FILE TYPE
1930        BPL .4            DIRECTORY ($0F)
1940 *---SYS FILE, LOAD & EXECUTE-----
1950        JSR MLI      SET PREFIX
1960        .DA #$C6,PATH
1970        JSR READ.THE.FILE
1980        BCS .7       ...ERROR IN READING
1990        JMP BUFFER
2000 .7     JMP QUITTER
2010 *--------------------------------
2020 READ.THE.FILE
2030        LDY #0       APPEND CURRENTLY SELECTED NAME
2040        LDA (SPNTR),Y     GET LENGTH OF NAME
2050        AND #$0F
2060        STA LENGTH
2070        LDX PATHNAME      CURRENT LENGTH
2080        LDA #'/'
2090 .1     INX
2100        INY
2110        STA PATHNAME,X
2120        LDA (SPNTR),Y
2130        DEC LENGTH
2140        BPL .1
2150        STX PATHNAME
2160        JSR MLI           OPEN THE FILE
2170        .DA #$C8,OPEN
2180        BCS RF.ERR
2190        LDA O.REF         FILE REFERENCE NUMBER
2200        STA R.REF
2210        JSR MLI           READ THE WHOLE FILE
2220        .DA #$CA,READ
2221        BCC CLOSE.ALL.FILES
2222        CMP #$4C          IS IT JUST EOF?
2223        SEC
2230        BNE RF.ERR        ...NO
```

```
2240 CLOSE.ALL.FILES
2250        JSR MLI           CLOSE THE FILE
2260        .DA #$CC,CLOSE
2270 RF.ERR RTS
2280 *--------------------------------
2290 SCAN.DIRECTORY
2300        STA CURTYP        TYPE WE ARE COLLECTING
2310        LDA #0            START WITH FIRST BLOCK
2320 .1     STA CURBLK
2330        LDA #BUFFER+4     FIRST 4 BYTES OF BLOCK SKIPPED
2340        STA DPNTR
2350        CLC               COMPUTE PAGE OF PNTR
2360        LDA /BUFFER+4
2370        ADC CURBLK
2380        STA DPNTR+1
2390        LDA ENTCNT
2400        STA LENGTH
2410 *--------------------------------
2420 .2     LDY #0
2430        LDA (DPNTR),Y
2440        AND #$F0
2450        BEQ .4       ...DELETED FILE
2460        CMP #$E0     ...HEADER?
2470        BCS .4       ...YES
2480        LDY #$10
2490        LDA (DPNTR),Y     LOOK AT FILE TYPE
2500        CMP CURTYP
2510        BNE .4            ...NOT CURRENT TYPE
2520 *---DIR or SYS file--------------
2530 .3     LDA DPNTR
2540        STA DIRBUF,X
2550        LDA DPNTR+1
2560        STA DIRBUF+256,X
2570        INX
2580 *---ADVANCE TO NEXT ENTRY--------
2590 .4     CLC
2600        LDA DPNTR
2610        ADC ENTLEN
2620        STA DPNTR
2630        BCC .5
2640        INC DPNTR+1
2650 .5     DEC LENGTH        AT END OF BLOCK YET?
2660        BNE .2            ...NO, CONTINUE IN BLOCK
2670        CLC
2680        LDA CURBLK
2690        ADC #2
2700        CMP ACTLEN+1
2710        BCC .1            ...YES, READ NEXT BLOCK
2720 *--------------------------------
2730        RTS
2740 *--------------------------------
2750 CLOSE  .DA #1,#0
2760 ONLINE .DA #2,#0,BUFFER
2770 OPEN   .DA #3,PATHNAME,OPNBUF
2780 O.REF  .BS 1
2790 READ   .DA #4
```

```
2800 R.REF  .BS 1
2810        .DA BUFFER,$9F00
2820 ACTLEN .BS 2
2830 PATH   .DA #1,PATHNAME
2840 *--------------------------------
2850 DISPLAY.VOLUMES
2860        JSR SETUP.DISPLAY.LOOP
2870        LDA #BUFFER
2880        STA BPNTR
2890        LDA /BUFFER
2900        STA BPNTR+1
2910 *--------------------------------
2920 .1     LDY #0
2930        LDA (BPNTR),Y
2940        AND #$0F
2950        BEQ .3       ...NO VOLUME HERE
2960 *--------------------------------
2970        JSR CHECK.FOR.SEL.LINE
2980 *--------------------------------
2990 .2     LDA (BPNTR),Y     GET UNIT NUMBER
3000        LSR               ISOLATE SLOT NUMBER
3010        LSR
3020        LSR
3030        LSR
3040        AND #7
3050        ORA #"0"
3060        JSR COUT          PRINT SLOT NUMBER
3070        LDA #"/"
3080        JSR COUT
3090        LDA (BPNTR),Y     GET UNIT NUMBER AGAIN
3100        ASL               SET CARRY IF DRIVE 2
3110        LDA #"1"          ASSUME DRIVE 1
3120        ADC #0            CHANGE TO 2 IF TRUE
3130        JSR COUT
3140        LDA #" "     PRINT TWO SPACES
3150        JSR COUT
3160        JSR COUT
3170        JSR PRINT.BPNTR.NAME
3180 *--------------------------------
3190 .3     CLC               POINT TO NEXT VOLUME NAME
3200        LDA BPNTR
3210        ADC #16
3220        STA BPNTR
3230        BCC .4
3240        INC BPNTR+1
3250 .4     DEC LENGTH        ANY MORE LEFT?
3260        BNE .1            ...YES
3270        RTS
3280 *--------------------------------
3290 PRINT.BPNTR.NAME
3300        LDY #0
3310        LDA (BPNTR),Y      GET NAME LENGTH
3320        AND #$0F
3330        TAX
3340 .1     INY          PRINT THE VOLUME NAME
3350        LDA (BPNTR),Y
```

```
       ORA #$80
       JSR COUT
       DEX
       BNE .1
*--------------------------------
.2     LDA #" "     PRINT TRAILING BLANKS
       JSR COUT
       INY
       CPY #16
       BCC .2
       LDA #$FF     NORMAL MODE NOW
       STA INVFLG
       INC MAX.LINE      COUNT THE LINE
       JMP CROUT
*--------------------------------
GET.KEY
.1     LDA $C000    READ KEY FROM KEYBOARD
       BPL .1
       STA $C010    CLEAR THE STROBE
       CMP #$8D
       BEQ .2       <RETURN>
       CMP #$88     <--
       BEQ .3
       CMP #$95     -->
       BEQ .7
       CMP #$8A     DOWN ARROW
       BEQ .7
       CMP #$8B     UP ARROW
       BEQ .3
       CMP #$9B     ESCAPE
       BNE .1       GET ANOTHER CHARACTER
       LDA #$9B     ...SET .NE.
.2     RTS
*---<UP OR LEFT ARROW>-----------
.3     LDY SEL.LINE      CURRENT BRIGHT LINE
       BNE .6            ...NOT TOP LINE
       LDY DIR.START     ARE WE DISPLAYING THE FIRST ONE?
       BEQ .5            ...YES
       DEC DIR.START     ...NO, MOVE TOWARD FIRST LINE
.4     LDY #0            MAKE FIRST LINE BRIGHT
       CLC
       RTS
.5     LDY MAX.LINE      MAKE LAST LINE BRIGHT
.6     DEY
       CLC
       RTS
*---<DOWN OR RIGHT ARROW>--------
.7     LDY SEL.LINE      CURRENT BRIGHT LINE
       INY               MOVE TOWARD LAST LINE
       CPY MAX.LINE      BEYOND END OF SCREEN?
       BCC .8            ...NO
       LDA MAX.DIRPNT    ...YES, CHECK IF SHOWING LAST LINE
       SBC #17
       BCC .4            ...YES
       CMP DIR.START
       BCC .4            ...YES
```

```
3920        INC DIR.START     ...NO, MOVE TOWARD LAST LINE
3930        LDY SEL.LINE
3940        CLC
3950 .8     RTS
3960 *--------------------------------
3970 DISPLAY.FILES
3980        JSR SETUP.DISPLAY.LOOP
3990        LDA DIR.START
4000        STA DIR.INDEX
4010        JSR CLEAR.LINE.OR.PRINT.MORE.MSG
4020 *--------------------------------
4030 .1     LDX DIR.INDEX
4040        LDY DIRBUF+256,X
4050        BEQ .4       ...END OF LIST
4060        STY BPNTR+1
4070        LDA DIRBUF,X
4080        STA BPNTR
4090        JSR CHECK.FOR.SEL.LINE
4100 *--------------------------------
4110 .2     LDY #$10
4120        LDA (BPNTR),Y
4130        BMI .3       ...SYS FILE
4140        LDY #Q.DIR
4150        .HS 2C
4160 .3     LDY #Q.SYS
4170        JSR MSG
4180        JSR PRINT.BPNTR.NAME
4190 *--------------------------------
4200        INC DIR.INDEX
4210        DEC LENGTH
4220        BNE .1
4230 .4     LDA DIR.INDEX
4240        CMP MAX.DIRPNT
4250 *--------------------------------
4260 CLEAR.LINE.OR.PRINT.MORE.MSG
4270        BEQ .1       CLEAR LINE
4280        LDY #Q.MORE
4290        BNE MSG      ...ALWAYS
4300 .1     JSR CLREOL
4310        JMP CROUT
4320 *--------------------------------
4330 SETUP.DISPLAY.LOOP
4340        LDA #16      MAX 16 LINES IN LIST
4350        STA LENGTH
4360        LDY #0
4370        STY MAX.LINE
4380        INY          SAME AS VTAB 3, HTAB 1
4390        STY CV
4400        JMP CROUT
4410 *--------------------------------
4420 CHECK.FOR.SEL.LINE
4430        LDA MAX.LINE      SEE IF CURRENT LINE SHOULD
4440        CMP SEL.LINE           BE INVERSE MODE
4450        BNE .1            ...NO
4460        LDA BPNTR         ...YES, SO SETUP POINTER
4470        STA SPNTR
```

```
4480        LDA BPNTR+1
4490        STA SPNTR+1
4500        LDA #$3F                 & SET INVERSE MODE
4510        STA INVFLG
4520 .1     RTS
4530 *--------------------------------
4540 MSG1   JSR COUT
4550        INY
4560 MSG    LDA QTS,Y
4570        BNE MSG1
4580        RTS
4590 *--------------------------------
4600 QTS    .EQ *
4610 Q.SDV  .EQ *-QTS
4620        .AS -"S/D  VOLUME NAME"
4630        .HS 00
4640 Q.VHELP .EQ *-QTS
4650        .HS 8D
4660        .AS -/USE ARROWS AND <RETURN> TO SELECT/
4670        .HS 8D
4680        .AS -/USE <ESCAPE> TO TRY AGAIN/
4690        .HS 8D
4700        .HS 00
4710 Q.SYS  .EQ *-QTS
4720        .AS -/SYS -- /
4730        .HS 00
4740 Q.DIR  .EQ *-QTS
4750        .AS -/DIR -- /
4760        .HS 00
4770 Q.MORE .EQ *-QTS
4780        .AS -/<<<MORE>>>/
4790 Q.CR   .EQ *-QTS
4800        .HS 8D00
4810 *--------------------------------
4820            .DUMMY
4830            .OR $800
4840 OPNBUF     .BS 1024
4850 DIRBUF     .BS 512
4860 PATHNAME   .EQ $280
4870 DIR.INDEX  .BS 1
4880 DIR.START  .BS 1
4890 MAX.DIRPNT .BS 1
4900 SEL.LINE   .BS 1
4910 MAX.LINE   .BS 1
4920 UNIT       .BS 1
4930 LENGTH     .BS 1
4940 CURTYP     .BS 1
4950 CURBLK     .BS 1
4960            .ED
4970 *--------------------------------
4980 BUFFER     .EQ $2000
4990 ENTLEN     .EQ BUFFER+$23   ENTRY LENGTH
5000 ENTCNT     .EQ BUFFER+$24   # ENTRIES PER BLOCK
5010 *--------------------------------
```

```
################################################################################
#  The ProDOS QUIT-code Installer
################################################################################
```

Bob Sander-Cederlof

July 1986

As I just mentioned, the code which downloads the QUIT-code from $D100-D3FF to $1000-12FF is located at $FCE5 inside ProDOS 1.1.1. Here is a commented listing of that code.

