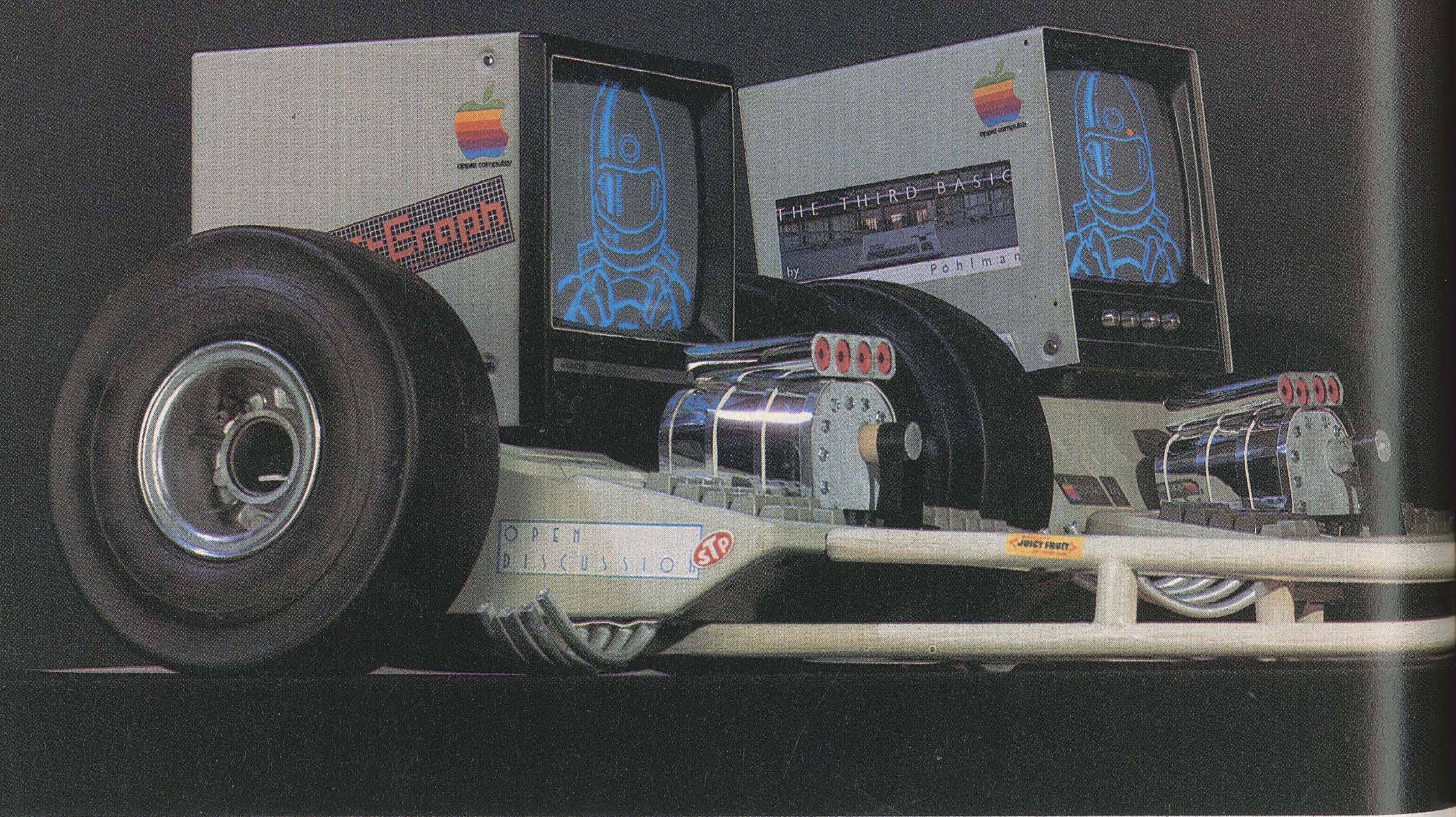
HOT ROD III



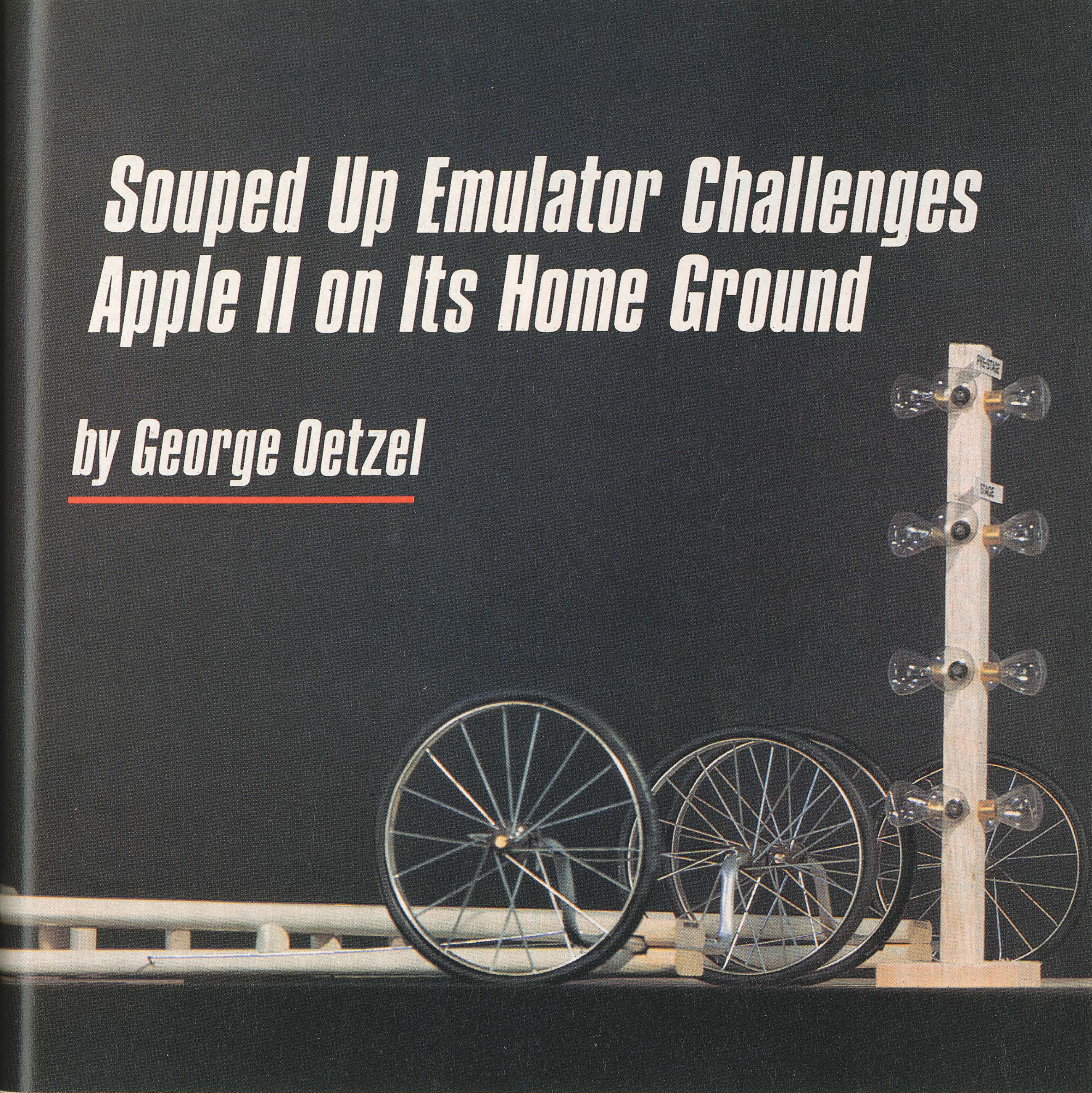
The folks at Apple tried hard to pretend that the Apple III is not a hobby computer. Granted, they provided an excellent operating system (SOS) and many other features that are desirable for a small-business computer. They also provided Apple II Emulation, which has everything needed to warm the hobbyist's heart. If Apple had provided a switch that turned the Apple III into a perfect imitation of the Apple II, it wouldn't have been very interesting. They didn't do that. The Emulation mode is reminiscent of a poor vaudeville mimic. You can recognize the character of the Apple II, but you don't have to look very far to tell that it isn't the real thing. This poor imitation has always seemed to be the bad news about the Apple II Emulation mode. The good news is that it is mostly done in software, so it can be changed. With a little imagination, you can make the Apple III emulate some versions of the Apple II that the company never built. That's where the excitement begins.

This series of three articles will describe the important hardware differences between the two machines, the organization of information on the Emulation disk, and specific custom Emulation modes. One will allow you to play certain Apple II games that couldn't be played before on the Apple III. Another gives you the use of the full keyboard and lower-case display. In a more exotic version, you can run Applesoft with full access to the Apple III hardware. It's a project for the computer hobbyist, with disk editing, assembly and disassembly of various program segments, and hardware details of two machines. With a little effort, you'll end up with the freedom to sit down at the keyboard and design a custom Apple to your liking. You will also understand a lot more about the operation of the Apple III in all of its personalities.

Emulation Disk Organization. Let's start with a discussion of the Emulation program and the disk on which it is distributed. The Emulation disk may seem a bit of a mystery, because it has no directory. It contains a straightforward program and copies of both Applesoft and

Souped Up Emulator Challenges Apple II on Its Home Ground

by George Oetzel



Integer Basic, all of which are loaded into memory when you boot the disk. All of the useful data is on disk tracks 0 through 9, but the entire disk is formatted so that it can be copied easily. Any Apple copy utility, such as the Apple III *System Utilities* or the *CopyA* program distributed with the DOS 3.3 master disk, will suffice.

When you press control-reset, a program in ROM loads disk block 0 into addresses $A000 through $A1FF and then does a jump to $A000. On the Emulation disk, this 512-byte boot program first checks to see that it's not in an Apple II environment and then loads the rest of the Emulation program and both versions of Basic into memory. Of course, both Basics can't be loaded in their ultimate memory locations because the Apple III has no language card. All the code (ROM in the Apple II) associated with Integer Basic is loaded into addresses $2000 through $5AFF. All the code (ROM in the Apple II Plus) associated with Applesoft goes into $5B00 through $95FF. The Emulation program fills $A000 through $B670, memory that later becomes part of DOS. Table 1 shows the details of the memory organization, along with the disk block numbers that correspond to each memory segment.

Most of the Emulation program involves responses to all the setup menu choices—which version of Basic do you want, and what imitation I/O card should be connected to the RS-232 port? When you hit return, the appropriate segments are loaded into high memory, the machine control registers are set for the Emulation mode, memory above $C000 is write-protected, and control transfers to the Apple II auto-start routine.

The organization of the Emulation disk is wonderfully simple. You may have noticed that a disk track includes exactly the same amount of data as can be stored in all the addresses beginning with a given hex digit—for example, $2000 through $2FFF. The people at Apple have certainly noticed, because everything that will be located in addresses $2*xxx* is on disk track 2, $3*xxx* corresponds with disk track 3, and so forth. The Emulation program is on tracks 0 and 1, with addresses $A*xxx* on track 0 and $B*xxx* on track 1. Table 1 not only provides a

Destination Address	Boot Address	Disk Block	Description
	Integer Basic Image		
C500–C5FF	2000–20FF	10	Slot 5 (Comm card) ROM
C600–C6FF	2100–21FF	10	Slot 6 (disk) ROM
C700–C7FF	2200–22FF	11	Slot 7 (Comm card) ROM
C800–CFFF	2300–2AFF	12–15	Expansion I/O ROM (empty)
D000–D7FF	2B00–32FF	15–21	Programmers aid #1
D800–DFFF	3300–3AFF	21–23	D8 ROM (empty)
E000–F7FF	3B00–52FF	23–29	Integer Basic
F800–FFFF	5300–5AFF	29–2D	Autostart Monitor
	Applesoft Basic Image		
C500–C5FF	5B00–5BFF	2D	Slot 5 (serial card) ROM
C600–C6FF	5C00–5CFF	2E	Slot 6 (disk) ROM
C700–C7FF	5D00–5DFF	2E	Slot 7 (Comm card) ROM
C800–CFFF	5E00–65FF	2F–32	Expansion I/O ROM (empty)
D000–F7FF	6600–8DFF	33–46	Applesoft Basic
F800–FFFF	8E00–95FF	47–4A	Autostart Monitor

Table 1. Address guide to the two Basic images after booting the Emulation disk. All addresses and disk blocks are hexadecimal values.

guide to the location in memory of the Emulation ROM image, but it also tells you where to look for the data on the Emulation disk. All you need are good tools allowing you to examine and modify the contents of the disk. Table 2 gives the rules for locating the disk block numbers, or the Apple II track and sector numbers, that contain the data for specific addresses in the Basic images.

Emulation Memory Page	Apple III Block Number	Apple II Track	DOS 3.3 Sector
N000	B0 (See note)	N	0
N100	B0	N	E
N200	B0 + 1	N	D
N300	B0 + 1	N	C
N400	B0 + 2	N	B
N500	B0 + 2	N	A
N600	B0 + 3	N	9
N700	B0 + 3	N	8
N800	B0 + 4	N	7
N900	B0 + 4	N	6
NA00	B0 + 5	N	5
NB00	B0 + 5	N	4
NC00	B0 + 6	N	3
ND00	B0 + 6	N	2
NE00	B0 + 7	N	1
NF00	B0 + 7	N	F

Note: B0 = ($10)*N/2. Computation for address $3500: Block number is ($10 * $3)/2 + $2 = $18 + $2 = $1A. Block $1A contains $3400–$35FF.

Table 2. Emulation disk Basic image location guide for Apple II and Apple III utilities.

While it is feasible to change the Emulation disk with any of numerous Apple II track/sector editors, it is easier to load patch programs and ensure that modifications look right if you edit a whole track, or two, at a time. Since editors for entire disk tracks are uncommon, a special program is in order. The *Trackmover* program in listing 1 at the end of this article is written in Integer Basic. Programming in Integer is unlikely to fill you with nostalgia for the early days of the Apple II. It's useful for modifying the Emulation and game programs, because you will probably want the miniassembler that comes with Integer Basic. You can also use the memory space from $D800 to $DFFF for utilities such as *The Inspector*, a first-rate track/sector utility from Omega Microware.

The comments included with the *Trackmover* listing explain the program logic and the peeks, pokes, and calls that make up for the small set of commands in Integer Basic. The machine language subroutines poked into memory in lines 2000 through 2080 obtain and save the IOB address and do the RWTS calls. Listing 2 shows this routine in assembly form, but as the Basic program pokes it into memory, you don't need to center listing 2. The IOB table that controls RWTS is explained in the

DOS 3.3 manual and, in more detail, in the book *Beneath Apple DOS* by Worth and Lechner. The sectors from each track are loaded in the order listed in table 2.

Start a modified Emulation disk with a copy of the original. Then use *Trackmover* to load tracks from the copy into memory. You can either make modifications immediately (using the Apple II Monitor) and rewrite the tracks on your custom Emulation disk, or you can save partially edited tracks in a DOS 3.3 binary file. The DOS file can then be reloaded and edited any time, and the *Trackmover* program will rewrite the tracks on the Emulation disk.

The next two articles will describe major modifications to the Emulation disk, but here is a useful change you can make to try out the procedure. Apple II programs often control the reset vector so that the Apple II must be turned off and rebooted to run another program. It's a double nuisance on the Apple III, because you have to reboot the Emulation disk first and then boot the next Apple II disk. You can take control of this process by changing the Monitor reset vector to the "old" Monitor entry point. Then the reset in the Apple II mode will result in the Monitor asterisk prompt. You can reboot with 6 control-P return.

There are two copies of the Monitor on the Emulation disk, and you will have to change both. Use *Trackmover* to load track 5 from the Emulation disk into a suitable Apple II location, say $5000. Go to the Monitor and dump the contents of $5AF0 through $5AFF:

```
*5AF0.5AFF
5AF0 - 83 7F 5D CC B5 FC 17 17
5AF8 - F5 03 62 FA 62 FA 40 FA
```

Now, try FFF0.FFFF. The contents of these addresses should be the same. If they aren't, you have a problem, either with your copy of *Trackmover* or with an operator malfunction. If they are the same, type

```
*5AFA:59 FF 59 FF
*5AF8.5AFF
5AF8 - F5 03 59 FF 59 FF 40 FA
```

This changes the nonmaskable interrupt and reset vectors so that they go to the Monitor cold-start entry point rather than to the auto-start routine. Return to Basic and the *Trackmover* program and rewrite the modified Monitor on track 5 of the Emulation disk. Next load track 9 into memory. Let's use $5000 again. This time the Monitor isn't in the same memory pages. Type

```
*55F8.55FF
55F8 - F5 03 62 FA 62 FA 40 FA
```

Does that look familiar? Sure enough.

```
*55FA:59 FF 59 FF
*55F8.55FF
55F8 - F5 03 59 FF 59 FF 40 FA
```

This procedure should look familiar, too. Return to Basic and use the *Trackmover* to replace the modified Monitor on track 9. Put the modified disk in the internal drive and press control-reset to reboot with your modified Emulation program. Load the Apple II program of your choice and press reset. Voila, the Monitor asterisk! Now, you have control of your computer.

Apple III Hardware. The most noticeable difference between the Emulation mode and a real Apple II is the big change in the game paddles. Many games designed for the Apple II won't run on the Apple III.

The Apple III has an eight-input, multiplexed analog-to-digital converter (A/D) to read the game paddles. Only four of its inputs are routed to the game ports on the back of the machine. The A/D measures the voltage applied to its terminals. The Apple II measures the resistance between them. Although the *Owner's Guide* presents a paddle circuit on page 130 and suggests that resistors from 1K to 700K can be used, don't try rewiring the 150K-ohm potentiometers from Apple II paddles into this circuit. Control becomes quite unsatisfactory with resistors much larger than about 5K ohms.

The software required to read the paddles on the two machines is very different and will be discussed in detail in part 2. Games that have internal routines to read the paddles don't work on the Apple III. Games that use the routines in the Monitor do work, because the Monitor subroutine has the same entry address and calling parameters. Many games that use joysticks use the Monitor routine. Virtually none of the single-paddle games do.

There is a widespread rumor that the Apple III won't generate color in the Emulation mode. That's partly true. If you get a high-priced RGB monitor, you won't get color displays in Emulation mode. The single-connector, composite (NTSC) color monitors don't have good enough resolution for satisfactory use with the normal eighty-column text display, but the NTSC color works both in Emulation mode and native mode. In spite of the fact that the label on the B/W video connector remains unchanged, a recent modification has routed the color video signal to that connector. On older machines, the fifteen-pin color video connector must be used for color displays. The fifteen-pin connection is a construction project of the ten-minute variety, using easy-to-get parts. The best advice for the Apple III owner who wants to use color is to get an NTSC color monitor to use only when color displays are desirable and stick to the "green screen" the rest of the time. Using both video ports, both can be connected all the time, and the cost of the two monitors is less than that of a single RGB color monitor.

The Apple III has three sound generators, only two of which can be used in Emulation mode. One is the Apple II standard that makes a click with every memory reference to addresses in the $C02x range. The second, activated by $C04x memory references, generates a short tone at about 1 kHz. It is used as the beep in most Apple III applications. The third is a six-bit D/A converter connected to the same 6522 VIA chip that controls memory bank selection (at address $FFE0). It is responsible for the audible message from the system diagnostic program: "I'm okay; system is normal." The logic that turns on the Emulation mode disables access to the 6522.

Chips for the Apple III system clock are now available in quantity. If you get one and want to read the clock in Emulation mode, you are out of luck. The assembly language instruction to read a clock byte is LDA $C070, but the only byte accessible in the Emulation mode is the milleseconds byte. The other seven bytes are switched in by changing the zero-page register ($FFD0), a function that is possible only in native mode.