```
  1000 *SAVE S.FCE5.FD3A
  1010 *--------------------------------
  1020        .OR $FCE5
  1030        .TA $800
  1040 DOWNLOAD.QUITTER
  1050        LDA $C083    SWITCH IN CORRECT D000 BANK
  1060        LDA $C083
  1070 *--------------------------------
  1080        LDA $00      SAVE 00...03 ON STACK
  1090        PHA
  1100        LDA $01
  1110        PHA
  1120        LDA $02
  1130        PHA
  1140        LDA $03
  1150        PHA
  1160 *---SETUP POINTERS FOR MOVING----
  1170        LDA /$1000   Destination Pointer
  1180        STA $03
  1190        LDA /$D100   Source Pointer-
  1200        STA $01
  1210        LDA #0
  1220        STA $00
  1230        STA $02
  1240 *--------------------------------
  1250        TAY          Y=0
  1260        LDX #3       Move 3 Pages
  1270 .1     DEY
  1280        LDA ($00),Y
  1290        STA ($02),Y
  1300        TYA
  1310        BNE .1       ...More in same page
  1320        INC $01
  1330        INC $03      Advance to next pages
  1340        DEX          Count the page
  1350        BNE .1       ...Copy another page
  1360 *---Restore $03...$00------------
  1370        PLA
  1380        STA $03
  1390        PLA
  1400        STA $02
  1410        PLA
  1420        STA $01
```

```
1430        PLA
1440        STA $00
1450 *--------------------------------
1460        LDA $C08B    Select normal D000 bank
1470        LDA $C08B
1480 *---Set up RESET Vector----------
1490        LDA #$1000   Lo-byte
1500        STA $3F2
1510        LDA /$1000   Hi-byte
1520        STA $3F3
1530        EOR #$A5     Power-up byte
1540        STA $3F4
1550 *--------------------------------
1560        JMP $1000
1570 *--------------------------------
```

The program above can be written in a lot less space, as follows:

```
1580
1590 *--------------------------------
1600        .OR $FCE5
1610        .TA $900
1620 SC.DOWNLOAD.QUITTER
1630        LDA $C083    Select D000 bank
1640 *--------------------------------
1650        LDY #0
1660 .1     LDA $D100,Y
1670        STA $1000,Y
1680        LDA $D200,Y
1690        STA $1100,Y
1700        LDA $D300,Y
1710        STA $1200,Y
1720        INY
1730        BNE .1
1740 *--------------------------------
1750        LDA $C08B    Select normal D000 bank
1760 *---Set up RESET Vector----------
1770        STY $3F2     RESET Vector Lo-byte
1780        LDA /$1000   Hi-byte
1790        STA $3F3
1800        EOR #$A5     Power-up byte
1810        STA $3F4
1820 *--------------------------------
1830        JMP $1000
1840 *--------------------------------
```

##
Using DP18 Under ProDOS
##

Bill Morgan

July 1986

A customer called up the other day to order the DP18 Source Code package, but he wanted it only if it ran under ProDOS. (That's 18-digit Binary Coded Decimal arithmetic for Applesoft.) Well, we hadn't tried to move it over before, but it didn't sound like too much of a problem, so I gave it a shot. It did turn out to be quite easy.

I first tried simply CONVERTing all the files over, including the binary object code, and RUNning the example programs. That almost worked! The DP18 arithmetic all operated just right, but the scheme of moving the Applesoft program up and BLOADing DP18 at $803 ran into a little trouble. The forward pointers in each line of the program weren't set up properly. Bob then pointed out to me that it's very easy to install a program between BASIC.SYSTEM and the buffers, so that might be the way to go in this situation. All it took was a little arithmetic to figure out that DP18 needs $1C pages and should therefore have an origin of $7E00. The .OR directive was the only line inside DP18 that I had to change!

After that I needed only two more things: a short machine language program to get the buffer from BI, issue the BLOAD command, and set the ampersand vector; and a one-line Applesoft routine that checks the vector to find out if DP18 is already installed and call the loader if not.

Here's the Applesoft routine:

```
 10  IF  PEEK (1014) + 256 *  PEEK (1015) <  > 32563 THEN
    PRINT  CHR$ (4)"BRUN INSTALL.DP18"
```

And here's all there is to the loader:

```
  1000 *SAVE S.INSTALL.DP18
  1010 *--------------------------------
  1020 BUFFER      .EQ $200
  1030
  1040 AMPERSAND   .EQ $3F6
  1050
  1060 DP.LINK     .EQ $7E00
  1070 AMP.LINK    .EQ $7E02
  1080
  1090 DOS.COMMAND .EQ $BE03
  1100 GET.BUFFER  .EQ $BEF5
  1110 FREE.BUFFER .EQ $BEF8
  1120
  1130 COUT1       .EQ $FDF0
  1140 *--------------------------------
  1150        .OR $300
  1160 *      .TF INSTALL.DP18
  1170
  1180 T      JSR FREE.BUFFER   kick others out
  1190        LDA #$1C
```

```
        JSR GET.BUFFER    get 28 pages
        BCS ERROR
        CMP #$7E          must be at $7E00
        BNE ERROR

        LDX #LENGTH
.1      LDA COMMAND,X     "BLOAD DP18"
        STA BUFFER,X
        DEX
        BPL .1
        JSR DOS.COMMAND   do it
        BCS ERROR

        LDX #1
.2      LDA AMPERSAND,X   save old vector
        STA AMP.LINK,X
        LDA DP.LINK,X     & point to DP18
        STA AMPERSAND,X
        DEX
        BPL .2
EXIT    RTS

ERROR   LDX #0
.1      LDA MESSAGE,X     show error message
        BEQ EXIT
        JSR COUT1
        INX
        BNE .1
*--------------------------------
MESSAGE .HS 8D
        .AS -/Error loading DP18/
        .HS 8D00

COMMAND .AS -/BLOAD DP18/
        .HS 8D
LENGTH  .EQ *-COMMAND-1
*--------------------------------
```

```
################################################################################
#  Updated Memory vs.File Maps for ProDOS
################################################################################
```

Bob Sander-Cederlof

August 1986

I am not sure how it happened, but I seem to have botched up the table on page 20 of the November 1985 issue. As I now understand it, the relationship between the PRODOS file image (which loads at $2000) and the image of ProDOS after it is loaded is as follows (the lines marked with * are the changed lines):

```
       2000-287E     ProDOS Installer Code
       287F-28FE     zeroes
       28FF-293C     Installer for /RAM Driver
       293D-29FF     zeroes

       2A00-2BFF     Aux 200-3FF     /RAM/ Driver
     * 2C00-2C99     FF00.FF99       /RAM/ Driver
       2C7F-2CFF                     zeroes
     * 2D00-4DFF     DE00-FEFF       MLI Kernel
       4E00-4EFF     BF00-BFFF       System Global Page
     * zeroes        D700-DDFF
       4F00-4F7C     D742-D7BE       Thunderclock driver
       4F80-4FFF     FF80-FFFF       Interrupt Code
     * 5000-56FF     D000-D6FF       Device Drivers
       5700-59FF     Alt D100-D3FF   QUIT Code
```

Looking at the same information from the viewpoint of the finished product, here is a map of ProDOS after it is loaded:

```
       4E00-4EFF     BF00-BFFF       System Global Page
     * 5000-56FF     D000-D6FF       Device Drivers
     * zeroes        D700-DDFF
       4F00-4F7C     D742-D7BE       Thunderclock driver
     * 2D00-4DFF     DE00-FEFF       MLI Kernel
     * 2C00-2C99     FF00-FF99       /RAM/ Driver
       4F80-4FFF     FF80-FFFF       Interrupt Code

       5700-59FF     Alt D100-D3FF   QUIT Code
```

```
################################################################################
#  Compatibility with the Laser-128
################################################################################
```

Bob Sander-Cederlof

August 1986

We borrowed a Laser-128 (popular clone of the Apple //c) the other day. It had been rumored that our software would not run on it, in spite of Central Point Software's sanguine claims. Sure enough, the S-C Macro Assembler would not operate, under either DOS or ProDOS. They boot and load, but no more.

A little investigation revealed what we expected: our software uses at least a half-dozen entry points into the Apple monitor which are not supported in the Laser-128 monitor. Most of them have to do with our "$" command, which lets you perform monitor commands without leaving the S-C environment. These patches will disable the "$" command and repair the "MEM" command. The addresses shown are for our current release disks.

```
DOS 3.3 $1000 version   1AE6:4C B3 1B 20 40 F9 A9 AD 4C ED FD
                        124A:E9 1A  (was 99 FD)
                        125D:E9 1A  (was 99 FD)

DOS 3.3 $D000 version   DAE6:4C B3 DB 20 40 F9 A9 AD 4C ED FD
                        D24A:39 DA  (was 99 FD)
                        D25D:E9 DA  (was 99 FD)

ProDOS version          8B45:4C 24 8C 20 40 F9 A9 AD 4C ED FD
                        8450:48 8B  (was 99 FD)
                        8463:48 8B  (was 99 FD)
```

Make a backup copy of the disk, and then boot the backup copy. When the assembler version you choose has loaded, type the letter X and the RETURN key. This should BRK out of the assembler into the Laser-128 monitor. Make the patches as shown above, and then type "3D0G" or control-RESET to get back into the assembler. It should be working correctly now. If you are fixing the DOS 3.3 version, you can now BSAVE the patched code on the file you originally loaded.

If you are fixing the ProDOS version, you now should BLOAD the type SYS file called SCASM.SYSTEM. The same patches you just made to the assembler should now be applied to the image of the SYS file, and then BSAVE the image on the disk:

```
       :BLOAD SCASM.SYSTEM,TSYS,A$2000
       :MNTR
       *2D45:4C 24 8C 20 40 F9 A9 AD 4C ED FD
       *2650:48 8B
       *2663:48 8B
       *3D0G
       :BSAVE SCASM.SYSTEM,TSYS,A$2000,L17920
```

One incompatibility remains for which we never found the cause: the esc-L shorthand command, to turn a CATALOG line into a LOAD command, does not work in 80-column mode. It does work just fine in 40-column mode. If any of you try these patches and find other problems, we would like to hear about them.

One more item: we found the Laser-128 monitor incorrectly disassembles the PLX command as PHX.

##
Thoughts on the ProDOS Bit Map
##

Louis Pitz

September 1986

I recently learned some more about ProDOS, the hard way. Yes, sometimes catastrophe is indeed the mother of invention, or at least of learning. I was trying to finish typing and saving a program when an electrical storm started. When I did a CATALOG, all the files seemed to be okay, but the footer info at the end about blocks free, used, and total was goofed up. Where I expected 86, 58, and 144, there was instead 681, 64999, and 144.

As an aside, there were only 144 total blocks because the disk is a combination of ProDOS and DOS 3.3, as described in AAL Sep 85 (page 11). But the lesson I learned would apply on regular ProDOS-only disks as well.

Note the logic in the goofed-up numbers: 681+64999 = 144 mod 65536. I suspected that, since everything else was okay, the volume bit map had been messed up. So I inspected the blocks on disk and confirmed my suspicion.

Further, the garbage in the volume bit map block was clearly extraneous, and none of the the good data (the first 144/8=18 bytes) had been changed. The garbage was $DC's in bytes $14A-1C0, inclusive. This is way past the end of the 'real' bytes even for a ProDOS-only disk (35 bytes). But ProDOS must have counted the 1-bits in the $DC bytes as free blocks. Then, subtracting this erroneously large number from 144, it got 64999. Yes! $DC=%11011100, and there are $77=119 such bytes, so that is 5*119=595 more "free" blocks to add to the 86 really free to get 681.

I've read Sandy Mossberg's article about the ProDOS CAT and CATALOG commands (Nibble, May 86), but the arithmetic counting used sectors must be buried deep in the MLI, associated with the GET-FILE-INFO call, according to my Beneath Apple ProDOS book. Apparently ProDOS must count all the 1-bits in the volume bit map blocks as free, regardless of the number of total blocks on the disk. In a way this seems like a bug, but I guess it was just a shortcut in coding.

The lesson I have learned is not to use the "unused" part of the volume bit map to store code, messages, or anything. For a ProDOS-only floppy, only 35 bytes are really used, and 477 bytes are wasted. Nevertheless, do not be tempted to use them. They are set to 0 upon formatting the disk, and ProDOS depends upon them staying that way! I've used the extra bytes in the DOS 3.3 VTOC before, but I had better resist this impulse in ProDOS.

```
################################################################################
#  New ProDOS Bug and Fix
################################################################################
```

Bob Sander-Cederlof

November 1986

The November 1986 issue of Open-Apple (Tom Weishaar's wonderful newsletter) tells of an important new discovery. For about a year Tom has been reporting on the symptom: Appleworks and Applewriter data disks suddenly turning up with track 0 destroyed. It only happened to 5.25" diskettes, and only one certain machines, and otherwise seemingly at random. For a complete description, get all of Tom's back issues.

Some of his readers from Australia seem to have tracked down the problem, and they suggest a solution. In the floppy driver code inside ProDOS, at $D6C3, there are four STA commands that turn off all four stepper motor windings. Tom says the purpose is to disable any 3.5" drives connected in a daisy chain to the same controller. I wonder, because this code has been here since 1983, long before the possiblility of 3.5" drives. Anyway, the code has a bad side-effect in some systems.

A quirk of the controller card is that STA operations to the stepper motor winding soft-switches also cause the card to write on the data bus. So you have the bus being driven in two directions at once: the cpu trying to store the A-register, and the controller card trying to send something meaningless. Besides resulting in garbage on the data bus, which causes no real damage in this case, apparently in some Apples with some controller cards it causes the card to go into WRITE mode. Whatever track the head is sitting on will then be clobbered.

The solution is to change the four STA operations to LDA. The disk drives will get the same message, without causing the bus contention. You can patch the PRODOS system file and re-SAVE it, on all your disks. If you have a hard disk, you should only have to do it one time. If you BLOAD the PRODOS file at $2000, the four instructions will be found at $56D3:

```
       56D3: 9D 80 C0  STA $C080,X
       56D6: 9D 82 C0  STA $C082,X
       56D9: 9D 84 C0  STA $C084,X
       56DC: 9D 86 C0  STA $C086,X
```

If you change all those "9D" bytes to "BD", which is the opcode for "LDA addr,X", the bug is supposed to disappear. Doing it from inside the S-C Macro Assembler, I did it this way:

```
       :BLOAD PRODOS,TSYS,A$2000
       :UNLOCK PRODOS
       :$56D3:BD N 56D6:BD N 56D9:BD N 56DC:BD
       :BSAVE PRODOS,TSYS,A$2000,L14848
       :LOCK PRODOS
```

I personally have never had ProDOS clobber a diskette. I have trashed some myself, by stupidity, but this hardware/software bug has never caused it. Nevertheless, I have now patched my disks, just in case. Many thanks to Tom, Open-Apple, and to the men in Australia.

##
New ProDOS Book: ProDOS Inside and Out
##

December 1986

Dennis Doms and Tom Weishaar, Technical Consultant and Publisher of Open-Apple, have conspired to bring us an interesting new book on programming under ProDOS, especially focussing on BASIC.SYSTEM.

"ProDOS Inside and Out" begins by explaining what an operating system is, progresses by describing files and directories, and goes on into simple commands. The next sections cover Applesoft programming and text file handling, followed by information about using machine language under BASIC.SYSTEM and using the ProDOS Kernel and MLI calls from BASIC.

This book does an excellent job of introducing the basic concepts of ProDOS, and then takes the reader on into quite advanced territory. It's very refreshing to find a book that doesn't assume you're already an expert and still has enough substance to help make you into one.

"ProDOS Inside and Out", by Dennis Doms and Tom Weishaar, from TAB Books. List is $16.95, we'll have it for $16 + shipping.