A very large number of Apple II owners have modified their computers to display lower-case characters and accept lower-case input from the keyboard. As a result, Apple II software that expects a lower-case display is rather common. It will undoubtedly become more common with the introduction of the Apple IIe. Since both of those functions are normal to the Apple III, it seems at first that it should be simple to make the changes in the Emulation mode. The display is easy to fix. The Apple II character set is a part of the Emulation program. Entry of lower-case characters is complicated by the fact that the Apple III keys are encoded in two bytes. Apple II software normally reads only the byte at $C000, which generates the key codes you see when the alpha lock key is pressed. The shift key has no effect on the alphabetic characters in this byte. To determine whether they are intended to be upper or lower case, it is necessary to read the B keyboard byte at $C008. Appendix G in the *Standard Device Drivers Manual* explains the bits in byte B. To use the lower-case characters, the Apple II Monitor must be modified to make use of the extra byte and eliminate the masks that convert all entered characters to upper case, regardless of the ASCII code that was input.

The next article is all about games. It includes modifications of the Emulation Monitor and software tools that allow easy conversion of many Apple II games so that they will read the Apple III paddles. A more complete explanation of the Emulation program and the registers that control the Emulation mode will be given in the third article. The discussion will include Emulation program and Monitor modifications that allow full use of lower case and the exotic Emulation modes possible with nonstandard states of the control registers.

```
100  GOTO 2000
120  FOR T=TS TO TE: POKE TR,T: POKE CM,C
140  FOR I=1 TO 16: POKE B1,AD: POKE SC,S(I)
160  CALL RW: REM                  Call RWTS
180  E= PEEK (RC): IF E=0 THEN 220
200  I=16:T=TE: REM                Force end of loop on RWTS
                                   error
220  AD=AD+SZ: NEXT I: NEXT T
240  RETURN
260  H= PEEK (-16384): IF H<127 THEN 260
280  POKE -16368,0:H$=""
300  IF H=206 THEN H$="N"
320  IF H=217 THEN H$="Y"
340  H=H-176
360  IF H>9 THEN PRINT H$
380  IF H<10 THEN PRINT H;
400  RETURN
420  L= LEN(H$)
440  IF L>2 THEN 640: REM          More than 2 digits = error
460  H=0:N=-1000
480  FOR J=1 TO L
500  FOR I=1 TO 16: REM            Locate character in array of hex
                                   digits
520  IF H$(J,J)#HX$(I,I) THEN 560
540  N=I-1:I=16: REM               Character found, fix N
560  NEXT I
580  H=16*H+N: REM                 Calculate return value
600  NEXT J
620  RETURN
640  H=-1000: RETURN
660  VTAB 23: TAB 10: POKE 50,127: PRINT "ERROR - REENTER";:
     POKE 50,255: RETURN
680  VTAB 23: TAB 10: PRINT "                         ";: RETURN
700  CALL -936: REM                Clear screen and home cursor
720  VTAB 2: TAB 1
740  PRINT "DRIVE 1 OR 2? ";: GOSUB 260: PRINT
760  IF (H=1) OR (H=2) THEN 800
780  GOSUB 660: GOTO 720
800  GOSUB 680
```

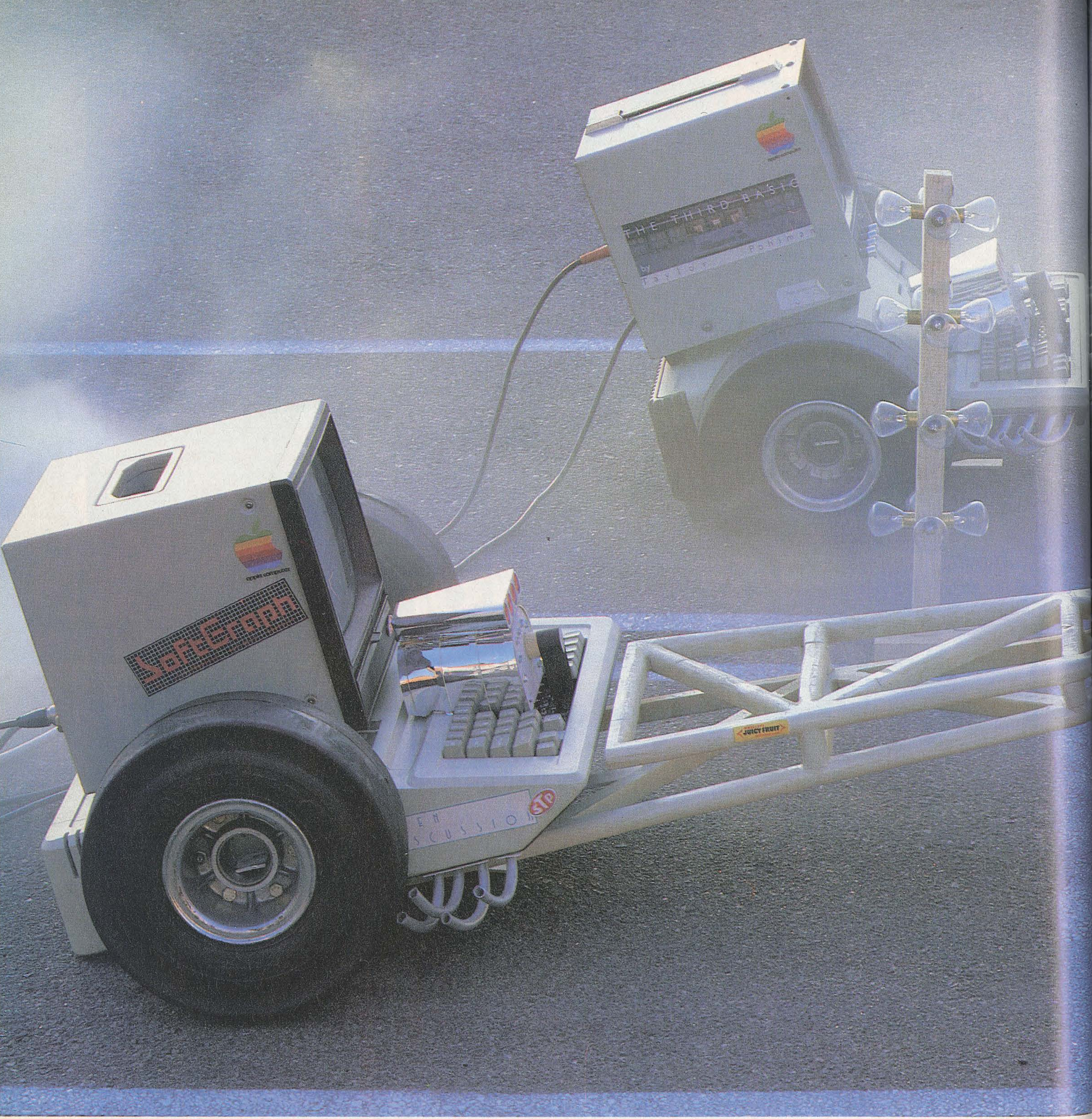
```
820  POKE DR,H: REM              Put selected drive into IOB table
840  VTAB 4: TAB 1
860  PRINT "1 = READ"
880  PRINT "2 = WRITE";: TAB 20: GOSUB 260: PRINT
900  C=1:C$="READ": IF H=1 THEN 960
920  C=2:C$="WRITE": IF H=2 THEN 960
940  GOSUB 660: GOTO 840: REM If not 1 or 2 then error
960  GOSUB 680
980  VTAB 5: TAB 20: POKE 50,63: REM INVERSE
1000 PRINT C$: POKE 50,255: REM NORMAL
1020 GOTO 1100
1040 VTAB 7: TAB 1
1060 FOR I=0 TO 6: FOR J=1 TO 6: PRINT "          ";: NEXT J:
     PRINT : NEXT I
1080 GOSUB 660
1100 VTAB 7: TAB 1
1120 INPUT "START TRACK (HEX) ",H$
1140 GOSUB 420
1160 IF (H<0) OR (H>35) THEN 1040
1180 TS=H
1200 PRINT
1220 GOSUB 680
1240 VTAB 9: TAB 1
1260 INPUT " LAST TRACK (HEX)   ",H$
1280 GOSUB 420
1300 IF (H<0) OR (H>35) THEN 1040
1320 TE=H
1340 T=TE-TS
1360 IF (T<0) OR (T>6) THEN 1040
1380 GOSUB 680
1400 VTAB 11: TAB 1
1420 PRINT "START ADDRESS  $";: GOSUB 260
1440 IF H>0 AND (H+T<8) THEN 1480
1460 GOSUB 660: GOTO 1400
1480 PRINT "000"
1500 GOSUB 680
1520 AD=H*16
1540 VTAB 14: TAB 10
1560 PRINT "ALL OK? (Y/N) ";: GOSUB 260
1580 IF H$="N" THEN 700
1600 IF H$="Y" THEN 1640
1620 GOSUB 660: GOTO 1540
1640 POKE B0,0: POKE VL,0
1660 GOSUB 680
1680 GOSUB 120
1700 IF E=0 THEN 1840
1720 VTAB 17: TAB 1
1740 POKE 50,63: PRINT "ERROR";: POKE 50,255
1760 I=E/16:J=E-16*I
1780 PRINT " CODE = ";HX$(I+1);HX$(J+1)
1800 GOTO 1880
1820 GOSUB 120
1840 VTAB 17: TAB 18: POKE 50,63
1860 PRINT "DONE": POKE 50,255
1880 VTAB 20: TAB 5
1900 PRINT "MORE? (Y/N) ";: GOSUB 260
1920 IF H$="Y" THEN 700
1940 PRINT : PRINT "END"
1960 CALL -1233
1980 END
2000 RW=768:SZ=1
2020 POKE 768,32: POKE 769,227: POKE 770,3: POKE 771,32:
     POKE 772,217: POKE 773,3
2040 POKE 774,176: POKE 775,6: POKE 776,160: POKE 777,13:
     POKE 778,169
2060 POKE 779,0: POKE 780,145: POKE 781,0: POKE 782,96:
     POKE 783,32: POKE 784,227
2080 POKE 785,3: POKE 786,132: POKE 787,0: POKE 788,133:
     POKE 789,1: POKE 790,96
2100 DIM S(16),H$(4),HX$(16),C$(5)
2120 FOR I=2 TO 15:S(I)=16-I: NEXT I
2140 S(1)=0
2160 S(16)=15
2180 HX$="0123456789ABCDEF"
2200 CALL 783: REM               Get address of IOB table from
                                 DOS
2220 AD= PEEK (0)+256*( PEEK (1)-256): REM Calculate IOB
                                 address
2240 SC=AD+5: REM                Sector
2260 B1=AD+9: REM                Buffer address, high byte
2280 B0=AD+8: REM                Buffer address, low byte
2300 TR=AD+4: REM                Track
2320 VL=AD+3: REM                Volume
2340 CM=AD+12: REM               RWTS command
2360 RC=AD+13: REM               Return code,< >0 indicates
                                 error
2380 DR=AD+2: REM                Drive
2400 TEXT : CALL -936:REM        Clear screen, home cursor
2420 VTAB 3
2440 PRINT "THIS PROGRAM READS FROM AND WRITES TO"
2460 PRINT "DISKS IN APPLE III BLOCK ORDER"
2480 PRINT
2500 PRINT "FULL DISK TRACKS ARE TRANSFERRED TO"
2520 PRINT "AND FROM MEMORY SUPERPAGES"
2540 PRINT : PRINT " EXAMPLE PAGE: $2000 - $2FFF": PRINT
2560 PRINT "PAGES $1 TO $7 ARE AVAILABLE"
2580 PRINT
2600 PRINT "ENTER ALL VALUES IN HEXADECIMAL"
2620 PRINT
2640 TAB 10: INPUT "HIT RETURN ",H$
2660 GOTO 700
```

Listing 1. *Trackmover* program to transfer disk tracks to and from memory. Integer Basic.

```
0300  20 E3 03  JSR  $03E3    ;Get IOB address
0303  20 D3 03  JSR  $03D3    ;Call RWTS
0306  B0 06     BCS  $030E    ;On error, return
0308  A0 0D     LDY  #$0D     ;IOB error-byte offset
030A  A9 00     LDA  #$00     ;Zero for no error
030C  91 00     STA  ($00),Y  ;Store in IOB
030E  60        RTS

030F  20 E3 03  JSR  $03E3    ;Get IOB address
0312  84 00     STY  $00      ;Low byte in $00
0314  85 01     STA  $01      ;High byte in $01
0315  60        RTS
```

Listing 2. Assembly listing for the pokes in listing 1 lines 2000 through 2080. It is not necessary to type this in.



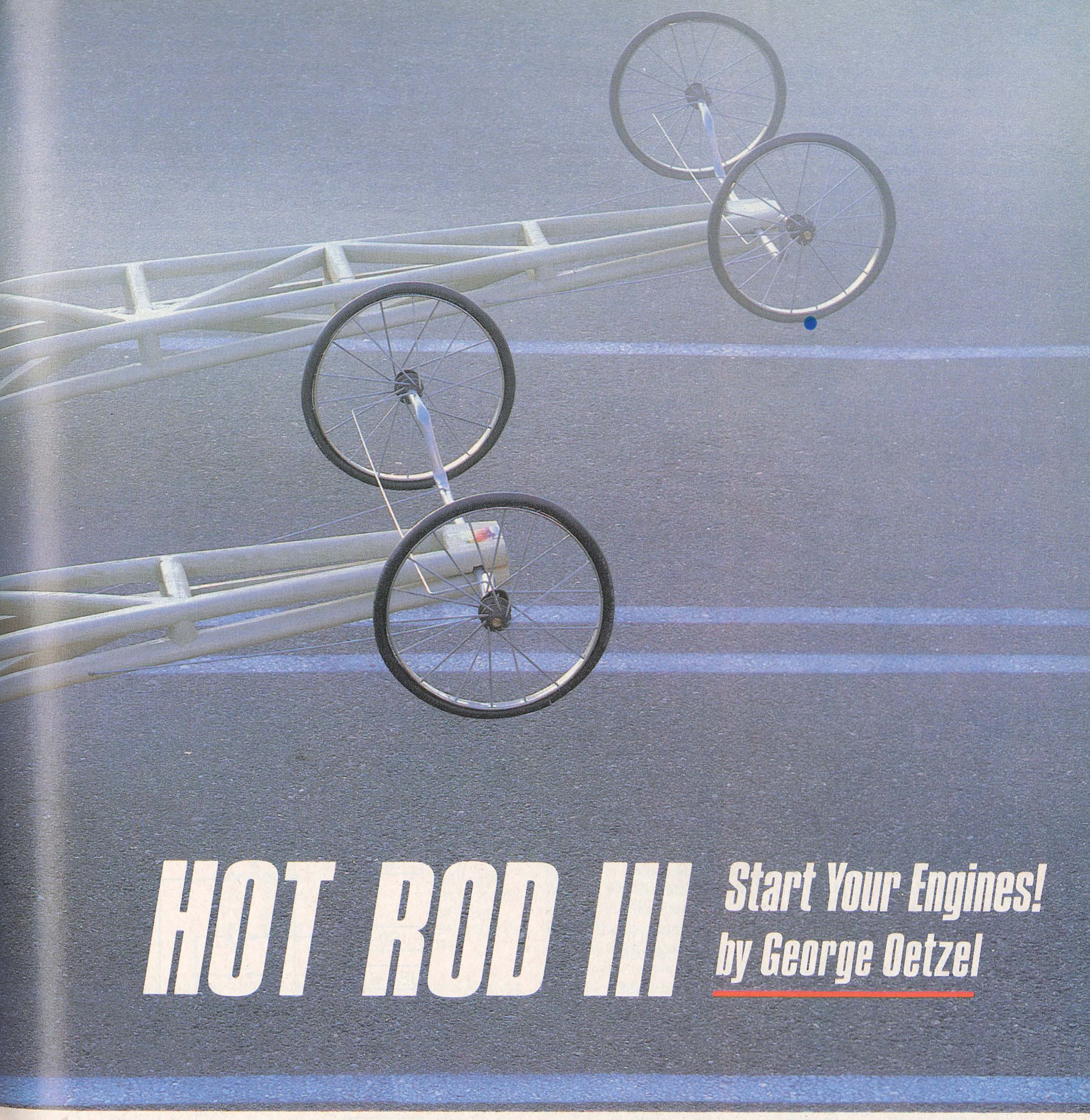
Few of the games written to use the Apple II game paddles will run on the Apple III in Emulation mode. The reason for this unfortunate situation is that the hardware in the Apple III requires very different software to read the paddles. Games that use the Monitor subroutine at $FB1E work on both machines, but many games contain their own routines that work only with the Apple II paddles. This article presents changes in the Emulation Monitor and a technique for altering games so they will work on the Apple III.

The previous article in this series described the organization of the Emulation disk and presented the *Trackmover* utility program to assist with alterations of the Emulation disk. The *Trackmover* or another disk zap utility will be needed to modify the Emulation program and some of the games.

The software solution to the game problem involves substantial modifications of the Apple II Monitor supplied on the Emulation disk. It also requires that you locate and change the routines in the games that read the paddles. This sounds formidable at first, but the tools in this article will allow you to convert a typical game in just a few minutes.

There is a catch, however. You must be able to modify the binary game file on the disk, which means that you can fix only those games that are available as DOS files. Some copy-protected games can be modified. The game file of many recent games is a normal DOS 3.2 or 3.3 file, even though the disk is copy-protected. The game file can be copied to another disk and modified as described here. You can often then run the game if you bload the modified game file, insert the original master disk in the drive, and then start the game with a call to the first address in the game file. It helps if the copy of the game file occupies the same tracks and sectors on the disk as the original, so the disk head is in the same position after the game is loaded from either disk.

In many, perhaps most, cases where the game file is a normal DOS



HOT ROD III

Start Your Engines!

by George Oetzel

file, the protection scheme won't balk if you modify the original disk using a track/sector editor. Of course, attempting to modify an original master that is so well protected that you can't make a backup is a fairly high-stakes computer game. Don't try it unless you are willing to accept some losses.

Here's an outline of the process we'll use to make games run on the Emulation Apple:

1. Revise the Monitor program so that the paddle initialization is done in subroutines.

2. Replace paddle initialization instructions (LDA $C070) in the game with calls to the appropriate subroutine.

3. Replace paddle test instructions, usually LDA $C064 or LDA $C065, with LDA $C066.

In many games only four bytes need to be changed.

Paddle Reading and Monitor Modifications. Let's look at the subroutines that read the paddles. Listing 1 is the paddle-reading subroutine from the Apple II Monitor. Listing 2 is the paddle-reading routine furnished on the Emulation disk. The substantially longer Apple III routine is broken into two pieces to preserve parts of the Monitor that other programs will likely use. There is space for this long routine because the omission of cassette tape routines leaves about one hundred unused bytes in the Monitor. These free bytes allow numerous interesting modifications of the Emulation Apple.

To a calling program, the two subroutines are functionally identical. The paddle number is in the X register when the subroutine is called. The paddle value is in the Y register, with the X register unchanged, on return from the subroutine.

The first, and largest, part of the Apple III routine is devoted to paddle selection. Commands must be given to select one of eight inputs to the analog-to-digital converter (A/D). The selection requires reference to one member of each of three pairs of memory addresses. Figure 1 summarizes the selection rules.

After the paddle has been selected, the operating principle of the Apple III routine is the same as that of the Apple II routine. Why is it so much more complicated? Apple decided that the reference voltage required to yield a paddle output of 255 (or $FF) should be 2.4 volts. That decision is based on joysticks that use 20 percent of the total potentiometer rotation for full joystick deflection. If the joystick is connected to 12 volts, 20 percent of the range yields a maximum output of 2.4 volts. If the Apple II counting routine is used, it requires about 2.55 volts' input to reach the full output value. The loop used in the Apple III remedies this problem by effectively counting only half-range and then doubling the output.