```
################################################################################
#  Commented Listing of ProDOS -- $DE00-DEF2
################################################################################
```

Bob Sander-Cederlof

December 1986

What happens when you call ProDOS MLI? In assembly language, MLI calls look like this:

```
       JSR $BF00
       .DA #command,IOB.Address
```

The instruction at $BF00 is a "JMP $BFB7" in ProDOS 1.1.1; it is possibly different in other versions. All of the following disassembly is for ProDOS 1.1.1. The changes in the new ProDOS 1.2 are minor, and if you have 1.2 you should be able to figure out what they are.

At $BFB7 there is some code I call LC.BRIDGE.ENTRY. It "remembers" what language card areas are switched in at $D000 and at $E000, and then turns on the language card so that it can jump into the MLI call processor.

```
       BFB7:  SEC      Set flag
              ROR MLI.ACTIVE.FLAG
              LDA $E000
              STA E000.BYTE  (BFF4)
              LDA $D000
              STA D000.BYTE  (BFF5)
              LDA $C08B
              LDA $C08B
              JMP $DE00
```

Now comes the good part. The following listing is of the code starting at $DE00, which decodes the bytes following your JSR $BF00 and performs your request.

Lines 1010-1080 define some page-zero variables used by MLI. Lines 1090-1220 define some items in the system global page. Lines 1230-1280 define some entry points inside the rest of MLI, not listed here.

MLI calls don't change the X and Y registers, so they are saved at line 1390. The return address (of the JSR $BF00) is pulled off the stack and saved at PARM.PNTR in page zero, so that it can be used to access your command code and IOB address. Lines 1410-1490 also compute the address of the next instruction, to be used later for a return address. This address is saved in the system global page, and is useful sometimes for debugging. (We have published several articles on enhanced error messages and tracers for MLI calls in previous issues of AAL.)

Lines 1500-1650 convert the command code to an index by a strange scheme. The legal command codes are (in hex): 40, 41, 65, 80 thru 82, and C0 thru D3. The hashing algorithm used here adds the high nybble of the command code to the whole code, and then masks it to the lower five bits. This compresses the range of the codes, without any overlapping.

```
       40,41 --> 04,05         C0-CF --> 0C-1B
       65    --> 0B            D0-D2 --> 1D-1F
```

```
80-82 --> 08-0A        D3    --> 00
```

This index is used then to look into the COMMAND.HASH.TABLE, which has the actual command codes in the indexed positions. If the original code is not found there, then the original code was an illegal command number. The hash index is also used to look up the parameter count in PARM.CNT.TABLE. I have appended the code for these two tables to the end of today's listing, at lines 3100 to the end.

Lines 1810-1920 branch various ways according to the command code. Most of the commands are not shown in this listing, but most of the code for READ BLOCK and WRITE BLOCK is shown (lines 2690-3080). When a command is finished, it eventually finds its way back to EXIT.TO.CALLER at line 2180.

Lines 2180-2560 get us back to our own code again, after the JSR $BF00. If the MLI call produced an error, the code number for that error will be in SYS.ERRNUM. The error code will be returned in the A-register, with carry SET. If there is no error to report, A=0 and carry is clear.

We will probably be presenting more sections of MLI disassembly in the near future. You may remember that we published portions of an earlier ProDOS version back in November and December of 1983.

```
1000 *SAVE S.MLI.DE00.DEF2
1010 *--------------------------------
1020 PARM.PNTR  .EQ $40,41
1030 COMMAND    .EQ $42
1040 UNIT.NO    .EQ $43
1050 BUFF.PNTR  .EQ $44,45
1060 BLOCK.NO   .EQ $46,47
1070 GEN.PNTR1  .EQ $48,49
1080 GEN.PNTR2  .EQ $4E,4F
1090 *--------------------------------
1100 CALL.QUIT           .EQ $BF03
1110 CALL.TIME           .EQ $BF06
1120 CALL.SYSERR         .EQ $BF09
1130 SYS.ERRNUM          .EQ $BF0F
1140 DRIVER.ADDR.TABLE   .EQ $BF10 thru BF2F
1150 BACKUP.BIT          .EQ $BF95
1160 MLI.ACTIVE.FLAG     .EQ $BF9B
1170 MLI.RETURN          .EQ $BF9C,D
1180 MLI.X               .EQ $BF9E
1190 MLI.Y               .EQ $BF9F
1200 LC.BRIDGE.EXIT      .EQ $BFA0
1210 E000.BYTE           .EQ $BFF4
1220 D000.BYTE           .EQ $BFF5
1230 *--------------------------------
1240 INTERRUPT.HANDLER   .EQ $DEF3
1250 FILING.FUNCTIONS    .EQ $E047
1260 CHECK.IF.MEM.FREE   .EQ $FC9F
1270 *--------------------------------
1280 JUMP                .EQ $FEF5,6
1290 *--------------------------------
1300        .OR $DE00
1310        .TA $800
1320 *--------------------------------
1330 *   JSR $BF00 comes here
```

```
*    .DA #$xx  command byte
*    .DA xxxx  IOB Address
*--------------------------------
MLI.ENTRY
       CLD
       STY MLI.Y
       STX MLI.X
       PLA          GET RETURN ADDRESS
       STA PARM.PNTR     WILL POINT AT BYTES
       CLC               FOLLOWING JSR $BF00
       ADC #4       COMPUTE ACTUAL RETURN
       STA MLI.RETURN    AND SAVE FOR LATER
       PLA
       STA PARM.PNTR+1
       ADC #0
       STA MLI.RETURN+1
*---Check Command Code-----------
       LDY #0
       STY SYS.ERRNUM
       INY
       LDA (PARM.PNTR),Y
       LSR          Hash it (CC/16 + CC) & $1F
       LSR
       LSR
       LSR
       CLC
       ADC (PARM.PNTR),Y
       AND #$1F
       TAX          Use hashcode as index
       LDA (PARM.PNTR),Y  Original command code
       CMP COMMAND.HASH.TABLE,X
       BNE ERR.CALL.NO    Not valid command
*---Get IOB Address--------------
       INY
       LDA (PARM.PNTR),Y
       PHA
       INY
       LDA (PARM.PNTR),Y
       STA PARM.PNTR+1
       PLA
       STA PARM.PNTR
*---Check Parm Count-------------
       LDY #0
       LDA PARM.CNT.TABLE,X
       BEQ MLI.GETTIME   ...only one with 0 parms
       CMP (PARM.PNTR),Y
       BNE ERR.PARM.CNT
*---Branch Various Ways----------
       LDA COMMAND.HASH.TABLE,X
       CMP #$65
       BEQ .1       ...QUIT CALL
       ASL
       BPL MLI.RWBLK        $80 or $81
       BCS MLI.CX.AND.DX    $Cx or $Dx
       LSR                  $40 or $41
       AND #$03
```

```
1900        JSR INTERRUPT.HANDLER
1910        JMP EXIT.TO.CALLER
1920 .1     JMP CALL.QUIT        $65
1930 *--------------------------------
1940 *   Command $82, Get the Date and Time
1950 *--------------------------------
1960 MLI.GETTIME
1970        JSR CALL.TIME
1980        JMP EXIT.TO.CALLER
1990 *--------------------------------
2000 *   Commands $80 and $81
2010 *--------------------------------
2020 MLI.RWBLK
2030        LSR          Make $00 and 01
2040        ADC #1       Into $01 and 02
2050        STA COMMAND  Store into command block
2060        JSR BLOCK.IO.SETUP  Do the I/O
2070        JMP EXIT.TO.CALLER
2080 *--------------------------------
2090 *   Commands $C0 thru $D3
2100 *--------------------------------
2110 MLI.CX.AND.DX
2120        LSR          Make command code into
2130        AND #$1F          an index
2140        TAX
2150        JSR FILING.FUNCTIONS
2160 *---fall into EXIT routine-------
2170 * (DE78) DE5A DE63 DE6E DEB0        callers
2180 EXIT.TO.CALLER
2190        LDA #0       Clear BACKUP bit
2200        STA BACKUP.BIT
2210        LDY SYS.ERRNUM    If any error code,
2220        CPY #1            then set carry
2230        TYA               and clear Z-bit
2240        PHP               Save this status
2250        SEI               Disable IRQ's
2260        LSR MLI.ACTIVE.FLAG  Clear this flag
2270        PLA          Get saved status
2280        TAX          and keep it in X-reg
2290        LDA MLI.RETURN+1
2300        PHA          Put return address on stack
2310        LDA MLI.RETURN
2320        PHA
2330        TXA          Now push the status for RTI
2340        PHA
2350        TYA          Get error code in A-reg
2360        LDX MLI.X    Restore X and Y
2370        LDY MLI.Y
2380        PHA          Error code on stack
2390        LDA E000.BYTE
2400        JMP LC.BRIDGE.EXIT
2410 *--------------------------------
2420 *   LC.BRIDGE.EXIT is code at $BFA0 in
2430 *   the system global page.  It restores
2440 *   the language card to the state it
2450 *   was in when JSR $BF00 was exectuted.
```

```
2460 *--------------------------------
2470 *  LC.BRIDGE.EXIT EOR $E000
2480 *                BEQ .1   BFAA
2490 *                STA $C082
2500 *                BNE .2   BFB5
2510 *  .1            LDA D000.BYTE  $BFF5
2520 *                EOR $D000
2530 *                BEQ .2   BFB5
2540 *                LDA $C083
2550 *  .2            PLA
2560 *                RTI
2570 *--------------------------------
2580 ERR.NO.DEVICE
2590        LDA #$28      "NO DEVICE CONNECTED"
2600        JSR CALL.SYSERR
2610 ERR.CALL.NO
2620        LDA #1        "BAD CALL TYPE"
2630        BNE DEAD
2640 ERR.PARM.CNT
2650        LDA #4        "BAD PARAMETER COUNT"
2660 DEAD   JSR CALL.CALL.SYSERR
2670        BCS EXIT.TO.CALLER  ...ALWAYS
2680 *--------------------------------
2690 BLOCK.IO.SETUP
2700        LDY #5        COPY REST OF COMMAND BLOCK
2710        PHP           FROM IOB TO ZERO-PAGE
2720        SEI           DO NOT ALLOW IRQ'S
2730 .1     LDA (PARM.PNTR),Y
2740        STA COMMAND,Y
2750        DEY
2760        BNE .1
2770        LDX BUFF.PNTR+1
2780        STX GEN.PNTR2+1
2790        INX
2800        INX
2810        LDA BUFF.PNTR
2820        BEQ .2
2830        INX
2840 .2     JSR CHECK.IF.MEM.FREE
2850        BCS .3        ...NOT FREE
2860        JSR BLOCK.IO
2870        BCS .3        ...I/O ERROR
2880        PLP           RESTORE IRQ STATUS
2890        CLC           NO ERRORS
2900        RTS
2910 *--------------------------------
2920 .3     PLP           RESTORE IRQ STATUS
2930 CALL.CALL.SYSERR JSR CALL.SYSERR
2940 *--------------------------------
2950 * (DEDA) DECE EC0A EE83 F0E4 F475   callers
2960 BLOCK.IO
2970        LDA UNIT.NO  Clean this up a little
2980        AND #$F0
2990        STA UNIT.NO
3000        LSR           Make it into index too
3010        LSR
```

```
3020        LSR
3030        TAX
3040        LDA DRIVER.ADDR.TABLE,X
3050        STA JUMP
3060        LDA DRIVER.ADDR.TABLE+1,X
3070        STA JUMP+1
3080        JMP (JUMP)
3090 *--------------------------------
3100        .OR $FD65
3110        .TA $800
3120 COMMAND.HASH.TABLE
3130        .HS D3.00.00.00.40.41.00.00
3140        .HS 80.81.82.65.C0.C1.C2.C3
3150        .HS C4.C5.C6.C7.C8.C9.CA.CB
3160        .HS CC.CD.CE.CF.00.D0.D1.D2
3170 PARM.CNT.TABLE
3180        .HS 02.FF.FF.FF.02.01.FF.FF
3190        .HS 03.03.00.04.07.01.02.07
3200        .HS 0A.02.01.01.03.03.04.04
3210        .HS 01.01.02.02.FF.02.02.02
3220 *--------------------------------
3230        .LIF
```

```
################################################################################
#  Another Update to Bob's ProDOS Program Selector
################################################################################
```

Bob Sander-Cederlof

December 1986

The following refers back to the new ProDOS Quit Code I wrote and published in the July 86 issue of AAL. It has been very popular, judging from the number of letters and phone calls we have received.

Eric Trehus (T'n'T Software) pointed out that I ignored one or more of the conventions Apple established for Quit-Code Program Selectors. On page 87 of the ProDOS Technical Reference Manual, the paragraph with number 2 states that the name of the system program should be stored in a buffer at $280, starting with a length byte. The first paragraph on page 88 says any non-standard Quit Code must begin with a CLD instruction, so programs can tell who loaded them.

If you want the CLD instruction there, go ahead and insert one between lines 1310 and 1320. I have not found it necessary for any programs I use.

Eric says that when going from BASIC.SYSTEM to APLWORKS.SYSTEM he needed the program name stored in $280. I have never run into the problem, but it is easy to fix. Eric suggested inserting the following two lines:

```
       2065    STA $280
       2125    STA $280,Y
```

[Eric's change takes six bytes, so you need to be sure the code still fits in $300 bytes.]

If you do it Eric's way, only the name of the system file gets stored, without any prefix. I wondered whether or not a full pathname should be there, so I consulted Gary Little's "Apple ProDOS--Advanced Features" book. On page 141, near the bottom, he says either a full or a partial pathname should be put at $280. We can get the full pathname into $280 without Eric's two lines, by simply changing line 4860 from "PATHNAME .BS 64" to "PATHNAME .EQ $280". This is my preference.

When I was trying out the above, I stumbled across a problem. If my Selector finds no SYS or DIR files in a directory, it still displays the pathname and prompt messages. If you then type the RETURN key, it may try to execute garbage, or try various other things. The only valid keystroke when no files are listed is ESCAPE, which will take you back to the list of volume names. Adding two lines makes it go there without displaying the empty list:

```
       1771    TXA     see if any files listed
       1772    BEQ .2  ...none listed, start over
```

We noticed the other day that when we ran Erv Edge's correction to my program (Aug 86, page 1), we reversed the information. We said change line 3390 from BNE .1 to BPL .1; actually, it is the other way around: change from BPL to BNE. Most of you figured that out already, but we are sorry for the confusion.