It is easy to fix a paddle or joystick so that its full range is about 2.6 volts, and it will work with the standard counting software. You can build a very nice paddle from scratch for about ten dollars. Directions for modifying the Cursor III joystick are given with figure 2.

Achieving simplicity in game alterations requires the rebuilding of the paddle software in the Monitor. *DOS Tool Kit* assembly listings of the four Monitor patches are given in listings 3 through 6 at the end of this article. Assemble the four routines and use the *Trackmover* program to install them as described in the following paragraphs.

Start with a copy of an Emulation disk that already has the modified reset vector described last month. Otherwise, make that modification along with those described in this article. Load track 5 into memory at address $5000, then exit to Basic and load the patch routines as follows:

```
BLOAD MONFIX1.OBJ0,A$561E
BLOAD MONFIX2.OBJ0,A$57C9
BLOAD MONFIX3.OBJ0,A$59CE
BLOAD MONFIX4.OBJ0,A$59FE
```

```
0000:              ;
0000:              ;
----- NEXT OBJECT FILE NAME IS AP2PADDLE.OBJ0
FB1E:                      ORG    $FB1E
FB1E:AD 70  C0     PREAD   LDA    $C070
FB21:A0 00                 LDY    #$00
FB23:EA                    NOP
FB24:EA                    NOP
FB25:BD 64  C0     PREAD2  LDA    $C064,X
FB28:10 04                 BPL    RTS2D
FB2A:C8                    INY
FB2B:D0 F8                 BNE    PREAD2
FB2D:88                    DEY
FB2E:60            RTS2D   RTS
```

Listing 1. Paddle-reading subroutine from the Apple II Monitor.

```
0000:              ;
0000:              ;
0000:              ;
FCC9:              PART2   EQU    $FCC9
0000:              ;
----- NEXT OBJECT FILE NAME IS AP3PADDLE.OBJ0
FB1E:                      ORG    $FB1E
FB1E:8A                    TXA
FB1F:48                    PHA
FB20:49 01                 EOR    #$01
FB22:AA                    TAX
FB23:AD 59  C0             LDA    $C059
FB26:AD 5E  C0             LDA    $C05E
FB29:AD 5A  C0             LDA    $C05A
FB2C:4C C9  FC             JMP    PART2
----- NEXT OBJECT FILE NAME IS AP3PADDLE.OBJ1
FCC9:                      ORG    $FCC9
FCC9:E8                    INX
FCCA:CA                    DEX
FCCB:F0 12                 BEQ    PDLSET
FCCD:AD 5F  C0             LDA    $C05F
FCD0:CA                    DEX
FCD1:F0 0C                 BEQ    PDLSET
FCD3:AD 58  C0             LDA    $C058
FCD6:CA                    DEX
FCD7:F0 06                 BEQ    PDLSET
FCD9:AD 5E  C0             LDA    $C05E
FCDC:AD 5B  C0             LDA    $C05B
FCDF:AD 5C  C0     PDLSET  LDA    $C05C
FCE2:A9 0F                 LDA    #$0F
FCE4:20 A8  FC             JSR    $FCA8
FCE7:A4 80                 LDY    $80
FCE9:AD 5D  C0             LDA    $C05D
FCEC:A2 48         INIT    LDX    #$48
FCEE:CA                    DEX
FCEF:10 FB                 BPL    INIT
FCF1:E8            PREAD2  INX
FCF2:B9 E6  BF             LDA    $BFE6,Y
FCF5:2A                    ROL    A
FCF6:AD 66  C0             LDA    $C066
FCF9:30 F6                 BMI    PREAD2
FCFB:8A                    TXA
FCFC:10 04                 BPL    MULT2
FCFE:A9 FF                 LDA    #$FF
FD00:D0 01                 BNE    OUTPUT
FD02:2A            MULT2   ROL    A
FD03:A8            OUTPUT  TAY
FD04:68                    PLA
FD05:AA                    TAX
FD06:60                    RTS
```

Listing 2. Paddle-reading subroutine from the Emulation Monitor.

Input selected	C058=0 C059=1	C05A=0 C05B=1	C05E=0 C05F=1
Ground reference	0	0	0
Apple II paddle 3	0	0	1
Apple II paddle 2	0	1	0
Not used	0	1	1
Apple II paddle 1	1	0	0
Apple II paddle 0	1	0	1
Clock battery	1	1	0
2.4 volt reference	1	1	1

Figure 1. Apple III paddle selection requires memory reference commands to one member of each of three pairs of addresses. The figure shows which input is selected for each of the eight possible combinations.

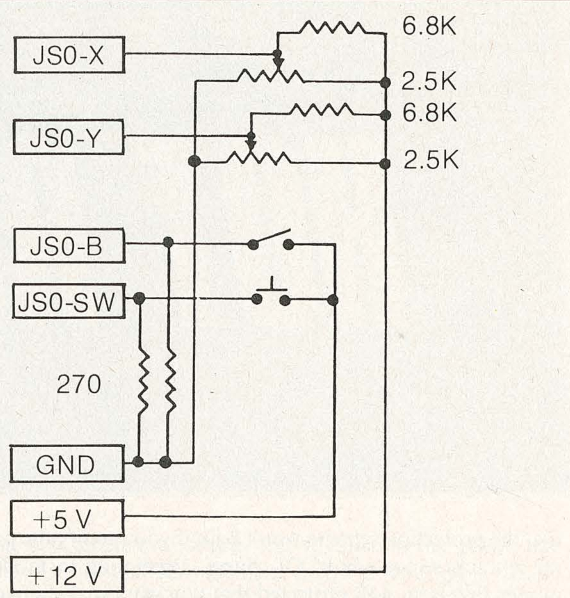


Figure 2. Modifying the Cursor III joystick for Apple II Emulation. The circuit of the Cursor III joystick is identical to that shown on page 130 of the *Owner's Guide*, using 2.5K potentiometers. Adding two 6.8K, 0.25 W resistors as shown will increase the maximum output voltage to about 2.6V for use with software that mimics the Apple II routines.

The Apple II uses a left-handed X-Y coordinate system for its screen display convention; the Apple III is right-handed. As a result no orientation of Cursor III is right for Apple II joystick applications. To fix this, interchange the connections at the two ends of one of the potentiometers. You will then have to recenter the joystick control. Loosen the set screw and take the joystick assembly off of the potentiometer shaft. Rotate the shaft until a simple Applesoft program indicates that the paddle value is about 128. Then reassemble the joystick. Getting the control centered properly with everything assembled and the set screw tight takes some patience.

It will be easier to modify track 9 if you save the entire modified portion of the Monitor as a single file:

```
BSAVE MONMOD,A$5600,L$500
```

Use the *Trackmover* to restore the $5000 address block to track 5 on the disk and then load track 9 into the $5000 block. Now bload Monmod,A$5100 and restore track 9 to the modified Emulation disk.

The Monitor modifications separate paddle selection and the A/D conversion timing so that the paddle selection is performed by subroutines. The assembly instructions to call these subroutines are:

```
paddle 0 select     20  D0  FE          JSR  $FED0
paddle 1 select     20  D7  FE          JSR  $FED7
paddle 2 select     20  17  FF          JSR  $FF17
paddle 3 select     20  0A  FF          JSR  $FF0A
```

There are two multipaddle subroutines. The standard paddle-reading routine at $FB1E accepts a paddle number in the X register and returns the resulting value in the Y register. This subroutine is used by Basic and many games. A comparable, alternative subroutine at $FEFE samples paddle 2 when paddle 1 is requested and vice versa. This allows use of a joystick that is plugged into port A (paddles 0 and 2) without modifying anything except the address of the subroutine call and the paddle 1 firing-button check.

Game Modification. The typical arcade game has a built-in paddle routine that is nearly identical to the one in the Apple II Monitor. Sometimes, though, there are changes designed to limit the numerical range of the output values. Here is an example taken from a game with a nonstandard routine to read paddle 0. The example also shows the changes you must make to fix the game so that it will read the Apple III paddles, using the modified Monitor. The memory contents are shown for the two modified instructions.

```
Address          Original version               Modified version

1800-  AD  70  C0  LDA   $C070  20  D0  FE   JSR  $FED0
1802-  A0  24      LDY   #$24                 LDY  #$24
1804-  EA          NOP                        NOP
1805-  EA          NOP                        NOP
1806-  AD  64  C0  LDA   $C064  AD  66  C0   LDA  $C066
1809-  10  05      BPL   $1810                BPL  $1810
180B-  C8          INY                        INY
180C-  C0  9E      CPY   #$9E                 CPY  #$9E
180E-  90  F6      BCC   $1806                BCC  $1806
1810-  C4  10      CPY   $10                  CPY  $10
1812-  D0  01      BNE   $1815                BNE  $1815
1814-  60          RTS                        RTS
```

Game conversion to run with the revised Monitor requires locating the paddle routines in the game, a little disassembly to be sure that the paddle reading isn't too convoluted, and modification of just a few bytes, as illustrated above. In many cases, the whole procedure can be completed in five minutes or so. There are a few hopeless cases in which a single LDA $C070 instruction initiates an interlaced set of instructions to read two or more paddles.

To fix a game so that it will run on the Apple III, you will have to locate the paddle-reading routines, make changes as previously shown, and save the modified game on the disk. The DOS bsave command balks at files larger than 32,767 bytes (129 sectors on the disk). If the game file is smaller, just bload Game,A$1000 from the Monitor. Then check the length of the file. The low-order byte of the length is in $AA60 after the load; the high-order byte is in $AA61. You will need this information to save the file after you have fixed it.

After the file is in memory, you have to find all the instances of LDA $C070 or BIT $CO70 instructions to find all the places where the game needs to be changed. There are utility programs (such as *The Inspector*, by Omega Microware) that have built-in memory and disk-search features.

If you don't have one of these, the program given in listing 7 will do the job for you. It loads into the page-three area that is universally used for small assembly language routines. After it is loaded, type *300G* from the Monitor to link it to the Monitor control-Y instruction. Memory address $00 contains the number of bytes in the pattern to be located, and the pattern is loaded into memory beginning at $01. Limit patterns to nine bytes or less to avoid destroying important zero-page locations. On a practical basis, a three-byte pattern almost always yields just a few locations. To find all of the LDA $C070 instructions in the memory range from $1000 to $90FF, enter this from within the Monitor.

```
0:3 AD 70 C0
1000.9000 control-Y
```

The address of the $AD byte for each paddle routine in the address range will be printed on-screen after you type return. Disassemble the paddle routine in each location to be sure what is going on. An LDA $C064 instruction, as shown in the example just given, indicates paddle 0. Similarly, LDA $C065 indicates paddle 1. Replace the LDA $C070 instruction with a JSR instruction to initialize the appropriate paddle. Also, replace LDA $C06*x* with LDA $C066.

If you have a joystick with connections only to a single joystick port, then you will want to use paddle 0 and paddle 2 where the normal Apple II organization uses paddles 0 and 1. If the two paddle routines are independent, then use JSR $FF17 in place of the LDA $C070 that starts the paddle 1 read in the game. You will also have to change the firing-button commands to use buttons 0 and 2. Button 1 is read by LDA $C062; change it to LDA $C063 for the paddle 2 button.

Many games using joysticks employ the Monitor routine at $FB1E to read them. These games run without modification if you can connect your joystick to the normal paddle 0 and 1 combination. The Monitor patches include an option that interchanges the logical paddle 1 and paddle 2 assignments. Change JSR $FB1E in the game to JSR $FEFE to perform the swap. The normal subroutine, at $FB1E, could be changed to the swapped configuration quite easily, but this isn't a universal solution. Game conversion often would still require changing the firing-button commands.

Because of the file-length limitation with the DOS bsave command, extremely long game files must be searched in pieces to find all of the paddle routines. The *Trackmover* program presented in part 1 of this series can be used to assist in this search. Initialize a DOS disk with a minimum hello file and then transfer the game program to it, using Apple's FID or another disk utility. DOS stores programs on a newly initialized disk in consecutive tracks and sectors. You can expect to find the hello program on track $12 and the first track/sector list for your game on track $13, sector $F. The track allocation continues with tracks $3 through $A. In spite of the fact that DOS is quite consistent, it's a good idea to find and check the track/sector lists to determine exactly what part of the disk contains your program. The catalog entry, with a pointer to the track/sector list, is on track $11, sector $F, if the game is the second program recorded after initializing the disk. Both the DOS 3.3 manual and *Beneath Apple DOS*, by Don Worth and Pieter Lechner, provide guides to the interpretation of catalog entries and track/sector lists.

Use the *Trackmover* program to load seven program tracks into the memory range $1000–$7FFF. Each track will be loaded in its proper memory order if you change two program lines in *Trackmover*.

```
2140 S(1) = 15
2160 S(16) = 0
```

Use the pattern-location program to find the paddle-control routines. Modify the paddle control with the Monitor or the Miniassembler, and then use the *Trackmover* to save the modified portions back to the same place on the disk. Continue with the remaining tracks that contain portions of the game program until you are sure that you have located and modified all of the paddle-control routines. Few games have more than two paddle control routines.

Next month, we'll go into the workings of the Emulation program itself, including the instructions that set up the Emulation mode. Study of the details of the Emulation program makes possible useful modifications, such as the reading and display of lower-case characters. It also makes possible more exotic Emulation Apples, if the machine-control register setup is different from that furnished on the Emulation disk. Meanwhile, try modifying some games to work on the Apple III. You'll find that the conversion is an enjoyable and rewarding challenge.

Listing 3. Emulation Monitor patch 1 for use with Apple II games.

```
0000:          ;
0000:          ;
0000:          ; ************************************
0000:          ;
0000:          ;    APPLE III EMULATION MODE
0000:          ;    PADDLE-SERVICE ROUTINES
0000:          ;
0000:          ;    MONITOR PATCH ROUTINE #1
0000:          ;    NORMAL PADDLE ENTRY
0000:          ;
0000:          ;    PADDLE NUMBER IN X
0000:          ;    VALUE READ RETURNS IN Y
0000:          ;
0000:          ;
0000:          ;
0000:          ; ************************************
0000:          ;
FCC9:            NORMSET   EQU   $FCC9
0000:          ;
----- NEXT OBJECT FILE NAME IS MONFIX1.OBJ0
FB1E:                      ORG   $FB1E
FB1E:20 C9 FC              JSR   NORMSET
FB21:A0 00                 LDY   #$00
FB23:EA                    NOP
FB24:EA                    NOP
FB25:AD 66 C0    PREAD2    LDA   $C066
FB28:10 04                 BPL   RTS2D
FB2A:C8                    INY
FB2B:D0 F8                 BNE   PREAD2
FB2D:88                    DEY
FB2E:60          RTS2D     RTS
```

Listing 4. Emulation Monitor patch 2.

```
0000:          ; ************************************
0000:          ;
0000:          ;    APPLE III EMULATION MODE
0000:          ;    PADDLE-SERVICE ROUTINES
0000:          ;
0000:          ;    MONITOR PATCH ROUTINE #2
0000:          ;    NORMAL PADDLE SETUP SEQUENCE
0000:          ;
0000:          ;
0000:          ;
0000:          ; ************************************
0000:          ;
FECE:            PDL0X     EQU   $FECE
FED5:            PDL1X     EQU   $FED5
FF15:            PDL2X     EQU   $FF15
FF08:            PDL3X     EQU   $FF08
0000:          ;
----- NEXT OBJECT FILE NAME IS MONFIX2.OBJ0
FCC9:                      ORG   $FCC9
FCC9:8A          NORMSET   TXA
FCCA:48                    PHA
FCCB:F0 0F                 BEQ   JMP0X
FCCD:CA                    DEX
FCCE:F0 09                 BEQ   JMP1X
FCD0:CA                    DEX
FCD1:F0 03                 BEQ   JMP2X
FCD3:4C 08 FF              JMP   PDL3X
FCD6:4C 15 FF    JMP2X     JMP   PDL2X
FCD9:4C D5 FE    JMP1X     JMP   PDL1X
FCDC:4C CE FE    JMP0X     JMP   PDL0X
```

Listing 5. Emulation Monitor patch 3.

```
0000:          ;
0000:          ; ************************************
0000:          ;
0000:          ;    APPLE III EMULATION MODE
0000:          ;    PADDLE-SERVICE ROUTINES
0000:          ;
0000:          ;    MONITOR PATCH ROUTINE #3
0000:          ;    PADDLE 0 & 1 INITIALIZATION
0000:          ;
0000:          ;
0000:          ;
0000:          ; ************************************
0000:          ;
FCA8:            WAIT      EQU   $FCA8
0000:          ;
----- NEXT OBJECT FILE NAME IS MONFIX3.OBJ0
FECE:                      ORG   $FECE
FECE:68          PDL0X     PLA
FECF:AA                    TAX
FED0:AD 5F C0    PDL0      LDA   $C05F
FED3:D0 05                 BNE   PX
FED5:68          PDL1X     PLA
FED6:AA                    TAX
FED7:AD 5E C0    PDL1      LDA   $C05E
FEDA:AD 59 C0    PX        LDA   $C059
FEDD:AD 5A C0              LDA   $C05A
FEE0:          ;
FEE0:          ;    PADDLE SELECT IS COMPLETE AT
                    THIS POINT
FEE0:          ;    FOLLOWING STATEMENTS INITIATE
                    THE A/D
FEE0:          ;    NOTE THAT X AND Y ARE
                    UNCHANGED
FEE0:          ;    FROM HERE THRU THE RTS
FEE0:          ;
FEE0:AD 5C C0    GO        LDA   $C05C
FEE3:A9 0F                 LDA   #$0F
FEE5:20 A8 FC              JSR   WAIT
FEE8:AD 5D C0              LDA   $C05D
FEEB:38                    SEC
FEEC:A9 0E                 LDA   #$0E
FEEE:E9 01       WAIT4     SBC   #$01
FEF0:D0 FC                 BNE   WAIT4
FEF2:60                    RTS
```

Listing 6. Emulation Monitor patch 4.

```
0000:          ;
0000:          ; ************************************
0000:          ;
0000:          ;    APPLE III EMULATION MODE
0000:          ;    PADDLE-SERVICE ROUTINES
0000:          ;
0000:          ;    MONITOR PATCH ROUTINE #4
0000:          ;    PADDLE INITIALIZATION FOR
0000:          ;    SWAPPED PADDLE 1 AND 2
0000:          ;
0000:          ;
0000:          ;
0000:          ; ************************************
0000:          ;
FECE:            PDL0X     EQU   $FECE
FED5:            PDL1X     EQU   $FED5
FEE0:            GO        EQU   $FEE0
FB25:            PREAD2    EQU   $FB25
FEFE:            SWAPSET   EQU   $FEFE
0000:          ;
----- NEXT OBJECT FILE NAME IS MONFIX4.OBJ0
FEFE:                      ORG   $FEFE
FEFE:8A          SWAPPED   TXA
FEFF:48                    PHA
FF00:F0 CC                 BEQ   PDL0X
FF02:CA                    DEX
FF03:F0 10                 BEQ   PDL2X
FF05:CA                    DEX
FF06:F0 CD                 BEQ   PDL1X
FF08:68          PDL3X     PLA
FF09:AA                    TAX
FF0A:AD 58 C0    PDL3      LDA   $C058
FF0D:AD 5A C0              LDA   $C05A
FF10:AD 5F C0              LDA   $C05F
FF13:D0 CB                 BNE   GO
FF15:68          PDL2X     PLA
FF16:AA                    TAX
FF17:AD 58 C0    PDL2      LDA   $C058
FF1A:AD 5B C0              LDA   $C05B
FF1D:AD 5E C0              LDA   $C05E
FF20:D0 BE                 BNE   GO
FF22:20 FE FE    SPREAD    JSR   SWAPSET
FF25:4C 25 FB              JMP   PREAD2
FF28:00                    BRK
```