```
################################################################################
#  Little Bugs in ProDOS /RAM
################################################################################
```

Bob Sander-Cederlof

January 1987

When a ProDOS device driver reads or writes a block, it is supposed to return an error code in the A-register. If there was an error, this number will be nonzero, and the Carry status bit will be set. If there was no error, the A-register is supposed to contain zero, and Carry will be clear.

The other day I was working with a program that was supposed to read and write blocks from the /RAM drive under ProDOS. My program printed the error code returned by the call, and I was getting a non-zero value in the A-register even when there was no error. I just assumed this was a trivial bug in ProDOS, and ignored it. However, I was not using the standard ProDOS /RAM driver, because I had installed the Applied Engineering driver which uses the entire RAMWORKS card for /RAM. Anyway, I just went on about my business.

A few days later the January 1987 issue of Nibble arrived at my doorstep. In a letter to the editor on page 123, from Steven Humpage, there was an explanation of the bug. The part of the device driver that resides in Main RAM is located at $FF00. At $FF47 is a routine named EXIT, which restores some zero-page locations, restores a vector in page 3, and returns. This code is called with either zero in A and Carry clear, or an error code in A and carry set. The routine begins with PHP, PHA and ends with PLP, PLA. WRONG! This swaps the P- and A-registers. Since the C-bit is bit 0 in the P-register, a zero (from the A- register) results in carry clear. However, the P-register con- tents come back in A, something like $36. Since the only error code the /RAM driver returns is $27, and this has bit 0 set, swapping registers leaves carry set. It also sets the V-bit, which could be unexpected. The error code in A now becomes what the status was, so we get an error code of $35, I believe.

The same bug exists in AE Prodrive. The bug can easily be fixed, by the following sequence. I am showing it as I do it from within the S-C Macro Assembler. If you do it from within BASIC.SYSTEM, you will have to CALL-151 to get into the monitor to install the patch.

```
       :UNLOCK PRODOS                  :UNLOCK PRODRIVE
       :BLOAD PRODOS,TSYS,A$2000       :BLOAD PRODRIVE
       :$2C5F:68 28                    :$245F:68 28
       :BSAVE PRODOS,TSYS,A$2000       :BSAVE PRODRIVE
       :LOCK PRODOS                    :LOCK PRODRIVE
```

I am assuming ProDOS 1.1.1. The bug has already been fixed in version 1.2.

While we are at it, Humpage also pointed out another bug which affects both ProDOS and Prodrive. The code which ProDOS puts into pages 2 and 3 in the AuxRAM has an error. The CMP #$0D at $34A should be CMP #$0E. As it is it allows writing to block 7, which is then mapped back over the top of page zero and one, clobbering things disastrously. This change write protects that block. You only need to change the PRODOS file, because Prodrive uses it also. Change $2B4B from $0D to $0E.

###
ProDOS-based Intelligent Disassembler
###

Bob Sander-Cederlof

February 1987

In response to the requests of many of you, I have at long-last developed a disassembler which runs under ProDOS. (This new product is distinctly different from the Rak-Ware DISASM, which runs under DOS 3.3. The Rak-Ware product is still the best one to use if you are using DOS.) Here are some of the features of the new S-C ProDOS DISASM:

* Input is from one or more binary object files, including file types BIN and SYS.
* Output is to one or more "S-C" type (compressed source code) files.
* Generates comment lines before each label listing all references to that label.
* Disassembly is "script" driven, allowing incremental enhancement.
* Input files may be positioned to specific starting addresses.
* Decodes ProDOS "MLI" calls as such.
* Allows pre-named symbols up to 32-characters long.

Of all the features, the most important may be the "script". This is essentially a "program", written in "disassembly language". The script allows you to define which input files to include and which output files to generate, to name symbols such as monitor entry points and major subroutines in your program being disassembled, to define table areas, and even to insert comments.

The script itself is written using the standard S-C Macro Assembler, and may be saved on a source file just as an assembly-language program would. As you gain knowledge about the program you are disassembling, you can add lines to the script.

Version 1.0 handles all of the 65C02 instruction set. Future enhancements which which are definitely planned include expanding to include the entire 65816 instruction set. Version 1.0 is for sale now for $50 including the commented source code.

```
################################################################################
#  Some Bugs in Apple's ProDOS Version 1.3
################################################################################
```

Bob Sander-Cederlof

March 1987

Apple recently released version 1.3 of ProDOS, and sent copies to all licensees. We started sending it out with the ProDOS version of the S-C Macro Assembler. But last week (around March 16th) we received a letter from Apple "recalling" version 1.3. It has two serious problems.

First, there is a "BRA" (BRanch Always) opcode in it. This means version 1.3 will not operate in an older Apple with only a 6502 microprocessor. If you have a 65C02 or 65802 or 65816, no problem here. You are safe in a IIgs, //c, or enhanced //e. You are also safe if you have upgraded the cpu chip yourself in an older machine. That offending instruction could just as well be changed to a BEQ opcode, because that would always branch in this case. With that change the 6502 machines work fine.

Second, when the Apple experts tried to implement the patch developed in Australia and reported first by Tom Weishaar in "Open Apple" to fix an elusive disk-trashing problem, they didn't do it right. This is the same fix I reported in the November AAL, page 13. (Turns out I didn't report it right either, because I overlooked one part of the patch. More on this below.) The way Apple did it causes severe problems with two-drive systems. When you are accessing drive two, version 1.3 keeps switching back to drive 1. If you try to use FILER to copy a volume from drive 1 to a blank disk in drive 2, and if you remember to write-protect the disk in drive 1, FILER will hang up with an I/O ERROR after initializing the drive 2 disk. FILER evidently applies the drive 1 write-protect status to drive 2, and gives up. I don't even want to experiment without having drive 1 write-protected! On the other hand, if you copy from drive 2 to drive 1 it works, but it takes a lot longer than it should to read each segment from the source disk in drive 2.

Both bugs can be fixed by rather simple patches. Boot up PRODOS, into either BAISC.SYSTEM or the S-C Macro Assembler. Then load the PRODOS file, with "BLOAD PRODOS,TSYS,A$2000". Then get into the monitor with "MNTR" from S-C Macro Assembler or "CALL-151" from BASIC.SYSTEM. Type the following patches:

```
       4CCD:F0         (was 80, changing BRA to BEQ)
       5204:BD 8E C0   (was EA EA EA)
       58C3:BD 80 C0 BD 82 C0 BD 84 C0 BD 86 C0
```

Then get back into the system by typing "3D0G", and save the new version with "BSAVE PRODOS,TSYS,A$2000".

The third patching line above replaces Apple's flawed loop which walked on too many soft-switches. Apple's loop does a LDA from C080, C082, C084, and C086; this is correct. It also does it from C088, C08A, C08C, and C08E. This is not correct. It turns off the motor and selects drive 1. The only correct one among this group of four was C08E, intended to be sure the selected drive is in read mode.

```
 Old Code        Apple's Loop      My Patch
-----------    ---------------    -----------
STA $C080,X       LDY #8          LDA $C080,X
STA $C082,X    .1 LDA $C080,X     LDA $C082,X
STA $C084,X       INX             LDA $C084,X
STA $C086,X       INX             LDA $C086,X
                  DEY
                  BNE .1
                  NOP
                  NOP
```

My patch puts the C08E load where Tom's Australian-connection originally put it, over some NOPs which immediately followed the JSR to the code shown above.

Now about my incomplete patch to version 1.1.1 from last November. I omitted the "LDA $C08E,X", which gets patched at location $5004 in this version. I also mis-typed the address for the other patches as going at $56D3 when they actually belong at $56C3. So, in version 1.1.1, following the same load-patch-save sequence above, the patches are:

```
       5004:BD 8E C0   (was EA EA EA)
       56C3:BD 80 C0 BD 82 C0 BD 84 C0 BD
               (changing 9D's to BD's)
```

Version 1.4 is due out soon, and we trust Apple will have it all right this time. Still, I am getting skittish.

Meanwhile, unless you are using a IIgs, you may wish to stick with version 1.1.1. The only real advantage the newer versions have is automatic recognition of the IIgs clock-calendar chip. You don't need this feature if you are not running in a IIgs.

```
################################################################################
#  Commented Listing of ProDOS -- $DEF3-DFE4
################################################################################
```

Bob Sander-Cederlof

March 1987

As I promised last December, here is another piece of ProDOS. This time I am unveiling the code which processes IRQ interrupts, and the handlers for the related MLI calls. All of the following applies to version 1.1.1 of ProDOS. Later versions may differ in this area.

The two MLI calls related to interrupts are $40 (Allocate Interrupt) and $41 (De-Allocate Interrupt). There is room inside ProDOS for connecting up to four user-coded routines for processing IRQ interrupts. The Allocate Interrupt call stores the address of your routine at the next available entry in the IRQ Path Table. This table exists in the MLI Global Page ($BF00-BFFF), and is shown in lines 1140-1170 in the listing below. When you boot ProDOS these four entries all contain $0000, indicating no interrupts are allocated. An MLI call of the form:

```
       JSR $BF00
       .DA #$40,IRQ.IOB
```

with an IOB like this:

```
       IRQ.IOB .DA #2
       IRQ.NUM .BS 1
               .DA MY.IRQ.PROCESSOR
```

will cause the address of MY.IRQ.PROCESSOR to be stored in the IRQ Path table. The index into the table pointing to the entry used will be converted to an integer from 1 to 4, and stored at IRQ.NUM in the IOB. The purpose of this number is to allow you to later de-allocate the interrupt if you wish. A call and an IOB like this will de-allocate an interrupt:

```
          JSR $BF00
          .DA $41,IRQ.IOB
       ---
       ---
       IRQ.IOB .DA #1
       IRQ.NUM .BS 1   (filled in by program)
```

Note that the first byte if the IOB is different this time, because there is only one parameter rather than two. It is important to de-allocate, because otherwise a sneaky interrupt could occur which would cause ProDOS to branch after your program is gone.

Another way to de-allocate is to store zeroes directly into the IRQ Path table, but Apple warns against this practice. It is quicker and easier, though.

There may be more than one source of IRQ interrupts in a given system. For example, you may be using both a clock card and a modem, both with interrupts. ProDOS allows you to have separate interrupt handlers for each of them installed. When an IRQ occurs the first handler installed will be called first. If that handler determines the IRQ is its own, it should process the IRQ and return with carry status clear. If not, the handler should return with carry status set. ProDOS will try giving the interrupt to each handler in turn, until one of them returns with carry clear. If none of them claim the IRQ, or if there are no processors allocated, "System Death" will occur: you will get error code $01, and a message to insert system disk and restart. It seems a nicer approach to unclaimed interrupts would be to count it, and continue processing. When the count exceeds some magic number, say 255, that would be the time to go through the agony of "System Death". I would also like to know the cause of "Death", if possible.

Lines 1400-1640 in the listing below show the code for allocating an interrupt. The code searches the IRQ Path table for an available entry (equal to 0000), and inserts the user's processor address in the first one found. If none are found you get error $25, INTERRUPT TABLE FULL. Actually only the high byte of each entry is checked, which means you cannot put an interrupt processing routine anywhere in page zero. The MLI call will allow you to do so, and it will even work, but if you later try to allocate another interrupt it will use the same table entry and clobber the first one. I suppose this is a bug in ProDOS, but not too likely to cause any problem because you are not likely to stick your code down in that page. Still, it COULD happen.

Lines 1660-1770 de-allocate an interrupt routine. If the interrupt index number is not in the range form 1 to 4, you will get error $53, BAD PARAMETER. Otherwise, the indicated entry will be zeroed.

When an IRQ interrupt occurs, if the status is such that interrupts are enabled, the processor status and the current PC-address will be pushed onto the stack; then processing will branch to the address currently at $FFFE and FFFF. What address is there will depend on which kind of Apple you are in, and whether ROMs or RAM are currently switched into the $D000-FFFF area. The original Apple II monitor and also the Apple II+ monitor vector IRQs into the $F8 monitor ROM area. A short routine there saves some registers and separates BRKs from IRQs (because they both share the same vector at $FFFE). IRQs then branch through another vector at $3FE and 3FF. The various Apple //e monitors vector IRQs and BRKs to an address at $C3FA, while the //c monitors send them to $C803.

When you boot ProDOS the installer/relocator code checks which kind of monitor you have. If your IRQ vector points anywhere below $D000, it assumes you have a "new style" monitor; if it points to anywhere between $D000 and $FFFF it assumes you have an "old style" monitor. The Apple II and II+ are old style, all others are new style. The installer/relocator stores a flag at $DFD8 so that the IRQ handler can tell what kind of machine it is in when an IRQ occurs later. This flag is shown at lines 2650-2710. In new style machines the vector found in ROM at $FFFE is copied into both Main and Auxiliary RAM banks at the same address, in case an interrupt occurs when RAM is switched on. In old style machines the address $FF9B is left in the RAM vector, pointing to a special IRQ handler shown in lines 3060-3400 below.

The vector at $3FE,3FF is set up to point to IRQ.ENTRY, a short routine inside the MLI Global Page. This is shown in lines 3000-3040. Since no matter what kind of monitor is resident the IRQ eventually vectors here, I will start the explanation here. Lines 3020-3030 turn on RAM at $D000-FFFF, so that ProDOS is accessible. It also write enables the RAM, because the IRQ processing will be storing a value at $DFCE, which identifies the current owner of the $C800 space (lines 1970-1980).

Lines 1800-1870 save the registers in the MLI Global Page. If you are in an old style machine lines 1880-1950 will save the processor status and return address in the Global Page as well.

ProDOS wants to make it easier to write IRQ processors, so it also makes sure you can use the stack and some page zero. If there are not at least 128 bytes left on the stack it will pop off 16 bytes and save them in a special buffer; if there are at least 128 bytes left this step is skipped. Then lines 2070-2120 save zero page locations $FA through $FF in a special buffer. Your IRQ processor can use these six bytes without worrying about saving and restoring them. (If you need more, you will have to save-restore them yourself.) This is all nice, but it does add to the general overhead for processing interrupts, which is already burdensome.

Lines 2130-2320 sequence through the installed interrupt processors until one of them claims the IRQ. Lines 2330-2350 signal DEATH if none of the processors claim the IRQ.

If the IRQ is properly claimed, lines 2360-2410 restore the six zero page bytes; lines 2420-2500 restore the 16 bytes of stack space if they were previously saved. Lines 2520-2630 restore some registers and the $C800 space if you are in an old style machine, and in any case branch to the IRQ.EXIT routine in the MLI Global Page.

IRQ.EXIT, shown in lines 2800-2960, restores the correct kind of memory (RAM or ROM) and then executes an RTI instruction. In an old style machine if the IRQ happened during a time when the RAM was switched on, this will send control to IRQ.EXIT.OLD, shown in lines 3690-3740. In a new style machine, or in any machine if the ROMs were on when the IRQ happened, the RTI will go back to the control of the monitor; exactly where that is depends on which monitor. Normally BANK.ID.BYTE contains the value $01. If an IRQ occurs in an old style machine when RAM is switched on, it will be changed to $00 or $FF depending on which $D000 bank is selected. Lines 2930-2940 change it back to $01 after one either $00 or $FF is processed.

One advantage to having both IRQ.ENTRY and IRQ.EXIT in the MLI Global Page is that you could substitute your own code if you wish. If you want to reduce overhead, and you know that you will always be running in a specific monitor configuration, you can patch in here. You could also patch in through the vector at $3FE, and avoid even more overhead. However, you would no longer be "standard".

I published a listing of lines 3070-3640 way back in December 1983, but I decided to include it here for completeness. This code is only used in an old style machine, and only when the IRQ occurs while RAM is switched on. The vector at $FFFE starts up the code at line 3140. The nonsense in lines 3140-3180 regarding location $45 makes sure we do not clobber the saved A-register. The old style monitor ROMs save the A-register at $45 rather than on the stack. This conflicts with use of the same location within both DOS 3.3 and ProDOS. QUESTION: Wouldn't it have been both easier and better to avoid using locaton $45 inside ProDOS? Kludge on top of kludge, if you ask me.

Lines 3290-3340 set up fake data on the stack for later use by an RTI instruction. Lines 3350-3380 do the same for an RTS instruction. Note that RTS requires an address with is one less than the actual address, while RTI requires the address un-modified. RTS pops the address, adds one, and branches; RTI pops the address and status, and branches without adding one. Line 3400 switches back to ROM. This means the next instruction will be executed from $FFCB in ROM, which is ALWAYS an RTS. Anyway it had BETTER be! If you ever make your own monitor ROM, be sure to leave an RTS here. (You will also need an RTS at $FF58, because a lot of I/O firmware expects that one is there.)

Lines 3420-3470 are executed in the old style machines if RAM is switched on when you hit RESET. That is, if you have the particular type of RAM card which leaves the F8 area switched on when you hit RESET. Many of them switch back to ROM when RESET occurs. Just in case, the code is here.

That about wraps it up. But it still leaves a lot of mystery in that part of IRQ processing which occurs inside the monitor ROMs. Each monitor version has its own unique code. The Apple II was simple, the II+ about the same. The three versions of //e and three versions of //c monitors of which I am aware are all mutually different. The IIgs is even more so. Perhaps in a future article we can rationalize them all.

```
1000 *SAVE MLI.IRQ
1010 *--------------------------------
1020 PARM.PNTR        .EQ $40,41
1030 COMMAND          .EQ $42
1040 SAVE.A           .EQ $45
1050 *--------------------------------
1060 CURRENT.ROM.SLOT .EQ $07F8   $C0 + Slot which owns $C800.
1070 *--------------------------------
1080 CALL.SYSERR      .EQ $BF09
1090 CALL.DEATH       .EQ $BF0C
1100 *--------------------------------
1110 SAVE.LOC45       .EQ $BF56   Used if in Apple II or II+
1120 SAVE.D000        .EQ $BF57       ditto
1130 *--------------------------------
1140 IRQ.PATH.1       .EQ $BF80   These are 0000 if not allocated,
1150 IRQ.PATH.2       .EQ $BF82   address of user IRQ handler
1160 IRQ.PATH.3       .EQ $BF84   if allocated.
1170 IRQ.PATH.4       .EQ $BF86
1180 *--------------------------------
1190 IRQ.A            .EQ $BF88
1200 IRQ.X            .EQ $BF89
1210 IRQ.Y            .EQ $BF8A
1220 IRQ.S            .EQ $BF8B
1230 IRQ.P            .EQ $BF8C
1240 BANK.ID.BYTE     .EQ $BF8D
1250 IRQ.RETURN       .EQ $BF8E,BF8F
1260 *--------------------------------
1270 IO.RESET.ROMS    .EQ $CFFF   De-select $C800 space.
1280 *--------------------------------
1290 IRQ.SV .EQ $FEDF thru $FEEE  (16 bytes saved from STACK)
1300 *--------------------------------
1310        .PH $DEF3
1320 *--------------------------------
1330 *   Handle $40 and $41 MLI calls
1340 *--------------------------------
1350 * (DEF3) DE57
1360 INTERRUPT.HANDLER
1370        STA COMMAND  Save in case anyone cares later.
1380        LSR          $40 or $41, lsb into CARRY
1390        BCS .5       ...$41 is DEALLOCATE
1400 *---$40 is ALLOCATE--------------
1410        LDX #3       FOR X = 3 TO 9 STEP 2
1420 .1     LDA IRQ.PATH.1-2,X
1430        BNE .2       ...ALREADY ALLOCATED
1440        LDY #3       FOUND HOLE, INSTALL IRQ
1450        LDA (PARM.PNTR),Y
```

```
1460        BEQ .3        BAD PARAMETER
1470        STA IRQ.PATH.1-2,X
1480        DEY
1490        LDA (PARM.PNTR),Y
1500        STA IRQ.PATH.1-3,X
1510        TXA           GIVE IRQ# TO CALLER
1520        LSR           MAKE 3,5,7,9 INTO 1,2,3,4
1530        DEY
1540        STA (PARM.PNTR),Y
1550        CLC           Signal NO ERROR
1560        RTS
1570 .2     INX           Next X
1580        INX
1590        CPX #11
1600        BNE .1
1610        LDA #$25      "INTERRUPT TABLE FULL"
1620        BNE .4        ...ALWAYS
1630 .3     LDA #$53      "BAD PARAMETER"
1640 .4     JSR CALL.SYSERR   (NEVER RETURNS)
1650 *---$41 is DEALLOCATE------------
1660 .5     LDY #1
1670        LDA (PARM.PNTR),Y
1680        BEQ .3        ...0 is illegal value
1690        CMP #5        Must be 1,2,3,4
1700        BCS .3        ...too large
1710        ASL           DOUBLE FOR INDEX
1720        TAX
1730        LDA #0        CLEAR THE ENTRY
1740        STA IRQ.PATH.1-2,X
1750        STA IRQ.PATH.1-1,X
1760        CLC           Signal NO ERROR
1770        RTS
1780 *-------------------------------
1790 *   If an IRQ occurs, we eventually get HERE.
1800 *-------------------------------
1810 IRQ.HANDLER
1820        LDA SAVE.A
1830        STA IRQ.A
1840        STX IRQ.X
1850        STY IRQ.Y
1860        TSX
1870        STX IRQ.S
1880        LDA ENHANCE.FLAG
1890        BNE .1        ...In a "new style" monitor
1900        PLA           ...In an Apple II or II+ monitor
1910        STA IRQ.P     Save P-reg and RETURN address
1920        PLA
1930        STA IRQ.RETURN
1940        PLA
1950        STA IRQ.RETURN+1
1960 .1     TXS           Keep P-reg and RETURN on stack
1970        LDA CURRENT.ROM.SLOT   Save $C800 Slot
1980        STA ROM.PAGE.BYTE
1990 *---Save some stack, maybe-------
2000        TSX           If in bottom half of stack,
2010        BMI .3             then save 16 bytes of it.
```

```
2020        LDY #15      SAVE 16 BYTES FROM STACK
2030 .2     PLA
2040        STA IRQ.SV,Y
2050        DEY
2060        BPL .2
2070 *---Save some page zero----------
2080 .3     LDX #$FA     SAVE 6 BYTES FROM PAGE ZERO
2090 .4     LDA 0,X      $FA...FF
2100        STA IRQ.SV-$FA+16,X
2110        INX
2120        BNE .4
2130 *---Call to first IRQ vector-----
2140        LDA IRQ.PATH.1+1
2150        BEQ .5       IRQ#1 EMPTY
2160        JSR IRQ.1    Try this IRQ level
2170        BCC .9       ...IRQ Claimed, Now Exit
2180 *
2190 .5     LDA IRQ.PATH.2+1
2200        BEQ .6       IRQ#2 EMPTY
2210        JSR IRQ.2    Try this IRQ level
2220        BCC .9       ...IRQ Claimed, Now Exit
2230 *
2240 .6     LDA IRQ.PATH.3+1
2250        BEQ .7       IRQ#3 EMPTY
2260        JSR IRQ.3    Try this IRQ level
2270        BCC .9       ...IRQ Claimed, Now Exit
2280 *
2290 .7     LDA IRQ.PATH.4+1
2300        BEQ .8       IRQ#4 EMPTY
2310        JSR IRQ.4    Try this IRQ level
2320        BCC .9       ...IRQ Claimed, Now Exit
2330 *---No IRQ vectors alive!--------
2340 .8     LDA #$01     Un-claimed Interrupt Error
2350        JSR CALL.DEATH    (NEVER RETURNS)
2360 *---IRQ PROCESSING COMPLETE------
2370 .9     LDX #$FA     RESTORE $FA...FF
2380 .10    LDA IRQ.SV-$FA+16,X
2390        STA 0,X
2400        INX
2410        BNE .10
2420 *---If saved, restore stack------
2430        LDX IRQ.S
2440        BMI .12      16 BYTES FROM STACK NOT SAVED
2450        LDY #0       RESTORE 16 BYTES TO STACK
2460 .11    LDA IRQ.SV,Y
2470        PHA
2480        INY
2490        CPY #16
2500        BNE .11
2510 *---Choose EXIT routine----------
2520 .12    LDA ENHANCE.FLAG
2530        BNE IRQXIT   ..."New style" monitor ROMs
2540        LDY IRQ.Y    ...Apple II or II+ monitor.
2550        LDX IRQ.X
2560        LDA IO.RESET.ROMS    Turn off $C800 bank
2570        LDA $CF00            Select Interrupted $C800 bank
```

```
2580 * (DFCE) DF5E DFCF          (Hi-byte filled in)
2590 ROM.PAGE.BYTE .EQ *-1       (Self-modifying code!)
2600        LDA ROM.PAGE.BYTE
2610        STA CURRENT.ROM.SLOT
2620 * (DFD5) DFC1
2630 IRQXIT JMP IRQ.EXIT
2640 *--------------------------------
2650 * (DFD8) DF49 DFBE
2660 ENHANCE.FLAG .HS 00 (Set to 01 by relocator if new
2670 *                     type ROM is found)
2680 *   ((If IRQ vector in ROM at $FFFE,F points below
2690 *      $D000, it is "new type".  The "old type",
2700 *      which is the original $F8 ROM in the Apple II
2710 *      or that of the Apple II+, points to $Fxxx.))
2720 *--------------------------------
2730 * (DFD9-E2) DF7C DF86 DF90 DF9A
2740 IRQ.1 JMP (IRQ.PATH.1)
2750 IRQ.2 JMP (IRQ.PATH.2)
2760 IRQ.3 JMP (IRQ.PATH.3)
2770 IRQ.4 JMP (IRQ.PATH.4)
2780 *--------------------------------
2790        .PH $BFD0
2800 *--------------------------------
2810 *   IRQ ENTRY/EXIT CODE IN GLOBAL PAGE
2820 *--------------------------------
2830 IRQ.EXIT
2840        LDA BANK.ID.BYTE
2850 IRQ.EXIT.1
2860        BEQ .2
2870        BMI .1
2880        LSR
2890        BCC .3
2900        LDA $C081    Switch on ROMs at D000-FFFF
2910        BCS .3       ...ALWAYS
2920 .1     LDA $C083    Switch on RAMs at D000-FFFF
2930 .2     LDA #1
2940        STA BANK.ID.BYTE
2950 .3     LDA IRQ.A
2960        RTI
2970 *--------------------------------
2980 *   An IRQ interrupt comes here when it occurs
2990 *      because of the vector at $3FE,3FF.
3000 *--------------------------------
3010 IRQ.ENTRY
3020        BIT $C08B    Switch on and write-enable
3030        BIT $C08B         RAM at D000-FFFF
3040        JMP IRQ.HANDLER
3050 *--------------------------------
3060        .PH $FF9B
3070 *--------------------------------
3080 *   IRQ CODE FOR APPLE II AND II+ MONTOR ROMS
3090 *      This code is used when IRQ happens while
3100 *      the RAM at D000-FFFF is switched on (inside
3110 *      an MLI call, for example) if we have the
3120 *      "new style" monitor ROMs.
3130 *--------------------------------
```