Listing 7. Program to locate byte patterns in Apple II memory.

```
0000:              ; ****************************************
0000:              ;
0000:              ;        BYTE PATTERN LOCATOR
0000:              ;  ENTER NUMBER OF BYTES IN PATTERN
0000:              ;  INTO LOCATION $00. ENTER PATTERN
0000:              ;  BEGINNING IN LOCATION $01.
0000:              ;
0000:              ;  INITIALIZE MONITOR HOOK BY 300G.
0000:              ;  SEARCH ADDRESS RANGE USING
                      XXXX.YYYY CTRL-Y RETURN
0000:              ;  PATTERN START ADDRESSES WILL BE
                      PRINTED ON-SCREEN
0000:              ;
0000:              ;
0000:              ;
0000:              ;
0000:              ; ****************************************
0000:              ;
0000:              NUMBYT    EQU   $00
0001:              PATTERN   EQU   $01
003C:              A1        EQU   $3C
003D:              A1H       EQU   $3D
003E:              A2        EQU   $3E
003F:              A2H       EQU   $3F
0040:              A3        EQU   $40
0041:              A3H       EQU   $41
03F8:              VECTOR    EQU   $3F8
FDDA:              PRHEX     EQU   $FDDA
FDED:              COUT      EQU   $FDED
0000:              ;
----- NEXT OBJECT FILE NAME IS PATTERN.300.OBJO
0300:                        ORG   $300
0300:              ;
0300:              ;  PUT JUMP INSTRUCTION TO PROGRAM
0300:              ;  START INTO $3F8 FOR CTRL-Y ENTRY
0300:              ;
0300:A9 4C                   LDA   #$4C
0302:8D F8 03                STA   VECTOR
0305:A9 10                   LDA   #>START
0307:8D F9 03                STA   VECTOR+1
030A:A9 03                   LDA   #<START
030C:8D FA 03                STA   VECTOR+2
030F:60                      RTS
0310:              ;
0310:              ; START PATTERN FINDER
0310:              ;
0310:A9 00         START     LDA   #$00
0312:A4 3C                   LDY   A1
0314:85 3C                   STA   A1
0316:20 32 03                JSR   SRCHP1
0319:18                      CLC
031A:90 03                   BCC   INCX
031C:              ;
031C:              ;  MAIN LOOP TO SEARCH MEMORY
031C:              ;
031C:20 30 03      LOOP      JSR   SRCHPG
031F:E6 3D         INCX      INC   A1H
0321:A5 3D                   LDA   A1H
0323:F0 0A                   BEQ   RTS1
0325:C9 C0                   CMP   #$C0
0327:F0 F6                   BEQ   INCX
0329:C5 3F                   CMP   A2H
032B:90 EF                   BCC   LOOP
032D:F0 ED                   BEQ   LOOP
032F:60            RTS1      RTS
0330:              ;
0330:              ;  SUBROUTINE TO SEARCH ONE
                      MEMORY PAGE
0330:              ;  FOR DESIRED PATTERN
0330:              ;
0330:A0 00         SRCHPG    LDY   #$00
0332:A5 01         SRCHP1    LDA   PATTERN
0334:D1 3C         SRCLOOP   CMP   (A1),Y
0336:F0 04                   BEQ   EQUAL
0338:C8                      INY
0339:D0 F9                   BNE   SRCLOOP
033B:60                      RTS
033C:84 40         EQUAL     STY   A3
033E:A5 3D                   LDA   A1H
0340:85 41                   STA   A3H
0342:A2 01                   LDX   #$01
0344:E8            NEXTBYT   INX
0345:8A                      TXA
0346:C5 00                   CMP   NUMBYT
0348:F0 02                   BEQ   TEST
034A:B0 11                   BCS   PRADR
034C:B5 00         TEST      LDA   NUMBYT,X
034E:C8                      INY
034F:F0 06                   BEQ   NEXTPG
0351:D1 3C         COMPARE   CMP   (A1),Y
0353:F0 EF                   BEQ   NEXTBYT
0355:D0 DB                   BNE   SRCHP1
0357:E6 3D         NEXTPG    INC   A1H
0359:D0 F6                   BNE   COMPARE
035B:F0 21                   BEQ   ENDER
035D:A5 41         PRADR     LDA   A3H
035F:20 DA FD                JSR   PRHEX
0362:A5 40                   LDA   A3
0364:20 DA FD                JSR   PRHEX
0367:A9 A0                   LDA   #$A0
0369:20 ED FD                JSR   COUT
036C:18                      CLC
036D:A5 40                   LDA   A3
036F:65 00                   ADC   NUMBYT
0371:A8                      TAY
0372:90 BE                   BCC   SRCHP1
0374:E6 41                   INC   A3H
0376:F0 06                   BEQ   ENDER
0378:A5 41                   LDA   A3H
037A:85 3D                   STA   A1H
037C:D0 B4                   BNE   SRCHP1
037E:68            ENDER     PLA
037F:68                      PLA
0380:60                      RTS
```

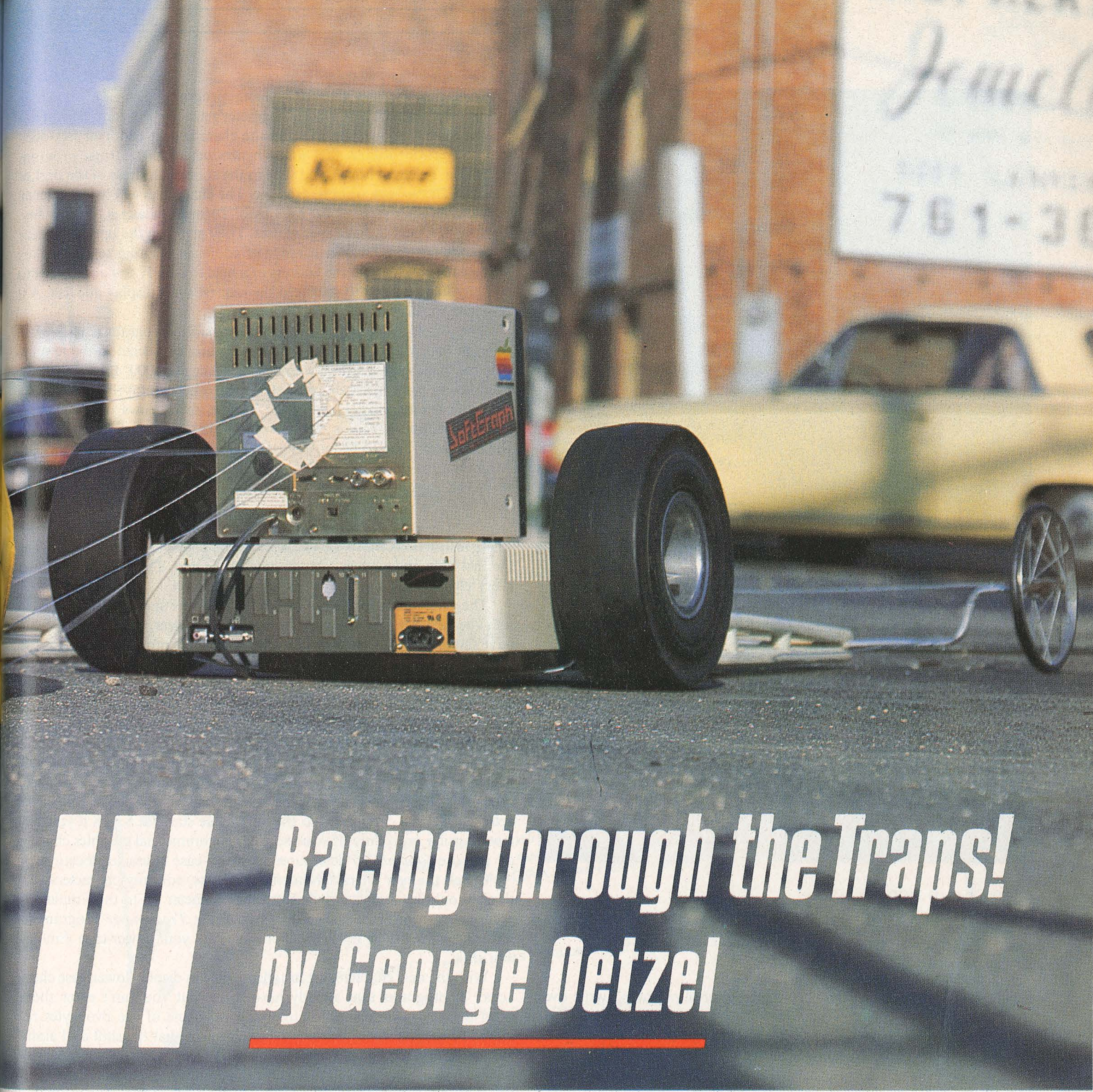
HOT ROD

The two previous articles in this series have described differences between the Apple II and Apple III hardware that affect the Emulation mode, the organization of the Basic and Monitor images on the Emulation disk, and changes in the Monitor and games programs that make it possible to play many Apple II games on the Apple III. Now we'll delve more deeply into the Emulation program itself to discover further variations of the Emulation Apple. The first, with lower-case character display and keyboard entry, permits the use of programs that use lower-case display. This includes programs for the Apple IIe that don't require the eighty-column card. More exotic configurations include 60K of RAM and the use of Apple II software in an Apple III hardware environment, with delightful and novel results.

When the Emulation program boots, the first twelve disk blocks are loaded in sequence to install the program into the address range $A000 through $B677. (Block 0 occupies $A000 through $A1FF; block 1 fills $A200 through $A3FF, and so forth.) The program is an interesting application of the "memory is cheap, but code is expensive" approach. Code segments are in $A000 through $A3C4 and $A4C7 through $A67D. The rest consists mostly of the two text screen images, complete with all the spaces.