```
3140 IRQ    PHA          SAVE A-REG
3150        LDA SAVE.A      ALSO SAVE SAVE.A
3160        STA SAVE.LOC45
3170        PLA          NOW PUT A-REG INTO SAVE.A
3180        STA SAVE.A
3190        PLA          PEEK AT STATUS
3200        PHA
3210        AND #$10     WAS IT "BRK"?
3220        BNE .2       ...YES, LET MONITOR HANDLE IT
3230        LDA $D000    CHECK WHETHER D000 BANK 1 OR 2
3240        EOR #$D8     "CLD" OPCODE
3250        BEQ .1       ...IN D000 BANK 1
3260        LDA #$FF     ...IN D000 BANK 2
3270 .1     STA BANK.ID.BYTE
3280        STA SAVE.D000
3290        LDA /IRQ.EXIT.OLD   PUSH FAKE "RTI" VECTOR
3300        PHA
3310        LDA #IRQ.EXIT.OLD
3320        PHA
3330        LDA #$04
3340        PHA
3350 .2     LDA /$FA41   PUSH FAKE "RTS" VECTOR INTO
3360        PHA               MONITOR ROM
3370        LDA #$FA41
3380        PHA
3390 CALL.MONITOR
3400        STA $C082    SWITCH TO MOTHERBOARD ROMS,
3410 *                   WHERE THERE IS AN "RTS" OPCODE
3420 *--------------------------------
3430 RESET  LDA RESET.VECTOR+1
3440        PHA          PUSH FAKE "RTS" INTO MONITOR
3450        LDA RESET.VECTOR
3460        PHA
3470        JMP CALL.MONITOR
3480 *--------------------------------
3490 RESET.VECTOR .DA $FA61   MON.RESET-1
3500 *--------------------------------
3510 IRQ.SPLICE
3520        STA IRQ.A
3530        LDA SAVE.LOC45
3540        STA SAVE.A
3550        LDA $C08B    FINISH WRITE-ENABLING RAM
3560        LDA SAVE.D000
3570        JMP IRQ.EXIT.1
3580 *--------------------------------
3590        .BS $FFFA-*  <<<EMPTY SPACE>>>
3600 *--------------------------------
3610 V.NMI      .DA $03FB
3620 V.RESET    .DA RESET
3630 V.IRQ      .DA IRQ  (Replaced by relocator with
3640 *                   the value from ROM vector if
3650 *                   the machine has "new style" monitor.
3660 *--------------------------------
3670        .PH $BF50
3680 *--------------------------------
3690 *   LITTLE PIECE OF IRQ EXIT CODE USED WITH
```

```
3700 *      OLD TYPE MONITOR ROMS
3710 *--------------------------------
3720 IRQ.EXIT.OLD
3730        LDA $C08B    SWITCH RAM ON, D000 BANK 1
3740        JMP IRQ.SPLICE
3750 *--------------------------------
3760        .LIST OFF
```

```
################################################################################
#  New Supplement to "Beneath Apple ProDOS" Available
################################################################################
```

Bob Sander-Cederlof

April 1987

Quality Software has published a new supplement to for "Beneath Apple ProDOS," which includes information on ProDOS versions 1.2 and 1.3. It is 30 pages longer than the previous edition, which covered version 1.1.1 of ProDOS.

You might be wondering, "What is a supplement, anyway?" The book "Beneath Apple ProDOS" ("BAP") contains much reference material needed to really take advantage of ProDOS capabilities. While other books now cover much of the same ground, "BAP" was the first one to put it all into print. The supplement, however, is unique: it contains a complete description of the internal details of both the ProDOS MLI kernel and BASIC.SYSTEM. If you are at all involved with the inner works of ProDOS, or are having trouble finding out the REAL scoop on some issues, you NEED the supplement. I have used my copies extensively, and depended heavily on it when writing the ProDOS version of S-C Macro Assembler.

The original supplement, for versions 1.0.1 and 1.0.2 cost $10. The second and third editions are $12.50 each. Incredibly low-priced! You must order these directly from Quality Software, at 21610 Lassen Street #7, Chatsworth, CA 91311. As I understand it, the supplement is only sold to owners of "BAP", and you have to use the coupon found on page 8-9 of "BAP" to do the ordering. ("BAP" itself is $19.95 retail, but we sell it for $18 here.)

```
################################################################################
#  Missing ProDOS Books
################################################################################
```

May 1987

We had a scare a couple of weeks ago: Quality Software and Simon & Schuster had both run out of copies of Beneath Apple ProDOS, that excellent reference on the inner workings of ProDOS, so it looked for a while like we might lose a valuable resource. All is well, though, the folks at Quality are planning a new printing, so we expect to have more copies of the book in a month or two. We'll just hold any orders until that time.

Curiously, both Addison-Wesley's ProDOS Technical Reference Manual and Simon & Schuster's Apple ProDOS: Advanced Features for Programmers have been out of stock at the publishers for a couple of months now. A-W tells us that a revised edition of Apple's manual will be published in late June. S & S has Advanced Features on backorder, but won't quote even a tentative delivery date.

```
##############################################################################
#  More About Patching Apple's ProDOS Releases
##############################################################################
```

Bob Sander-Cederlof

May 1987

You remember in our March issue we talked about patches to fix Version 1.3 of ProDOS. Apple has pulled this version off the market, but there are still a lot of copies floating around. The patches we gave in the March article should make ProDOS 1.3 as good as any other version, but who knows?

Anyway, we heard through the grapevine that some unofficial copies of version 1.4 are out, and that a brand new bug has surfaced in this one. It seems someone put a "LDA $C09C,X" where "LDA $C08E,X" should be.

I ran across a listing in the Washington Apple Pi newsletter (May 1987 issue, page 16) of an Applesoft program which can install all known necessary patches in versions 1.1.1, 1.2, 1.3, and 1.4 of ProDOS. The program was originally written by Stephen Thomas to fix version 1.1.1, when the problem of the four STA's to the stepper motor soft-switches was discovered. (See Nov 86 AAL) Later Glen Bredon modified it to make the corresponding patches to later versions, as well as to fix the additional new bugs. I have further modified it, in an attempt to make it easier to understand.

```
100  TEXT : HOME :E = 0: PRINT "PRODOS PATCH PROGRAM"
110  IF  PEEK (116) < 128 THEN E = 1: GOTO 900: REM ENUF MEM?
120  ONERR  GOTO 900
130  REM ---READ PRODOS FILE---
140  PRINT  CHR$ (4)"UNLOCK PRODOS"
150  PRINT  CHR$ (4)"BLOAD PRODOS,TSYS,A$2000"

200  REM ---SEARCH $4000-$60FF FOR PATTERN---
210 V = 1: FOR B = 64 TO 96:A = B * 256
220  IF  PEEK (A + 4) <  > 189 THEN 250
230  IF  PEEK (A + 5) <  > 156 THEN 290
240  IF  PEEK (A + 6) = 192 THEN V = 3:B = 96: GOTO 290: REM VERSION 1.4
250  IF  PEEK (A + 4) <  > 234 THEN 290
260  IF  PEEK (A + 5) <  > 234 OR  PEEK (A + 6) <  > 234 THEN 290
270  IF  PEEK (A + 7) <  > 234 OR  PEEK (A + 8) <  > 234 THEN 290
280 V = 2:B = 96: REM VERSION BEFORE 1.4
290  NEXT B:E = 2: ON V GOTO 900,300,700

300  REM ---FOUND VERSION BEFORE 1.4---
310  POKE A + 4,189: POKE A + 5,142: POKE A + 6,192: REM "LDA $C08E,X"

400  REM ---LOOK FOR OTHER PATCH AREA---
410 A =  PEEK (A + 2) + 256 *  PEEK (A + 3) - 13 * 4096 + A + 5
420 E = 3: IF A < 4 * 4096 OR A > 6 * 4096 THEN 900
430  IF  PEEK (A) <  > 157 OR  PEEK (A + 3) <  > 157 THEN 500
440  IF  PEEK (A + 6) <  > 157 OR  PEEK (A + 9) <  > 157 THEN 500

450  REM ---FOUND VERSION 1.1.1 OR 1.2, SO CHANGE "STA" TO "LDA"---
460  FOR I = 0 TO 9 STEP 3: POKE A + I,189: NEXT I
470 V$ = "1.1.1": GOTO 800
```

```
500  REM  ---VERSION 1.3---
510  FOR I = 0 TO 12: READ B: IF  PEEK (A + I) <  > B THEN  GOTO 900
520  NEXT I: DATA 160,8,189,128,192,232,232,136,208,248,234,234,96
530  FOR I = 0 TO 11: READ B: POKE A + I,B: NEXT I
540  DATA 189,128,192, 189,130,192, 189,132,192, 189,134,192
550 A = 4 * 4096 + 12 * 256 + 12 * 16 + 13: REM  ADDRESS = $4CCD
560  FOR I = 0 TO 3: READ B: IF  PEEK (A + I) <  > B THEN 900
570  NEXT I: POKE A,240: REM CHANGE "BRA" TO "BEQ"
580 V$ = "1.3": GOTO 800
590  DATA 128,6,190,0

700  REM ---VERSION 1.4---
710  POKE A + 5,142: REM "LDA $C09C,X" TO "LDA $C08E,X"
720 V$ = "1.4"

800  REM ---WRITE PATCHED VERSION ON DISK---
810  PRINT  CHR$ (4)"BSAVE PRODOS,TSYS,A$2000"
820  PRINT  CHR$ (4)"LOCK PRODOS"
830  PRINT "PATCHES COMPLETED TO VERSION "V$: END

900  REM ---ERROR HANDLER---
910  PRINT  CHR$ (7)"ERROR!  NO PATCHES WERE MADE."
915  ON E GOTO 930,940,950
920  PRINT "PRODOS FILE NOT FOUND.": END
930  PRINT "NOT ENOUUGH ROOM TO LOAD PRODOS.": END
940  PRINT "PATCH LOCATION NOT FOUND.": END
950  PRINT "PRODOS FILE MAY HAVE BEEN PATCHED,"
960  PRINT "ALREADY, OR IS NOT A COMPATIBLE VERSION."
970  END
```

Lines 100-150 read the ProDOS system file into memory. Then Lines 200-290 search every page from $4000 through $60FF for either five NOPs starting at $xx04 or a "LDA $C09C,X" instruction at $xx04. If neither is found, nothing is patched. If the five NOPs are found, we have version 1.1.1, 1.2, or 1.3. If the LDA is found, we have version 1.4. If it is version 1.4, the only patch needed is to change it to "LDA $C08E,X", which is done at lines 700-720.

Older versions all need "LDA $C08E,X" poked where the five NOPs were, so line 310 takes care of this. Then we look at the address in the operand field of the instruction just prior to the five NOPs. This is a JSR to a little subroutine which we need to modify. Line 410 computes the location within the system file image for the twelve bytes we need to change.

There are several possible versions of this subroutine. If it is a series of "STA $C08x,X" instructions, we have version 1.1.1 or 1.2 and the STA opcodes should be changed to LDA opcodes. Lines 430 and 440 test for STA opcodes, and lines 450-470 make the changes. On the other hand, if the subroutine is like Apple put in version 1.3 we will replace it with a series of four LDAs just like we made in the older versions. Lines 500-590 handle this, and also change an errant "BRA" opcode to a "BEQ" opcode.

Finally, lines 800-830 write out the modified code and re-LOCK the file. I would be careful to check the changes made before doing this to every copy I own, if I were you. And bear in mind that Apple as a company has never authorized any of these changes. (They have only made them necessary, by their own incorrect changes.)

While this article was waiting for the press, Apple finally sent out correct copies of version 1.4. I received my master copy June 1st, and checked it against our patched version 1.3. They were identical except for the copyright dates and version numbers. The official date on this GOOD version 1.4 is April 17, 1987.

```
################################################################################
#  Correction to ProDOS Patcher
################################################################################
```

Bob Sander-Cederlof

June 1987

Mike McConnell called with a correction to my Applesoft ProDOS Patcher that affected its ability to find and fix versions 1.1.1 and 1.2. Lines 430 and 440 PEEKed at A, A+1, A+2, and A+3; in fact they should be PEEKing at A, A+3, A+6, and A+9. Change those two lines to:

```
     430 IF PEEK(A)<>157 OR PEEK(A+3)<>157 THEN 500
     440 IF PEEK(A+6)<>157 OR PEEK(A+9<>157 THEN 500
```

Then I noticed an error in the REMark at line 450. What it should say is:

```
     450 REM ---FOUND VERSION 1.1.1 OR 1.2,
                        SO CHANGE "STA" TO "LDA"---
```

```
################################################################################
#  Review: "Apple IIgs ProDOS-16 Reference"
################################################################################
```

Bob Sander-Cederlof

July 1987

Apple says this is "a manual for software developers, advanced programmers, and others who wish to understand the technical aspects of the Apple IIgs operating system." Here is a brief run-down of the contents:

Chapters 1-5 -- About P16 (Files, Memory Management, External Devices, and the Operating Environment)

Chapter 6 -- Programming with P16 (Revising a ProDOS-8 Application, and Using the Apple IIgs Programmer's Workshop)

Chapter 7 -- Adding Routines to P16 (Interrupt Handlers)

Chapters 8-13 -- Making P16 Calls (Complete description of all of the MLI calls supported by P16)

Chapters 14-17 -- The System Loader (How to use the system loader to load and relocate programs, including a description of the 16 System Loader Calls.)

Appendix A -- P16 File Organization (Exactly the same as ProDOS-8, except that more file types are defined)

Appendix B -- Comparison of Apple II Operating Systems

Appendix C -- The ProDOS 16 Exerciser (Tells about the disk which comes with the book)

Appendix D -- System Loader Technical Data (Most of the information about the Object module format expected by the System Loader. More detail will be available someday in the "Apple IIgs Programmer's Workshop Reference".)

Appendix E -- Complete list of Error Codes for P16 and the System Loader

There is a good index, as well as a glossary. And to cap it off, a rather complete Reference Card. The card is printed with major headings in red ink, to make it easier to locate items in a hurry. It totals eight full-size pages, and includes all of the MLI calls, System Loader calls, and Error Codes. Most of the info you need to understand and build file description blocks is also included.

All of the important information is here. However, there are no programming examples. I suspect there were not any good ones available at the time the book was written. We still feel the need for a book like Gary Little's "Apple ProDOS Advanced Features" (which we cannot get anymore) which would lead us through step-by-step in writing ProDOS-16 programs. Gary's book was for the old ProDOS-8, but he or someone should bring out a ProDOS-16 book like it. We also wish Don Worth and Pieter Lechner would give us the equivalent to "Beneath Apple DOS" and "Beneath Apple ProDOS". This is probably asking too much, considering the size of the job.

The publisher's price for this 338-page book is $29.95, and it will be available at most bookstores. Or, you can order it from us for $27, plus shipping.

```
################################################################################
#  Another ProDOS-8 Bug in the IIgs
################################################################################
```

Bob Sander-Cederlof

July 1987

Back in December of 1986 we noticed that a target file written from the ProDOS S-C Macro Assembler when running on a IIgs contained garbage from locations $9B through $FF of every page of the file. We patched the Assembler at that time and made it work correctly, blaming the new version of ProDOS.

The problem seemed to be related to the fact that target file processing used a one-byte data buffer at location $009A (yes, in page zero). Now, there is nothing in any ProDOS documentation warning against using a data buffer in page zero. Furthermore, ProDOS does not return any error code for such a buffer. I assume, and I still think I am correct, that the designers of ProDOS expected this to be legal. Nevertheless, it does not work correctly in the IIgs.

It turns out ProDOS was only indirectly at fault. Both the old and the new versions of ProDOS-8 show the same failure, but it is due to the 65816 processor rather than any changes to the ProDOS code.

The code at fault is the subroutine which transfers bytes of data from the caller's data buffer into the file buffer. This subroutine is at $F326 in ProDOS 1.1.1; it is at $F311 in versions 1.2, 1.3, and 1.4. The file buffer is the one specified when MLI was called to OPEN the file. (The one that always has to begin on a page boundary.)

This subroutine uses pointers at $4E,4D and $4C,4D to access the data buffer and file buffer, respectively. To simplify indexing, a trick is used. It is the trick that causes it to fail with pagezero data buffers in a IIgs.

A subroutine at $F110 (in version 1.1.1) or $F0F8 (later versions) sets up the two pointers. The pointer to the data buffer is modified to point some distance BEFORE the actual data buffer. The distance is equivalent to the low-order byte of the current file MARK. This way the same Y-register value can be used to index both the data and file buffers. Except in a IIgs, when the data buffer is in page zero.

For example, here are the data buffer pointer and Y-register values for three cases that might occur during a ".TF" write:

```
       data buffer at $009A
       file buffer at $7C00
                                eff.addr  eff.addr
       mark   $4E,4F  Y-reg   non-IIgs   IIgs
       -----  ------  -----   --------  --------
       $xx99   $0001   $99     $009A    $00/009A
       $xx9A   $0000   $9A     $009A    $00/009A
       $xx9B   $FFFF   $9B     $009A    $01/009A
```

Notice that the last value on the last line has bank 1, rather than bank 0! For an explanation of how this happens, see the last paragraph on page 119 of "Programming the 65816" by Eyes & Lichty. Whenever an indexed instruction specifies a 16-bit address and assumes the data bank as its bank, then, if the index plus the base exceeds $FFFF the effective address will be in the next bank. (This allows data tables to straddle bank boundaries.)

What happens then, during a write? All bytes from $xx00 through $xx9A of each 256 bytes (in the case of my .TF processor) are written correctly. Bytes from $xx9B through $xxFF are taken instead from bank 1, location $009A (the AUX bank). Whatever data exists there will be written on the file.

I wanted to test out my theories, so I wrote a quick and dirty little program to OPEN a file, WRITE 256 data bytes on it, and CLOSE it. I ran it using both versions 1.1.1 and 1.4 of ProDOS-8, and on both an Apple //e and a IIgs. Both versions of ProDOS worked correctly on the //e, and both failed on the IIgs.

My test program is so "quick and dirty" that you have to CREATE the file directly before running the program. If you want to try it, type "CREATE TESTFILE,TTXT" before running the program. Then to look at the data, type "BLOAD TESTFILE,A$2000,TTXT" and use the monitor to print out the contents of $2000-20FF. You may also need to change the pathname to that of your test disk.

By the way, the very same problem exists for READ calls using a data buffer in page zero. For example, using a one-byte buffer at $009A would cause all bytes within the file which are at posiitons $xx9B through $xxFF to stored at $01009A in a IIgs. Apparently nobody has tried this yet.

The current ProDOS version of the S-C Macro Assembler works correctly in the IIgs. There have been three changes to make this possible. First, we changed to the most recent release of the PRODOS file. Second, I moved my .TF buffer out of page zero. Third, I modified the "$" monitor section to work with the new IIgs monitor. (This version still works in all older machines as well.) If you have recently acquired a IIgs and need an upgrade to your S-C Macro Assembler, let us know.

Don't you suppose that there are more programs out there besides ProDOS which could stumble over this difference in the way indexing works? And more besides our Assembler which will stumble over this quirk in ProDOS? Be wary.

```
1000 *SAVE S.TEST.WPZ
1010 *--------------------------------
1020 *   TEST WRITING FROM A BUFFER IN PAGE ZERO
1030 *--------------------------------
1040 DATABUF .EQ $9B
1050 MLI     .EQ $BF00
1060 PRBYTE .EQ $FDDA
1070 *--------------------------------
1080 T
1090        JSR MLI      OPEN THE FILE
1100        .DA #$C8,IOB.OPEN
1110        BCS .99      ERROR
1120        LDA O.REF    GET THE REFERENCE NUMBER
1130        STA W.REF
1140 *--------------------------------
1150        LDA #0       WRITE $00...$FF ON THE FILE
1160        STA DATABUF
1170 .1     JSR MLI
1180        .DA #$CB,IOB.WRITE
1190        BCS .99      ERROR
1200        INC DATABUF
```

```
1210        BNE .1
1220 *--------------------------------
1230        JSR MLI
1240        .DA #$CC,IOB.CLOSE
1250        BCS .99      ERROR
1260        RTS
1270 *--------------------------------
1280 .99    JMP PRBYTE   PRINT THE ERROR CODE
1290 *--------------------------------
1300 IOB.OPEN
1310        .DA #3
1320        .DA PATHNAME
1330        .DA FILEBUF
1340 O.REF  .BS 1
1350 *--------------------------------
1360 IOB.WRITE
1370        .DA #4
1380 W.REF  .BS 1
1390        .DA DATABUF
1400        .DA 1
1410 ACTLEN .BS 2
1420 *--------------------------------
1430 IOB.CLOSE
1440        .DA #1
1450        .DA #0
1460 *--------------------------------
1470 PATHNAME
1480        .DA #PSZ-1
1490        .AS "/TEST/TESTFILE"
1500 PSZ    .EQ *-PATHNAME
1510 *--------------------------------
1520        .BS *+255/256*256-*   FORCE PAGE BOUNDARY
1530 FILEBUF    .BS 512           FOR FILE BUFFER
1540 *--------------------------------
9999        .LIF
```

```
################################################################################
#  How to Clear the Back-Up Bit
################################################################################
```

Bob Sander-Cederlof

October 1987

I received a letter from Paul R. Santa-Maria today, with a very good question: "How is the backup bit in the file access byte cleared in ProDOS 8?" Paul is writing a program that can use the backup bit, but he needs to be able to clear it.

The information about this bit in the various reference manuals is contradictory and incomplete. Apple's ProDOS Technical Reference Manual (even the new ProDOS-8 edition) says:

ProDOS sets bit 5, the backup bit, of the access field to 1 whenever the file is changed (that is, after a CREATE, RENAME, CLOSE after WRITE, or SET_FILE_INFO operation). This bit should be reset to 0 whenever the file is duplicated by a backup program.

Note: ONly ProDOS may change bits 2-4; only backup programs should clear bit 5, using SET_FILE_INFO.

As Paul pointed out in his letter, these two paragraphs contradict each other. Other references to "backup bit" listed in the index did not clear up the difficulty.

Paul noticed that one of the bytes in the System Global Page is called BUBIT (at $BF95). The only explanation of this bit is that it can be changed before MLI calls, and a comment "BACKUP BIT DISABLE, SETFILEINFO ONLY".

Neither of us could find any further information in Apple's manuals, or even in the various third-party books.

I did get some help from the supplement to "Beneath Apple ProDOS", and also from my Apple itself. First I did a search of the ProDOS code while it was in RAM and found two references to $BF95, at $DE7A and at $F7EF. (These are the addresses in Version 1.1.1, and are slightly different from the addresses in Version 1.2, 1.3, and 1.4.) The first reference is at the general exit from all MLI calls, and it stores a zero at $BF95 (BUBIT). The second is inside the SET FILE INFO processor. Here is a piece of the code:

```
   F7EF-  LDA BUBIT
          EOR #$20
          AND $FE7D        CURRENT ACCESS BITS
          AND #$20         ISOLATE BACKUP BIT
          STA $FEB4
```

According to the BAP Supplement, $FEB4 is later ORed into the Access Bits, immediately before the update is complete.

Apparently the steps necessary to clear the backup bit are:

1. read the current file information using GET FILE INFO;
2. clear the backup bit in the access byte and set at least bit 5 of $BF95 to 1;
3. and use SET FILE INFO to install the change.

I wrote a test program to perform those steps, and it worked!

My program displays some information, so that I can see what it has done. Line 1170 reads the current file info and displays it in hex. The first byte displayed is the byte with the access bits. Lines 1180-1200 clear bit 5, the backup bit, in the access byte. Line 1210 changes BUBIT ($BF95) from $00 to $FF, so that SET FILE INFO will not set the backup bit. Lines 1220-1240 call MLI to SET FILE INFO. Finally, lines 1260-1380 read the file info and display it again, to see if it worked.

To make my test program simple, I assembled the pathname of a file I knew was on the mounted volume. The pathname is in line 1480. You should substitute here the name of the file you really want to play with.

By the way, there is another way to clear the backup bit. You can read and write directory sectors directly, using the READ_BLOCK and WRITE_BLOCK calls. If you are writing a super snazzy backup program, you may want to do it this way. It can be easier to follow the directory tree using such direct access.

```
1000 *SAVE CLEAR.BUBIT
1010 *--------------------------------
1020 MLI    .EQ $BF00
1030 BUBIT  .EQ $BF95
1040 *--------------------------------
1050 BELL   .EQ $FBDD
1060 CROUT  .EQ $FD8E
1070 PRBYTE .EQ $FDDA
1080 COUT   .EQ $FDED
1090 *--------------------------------
1100        .MA MLI
1110        JSR MLI
1120        .DA #]1,]2
1130        BCS ERROR
1140        .EM
1150 *--------------------------------
1160 CLEAR.BUBIT
1170        JSR GET.FILE.INFO.AND.DISPLAY.IT
1180        LDA INFO+3
1190        AND #$DF          CLEAR BACKUP BIT
1200        STA INFO+3
1210        DEC BUBIT         BUBIT = $FF
1220        LDA #$07
1230        STA INFO
1240        >MLI $C3,INFO     SET INFO, CLEARING BUBIT
1250 *--------------------------------
1260 GET.FILE.INFO.AND.DISPLAY.IT
1270        LDA #$0A
1280        STA INFO
1290        >MLI $C4,INFO     READ AND DISPLAY NEW INFO
1300        LDY #3
1310 .1     LDA INFO,Y
1320        JSR PRBYTE
1330        LDA #"."
1340        JSR COUT
1350        INY
1360        CPY #18
```

```
1370        BCC .1
1380        JMP CROUT
1390 *--------------------------------
1400 ERROR  JSR PRBYTE
1410        JMP BELL
1420 *--------------------------------
1430 INFO   .HS 0A
1440        .DA PATH
1450        .BS 15
1460 *--------------------------------
1470 PATH   .DA #LEN
1480        .AS /PRODOS/
1490 LEN    .EQ *-PATH-1
1500 *--------------------------------
```

##
It's 1988, and ProDOS Thinks it's 1982
##

Bob Sander-Cederlof

December 1987

If you are still using ProDOS 1.1.1, and you have some sort of clock card such as Thunderclock, TimeMaster, or any other "standard" ProDOS clock, you have a problem. Apple built this bug into ProDOS, and they came out with the new versions (they call it ProDOS-8 version 1.4 now) just in time.

In my article about the clock driver in the November 1983 issue of AAL (pages 25-28), I discussed the problem. It seemed a little more remote at the time. Apple based ProDOS on the Thunderclock, even though that device does not keep track of the year. The ProDOS clock driver reads the Month, Day, and Day of Week information and does some arithmetic to determine which of six years could produce that day of week on the corresponding month and day. ProDOS 1.1.1 and earlier versions could produce dates from 1982 through 1987. When 1988 rolled around a few weeks ago, hundreds of thousands of Applers around the world slipped back in time to 1982.

And it is not funny! Some programs will not let you operate if the dates are not correct!

Well, there are at least four ways around the problem. You can remove your clock card, and type the date in manually wherever it is really needed. Not very nice.

Or, you can get the up-to-date version of ProDOS, now called ProDOS-8 Version 1.4. You can get it, and then you can copy it to every floppy (both 3 1/2 and 5 1/4), to every RamFactor, to every hard disk in sight. This is tedious, but it is the best solution. If you have a friendly dealer, you can get it from the IIgs system disk. But don't copy the file named PRODOS from this disk (that is only a loader now). Instead, copy the file named P8 from the subdirectory SYSTEM. P8 is a longer file than version 1.1.1 of PRODOS was, so if you use BSAVE to put it on your disks be sure to specify the L parameter. Something like this should do the trick:

Boot any ProDOS disk, preferably one with version 1.4 so the correct dates will get into the file directories you are updating. Get into the S-C Macro Assembler or Applesoft. With the latest IIgs system disk in your drive, type:

```
BLOAD SYSTEM/P8,TSYS,A$2000
```

Now put the disk you want to update into a drive, and type the following. You may want to include slot and drive parameters, or set the prefix to the appropriate value for a ram disk or hard disk.

```
UNLOCK PRODOS
BSAVE PRODOS,TSYS,A$2000,L$3C7D
LOCK PRODOS
```

A third approach saves you a trip to the dealer. You can simply PATCH the copies of ProDOS version 1.1.1 to give you the correct year. When you BLOAD the file named PRODOS at $2000, the six-year table is at $4F76. If you look there now you will find the following bytes:

```
4F76: 54 54 53 52 57 56 55
```

These correspond to the years 1984, 1984, 1983, 1982, 1987, 1986, and 1985. Notice that 1984, being a leap year, takes up two of the values. Patch these seven bytes, using the monitor, as follows:

```
4F76:5A 59 58 58 57 56 5B
```

The table now includes the years from 1986 through 1991. If you want 1992 in there also, substitute 5C where I have 57 and 56 above. Both 1988 and 1992 are leap years, so they both take two table positions. When ProDOS 1.4 was released it was still 1987, so there was not room for 1992 in the table.

A fourth possible solution was suggested by reader Garth O'Donnell. You can replace the clock driver inside ProDOS with one that reads the year directly from your clock card! This is what happens when you boot Version 1.4 in a IIgs, because P8 senses that you are in a IIgs and plugs in a different driver. But if you are still using an older Apple, as most of us are, you can modify the PRODOS file to load an intelligent driver for your own clock card. Of course, if you are using a Thunderclock, the driver with the above patches is the best you can do. But if you have a TimeMaster, as Garth does, you can use a program like he wrote.

I decided to try my hand at modifying the standard clock driver so that it uses the year information in the TimeMaster. The following program is derived directly from the standard driver, with as few modifications as possible. It still resides in the ProDOS SYS file at $4F00, but it is a lot shorter. (Maybe you can think of something useful to do with the extra 45 bytes!) It still depends upon the standard ProDOS loader to plug in the actual slot number in lines 1260 and 1310. The major change I made was to call on the ":" instead of the "#" mode. The "#" mode is a ThunderClock mode, which does not return the year. The ":" mode is a TimeMaster mode, which does return the year.

If you have an Applied Engineering Serial Pro card, which includes a TimeMaster compatible clock, you can use the driver I wrote by making the single change as shown in the comments on line 1090. Or, maybe you could use those extra 45 bytes for a subroutine that would check which clock is in the slot and make the appropriate changes at run time.

```
1000 *SAVE S.CLOCK.1988
1010 *--------------------------------
1020 *  IF THE PRODOS BOOT RECOGNIZES A TIMEMASTER,
1030 *  A "JMP $D742" IS INSTALLED AT $BF06 AND
1040 *  THE SLOT ADDRESS IS PATCHED INTO THE FOLLOWING
1050 *  CODE AT SLOT.A AND SLOT.B BELOW.
1060 *--------------------------------
1070 *   DEFINE CLOCK ENTRY POINT
1080 *--------------------------------
1090 CLOCK  .EQ $C108    <<<USE $C11D FOR AE SERIAL PRO>>>
1100 *--------------------------------
1110 DATE   .EQ $BF90    $BF91 = YYYYYYYM
1120 *                   $BF90 = MMMDDDDD
1130 TIME   .EQ DATE+2   $BF93 = 000HHHHH
1140 *                   $BF92 = 00MMMMMM
1150 MODE   .EQ $5F8-$C0 TIMEMASTER MODE IN SCREEN HOLE
1160 *--------------------------------
1170        .OR $4F00
```

```
1180        .TF B.CLOCK.DRIVER
1190        .PH $D742
1200 *--------------------------------
1210 PRODOS.TIMEMASTER.DRIVER
1220        LDX SLOT.B   $CN
1230        LDA MODE,X   SAVE CURRENT TIMEMASTER MODE
1240        PHA
1250        LDA #":"     SEND ":" TO TIMEMASTER
1260        JSR CLOCK+3        SELECT TIMEMASTER MODE
1270 SLOT.A .EQ *-1
1280 *--------------------------------
1290 *      READ TIME & DATE INTO $200...$211 IN FORMAT:
1300 *--------------------------------
1310        JSR CLOCK
1320 SLOT.B .EQ *-1
1330 *--------------------------------
1340 *      CONVERT ASCII VALUES TO BINARY
1350 *      $3E -- MINUTE
1360 *      $3D -- HOUR
1370 *      $3C -- YEAR
1380 *      $3B -- DAY OF MONTH
1390 *      $3A -- MONTH
1400 *--------------------------------
1410        CLC
1420        LDX #4
1430        LDY #12      POINT AT MINUTE
1440 .1     LDA $203,Y   TEN'S DIGIT
1450        AND #$0F     IGNORE TOP BIT
1460        STA $3A      MULTIPLY DIGIT BY TEN
1470        ASL          *2
1480        ASL          *4
1490        ADC $3A      *5
1500        ASL          *10
1510        ADC $204,Y   ADD UNIT'S DIGIT
1520        SEC
1530        SBC #$B0     SUBTRACT ASCII ZERO
1540        STA $3A,X    STORE VALUE
1550        DEY          BACK UP TO PREVIOUS FIELD
1560        DEY
1570        DEY
1580        DEX          BACK UP TO PREVIOUS VALUE
1590        BPL .1       ...UNTIL ALL 5 FIELDS CONVERTED
1600 *--------------------------------
1610 *      PACK MONTH AND DAY OF MONTH,
1620 *--------------------------------
1630        TAY          MONTH (1...12)
1640        LSR          00000ABC--D
1650        ROR          D00000AB--C
1660        ROR          CD00000A--B
1670        ROR          BCD00000--A
1680        ORA $3B      MERGE DAY OF MONTH
1690        STA DATE     SAVE PACKED DAY AND MONTH
1700 *--------------------------------
1710        LDA $3C      YEAR
1720        ROL          MERGE TOP MONTH BIT
1730        STA DATE+1   YYYYYYYM
```

```
*--------------------------------
       LDA $3D      GET HOUR
       STA TIME+1
       LDA $3E      GET MINUTE
       STA TIME
       PLA          RESTORE TIMEMASTER MODE
       LDX SLOT.B   GET $CN FOR INDEX
       STA MODE,X
       RTS
*--------------------------------
       .EP
*--------------------------------
```

##
Printing the ProDOS Date and Time
##

Bob Sander-Cederlof

February 1988

ProDOS-8 stores the date and time information from in four bytes in the Global Page starting at $BF90:

```
     $BF90:  MMMDDDDD   Low-order bits of Month, Day (1-31)
     $BF91:  YYYYYYYM   Year (0-99), high bit of Month
     $BF92:  00mmmmmm   Minute (0-59)
     $BF93:  00hhhhhh   Hour (0-23)
```

The following subroutine, lines 1000-1590, will print out the date in the form DD-MMM-YY. Lines 1600-1800 are an alternative method for printing out the 3-letter month name abbreviation. Lines 1810-1910 print the time in the form hh:mm.

The value stored in the Global Page may not be current. It is automatically updated every time you close or flush a file, or you can force it to be updated by using the MLI call shown in lines 1920-end. If you have looked into the Global Page description in the manuals, you may have noticed that $BF06 is a vector to the date/time update code. Don't try to use it directly unless you are sure the Language Card is properly switched before and after the call. The best way is to use the MLI call, as I did.

```
  1000 *SAVE PRINT.DATE
  1010        .LIST MOFF
  1020 *--------------------------------
  1030 *   Subroutine to print date from ProDOS Global Page
  1040 *      in form DD-MMM-YY.
  1050 *   Two different methods for printing the 3-letter month
  1060 *      name are shown, with month-name table in normal and
  1070 *      transposed order.
  1080 *--------------------------------
  1090 DATE   .EQ $BF90,BF91   Date in form: MMMDDDDD, YYYYYYYM
  1100 *                       Time in form: 00mmmmmm, 000hhhhh
  1110 COUT   .EQ $FDED
  1120 *--------------------------------
  1130 PRINT.DATE
  1140        LDA DATE     Get MMMDDDDD
  1150        AND #$1F     Isolate Day of Month
  1160        JSR PD       Print the day number
  1170        LDA #"-      Print a dash
  1180        JSR COUT
  1190 *----PRINT MONTH FROM TABLE------
  1200        LDA DATE+1   Get YYYYYYYM
  1210        LSR          High bit of Month-number into Carry
  1220        PHA          Save 0YYYYYYY on stack
  1230        LDA DATE     Get MMMDDDDD
  1240        ROR          MMMMDDDD
  1250        LSR          0MMMMDDD
  1260        LSR          00MMMMDD
  1270        LSR          000MMMMD
  1280        LSR          0000MMMM   Month number (1-12)
```

```
1290        TAX
1300        LDA MONTH.TBL.1-1,X  1st letter
1310        JSR COUT
1320        LDA MONTH.TBL.2-1,X  2nd letter
1330        JSR COUT
1340        LDA MONTH.TBL.3-1,X  3rd letter
1350        JSR COUT
1360        LDA #"-      Print dash
1370        JSR COUT
1380 *----PRINT YEAR------------------
1390        PLA          GET 0YYYYYYY FROM STACK
1400 *---Fall into PD subroutine------
1410 PD     LDX #"0"-1   Start with ASCII zero-1
1420        SEC          Set up subtraction
1430 .1     INX          Increment ten's digit
1440        SBC #10      Take out ten
1450        BCS .1       Still more tens
1460        ADC #"0"+10  Add back one ten, and make ASCII
1470        PHA          Save unit's digit
1480        TXA          Get ten's digit
1490        JSR COUT     Print ten's digit
1500        PLA          Get unit's digit
1510        JMP COUT     and print it
1520 *--------------------------------
1530        .MA AS
1540        .AS -/]1/
1550        .EM
1560 *--------------------------------
1570 MONTH.TBL.1 >AS "JFMAMJJASOND"
1580 MONTH.TBL.2 >AS "AEAPAUUUECOE"
1590 MONTH.TBL.3 >AS "NBRRYNLGPTVC"
1600 *--------------------------------
1610 ALTERNATIVE.MONTH.PRINTER
1620        LDA DATE+1   GET YYYYYYYM
1630        LSR          M INTO CARRY
1640        LDA DATE     GET MMMDDDDD
1650        ROR          MMMMDDDD
1660        LSR          0MMMMDDD
1670        LSR          00MMMMDD
1680        LSR          000MMMMD
1690        LSR          0000MMMM
1700        STA TEMP     Multiply month number by 3
1710        ASL
1720        ADC TEMP
1730        TAX          Index is 3,6,9,...
1740        LDY #3       Print 3 consecutive letters
1750 .1     LDA MONTH.TABLE-3,X
1760        JSR COUT
1770        INX          Next letter
1780        DEY
1790        BNE .1
1800        RTS          Finished
1810 *--------------------------------
1820 TEMP   .BS 1
1830 MONTH.TABLE >AS "JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC"
1840 *--------------------------------
```

```
1850 PRINT.TIME
1860        LDA DATE+3   Get 00hhhhhh
1870        JSR PD
1880        LDA #":"
1890        JSR COUT
1900        LDA DATE+2   Get 00mmmmmm
1910        JMP PD
1920 *---------------------------------
1930 UPDATE.DATE.AND.TIME
1940        JSR $BF00    MLI ENTRY POINT
1950        .DA #$82,0000     GET DATE/TIME, NO PARMS
1960        RTS
1970 *---------------------------------
1980        .LIST OFF
```

```
################################################################################
#  BLOADing Directories
################################################################################
```

Bob Sander-Cederlof

April 1988

Did you know that ProDOS will let you BLOAD a directory just like any other kind of file? I did not until today.

For example, if I want to load the directory of my Sider hard disk into memory, I can type BLOAD /HARD1,TDIR,A$1000. I can load in any subdirectory the same way. This works under both BASIC.SYSTEM (Applesoft) and SCASM.SYSTEM (the S-C Macro Assembler) shells.

In both cases, if you care to, you can find the length of the directory in bytes in locations $BEDB and $BEDC. $BEDB will always contain 00, and $BEDC will be the number of pages in the directory, or twice the number of blocks.

I tried BSAVEing... but it is prohibited. You get a FILE LOCKED message for your efforts.

###