First, let's install a new character set and keyboard handler so we can use programs written for an Apple with lower-case display, including some new programs written for the Apple IIe.

Installation and use of a character set with lower case involves three types of changes to the Emulation mode. The first is the installation of the character set, which is a modification of the Emulation program. The second is the modification of the KEYIN routine in the Apple II Monitor so that it will read and interpret both bytes from the Apple III keyboard. The third is the treatment of inverse and flashing modes in a consistent way. This involves some decisions about what type of consistency



Racing through the Traps!

by George Oetzel

you want and how much work you are willing to do to achieve it.

The problem with inverse and flashing characters arises because there are only 256 character codes recognized by the character generator. The Apple II normally displays sixty-four characters, which are translated into sixty-four character codes for inverse, sixty-four for flashing, and two normal sets. Adding lower case expands the character set to ninety-six characters. You can't have full sets of inverse, flashing, and normal characters, because there are too few character codes. Something has to give. You can settle for inverse or flashing display of the wrong character part of the time, but the most satisfactory solution is to eliminate the flashing mode entirely.

A simple program will display the entire character set on the screen. It is written in Integer Basic, as are the other Basic programs in these articles. Integer Basic is useful for applications that involve modifications of binary files, because it includes the Miniassembler and can include other utilities in the $D800 through $DFFF address space.

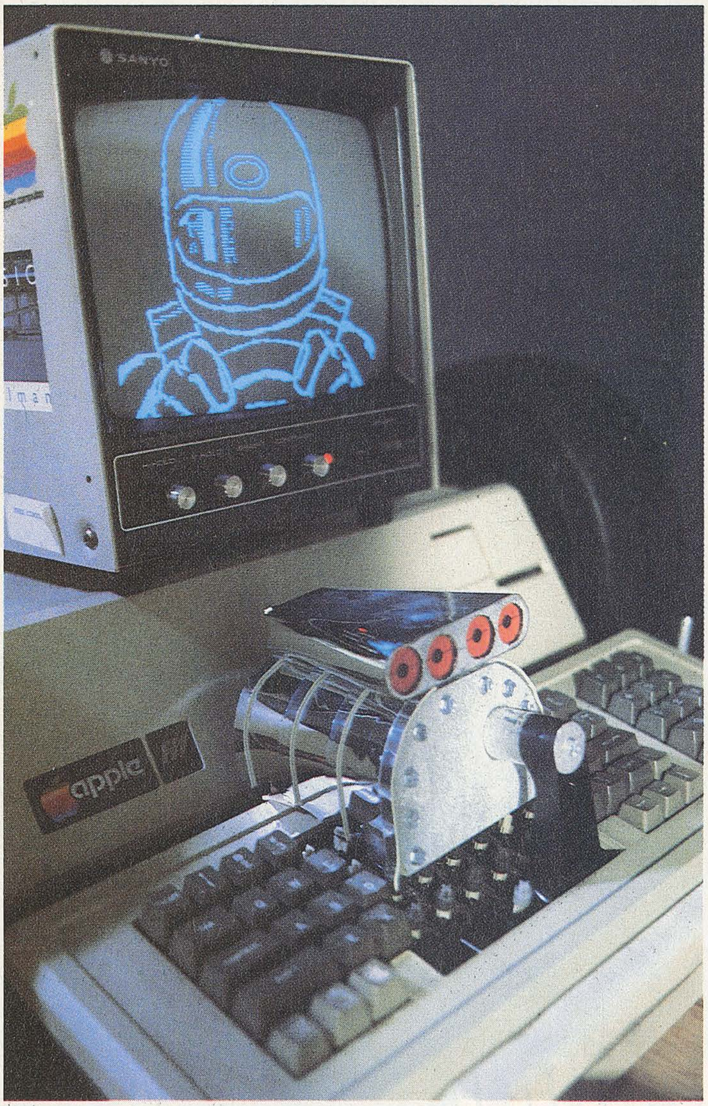
First, clear the screen:

```
100 CALL -936
```

Next, set up a loop to poke eight rows, each containing thirty-two character codes, directly to the screen memory locations.

```
120 FOR R=0 TO 7
140 I=R+1+R/2
160 FOR X=0 TO 31
180 C=32*R+X
```

The variable C is the character code. The character displayed does not have to be the one suggested by the corresponding ASCII value, as is clear from the absence of lower-case letters in the Apple II character set. Next, calculate the screen addresses and poke in the characters.



```
200 J=(I-1) MOD 8
220 K=(I-1)/8
240 POKE 1024+128*J+40*K+X,C
260 NEXT X
280 NEXT R
```

Finally, move the cursor down so the Integer prompt doesn't write over the display.

```
300 VTAB 12 : PRINT
320 END
```

The Apple III character generator uses 128 character images to generate the 256 displayed characters. Each character image does double duty, once in the normal character set, with its character code larger than 128, and again in the inverse set, with the character code 128 less than the corresponding normal code. On the distributed Emulation disk, the images in one sixty-four-character set have the most significant bit of every byte set, so that the characters will flash in the inverse mode. (See the *Standard Device Drivers Manual*, page 166.) When the lower-case letters are added, they must appear in either inverse or flashing format for character codes 97 through 127 and in normal display for character codes 225 through 255. The character ROM in the Apple II has 256 characters, giving more freedom in character-set design.

The easiest way to make a new character set is to start with one that is already nearly what you want. Several commercial graphics packages for the Apple include full character sets in the proper format. Two examples are the Apple *DOS Tool Kit* and the Penguin *Complete Graphics System.*

You can use the character-editing program, *Charedit*, shown in listing 1, to examine and modify character sets. The program has an initial menu that allows you to select the character-set editor or hex-decimal conversion utilities that are a byproduct of the normal program operation. Select option 1, and type in the starting address of the part of the character set you want to examine. The characters will begin to scroll by, with an address label for each byte. To make changes, press a key. Two keys, C and S, have special uses. They give you the option to clear or set the flashing bit over a range of addresses. Hitting any other key yields a request to enter the address of a byte to edit. If you don't want to edit anything, enter 0 to return to the request for a display address. A 0 response to this request ends the program.

The Apple II character set in the Emulation program is in $AA86 through $AE85. Use the *Trackmover* program, presented in the first article, to load track 0 from an Emulation disk at $4000. Each character occupies an eight-byte cell, one byte for each horizontal row of dots. As furnished, the characters sit in the bottom seven rows of the display space, and the top row is blank. To provide for lower-case descenders, the characters have to be moved to the top of their display windows. The Monitor provides a quick fix:

```
5000<4A87.4E85M
53FF:0
```

These instructions move the character set to $5000 for editing, removing the first byte and adding a blank byte at the end. The effect is to move every character to the top of its display window.

The *Charedit* program can be used to examine and edit the character set. The first $1FF bytes have the flashing bit clear, and the next $200 have the flashing bit set. Providing for descenders moved the tops of the upper-case characters to hex addresses that end in 0 or 8 and displaced the alignment of the flashing bits from the character cells by one byte. Use the C option to clear the flashing bits in $51FF through $53FF. The character cell that will become the letter "a" begins at $5308, corresponding with ASCII code 97.

The *DOS Tool Kit* character set has ninety-six characters. If you load it at $3000, the "a" is at $3208. Use the Monitor move command, 5308<3208.32FFM, to add it to the new character set. The Penguin character set has 128 characters, so the lower case starts at $3308 if you follow a similar procedure. If you use another commercial graphics character set, use *Charedit* to determine the lower-case character location.

Check the new characters for flashing bits, edit any characters you think should be changed, and return the character set to the Emulation program with 4A86 < 5000.53FFM. Use the *Trackmover* program to restore the modified Emulation program to your lower-case Emulation disk.

With the new character set installed, you can display lower-case characters in programs that already have them, but you can't enter them from the keyboard. The Monitor reads only one of the two bytes required to decode the keyboard. It doesn't know that the shift and alpha lock keys exist.

A revised Monitor KEYIN subroutine and a few minor patches in the character handling elsewhere will remedy the problem. The KEYIN3 subroutine is shown in listing 2. To install it, load track 5 from your lower-case Emulation disk at address $5000 and track 9 at address $4000. Bload KEYIN3,A$59FE ($FEFE when the Monitor is loaded in its proper location). Four small changes will complete the Monitor modifications.

The RDKEY subroutine, at $FD0C, normally converts characters to flashing mode when the cursor backs over them. If it is unchanged, the flashing versions of the lower-case characters become a confusing assortment of inverse numbers and punctuation marks, and the numbers become inverse lower-case letters. All characters will be converted to proper inverses if three bytes are altered:

	Original		Modified	
FD11:	29 3F	AND #$3F	29 7F	AND #$7F
FD13:	09 40	ORA #$40	EA	NOP
			EA	NOP

Modify the track 5 memory image from the Monitor with 5812:7F EA EA.

In the Monitor KEYIN routine, replace the instruction to read the keyboard with a jump to the new KEYIN3 routine. The Monitor command 5828:4C FE FE will replace the LDA $C000 (AD 00 C0) with JMP $FEFE. All character input to the Monitor is converted to upper case by an AND #$DF (29 DF) instruction at $FD82. The Monitor won't interpret lower case, and it isn't likely that this part of the Monitor is used by other programs. The conversion can be eliminated with 5882:EA EA if you want to be able to enter lower case in the Monitor anyway.

A routine to store $3F in the inverse flag location at $32 should be changed to store $7F instead. Enter 5982:7F to modify the track 5 image. This completes the modification of the Monitor image used with Integer Basic. Copy the modified Monitor to the Applesoft version with 4300 < 5800.5AFFM. Then write the modified tracks 5 (from $5000) and 9 (from $4000) back on the Emulation disk.

With the character set installed in the Emulation program and the Monitor patched for lower-case entry, one further change is needed on the Emulation disk. The Applesoft inverse and flash commands should also be made to produce readable output. The *Pattern Location* program, presented in the second article as a tool for locating paddle routines in games, serves equally well to analyze the inverse and flash commands in Applesoft. Since we know that the inverse flag is stored in $32, one of three commands must put the value there: STY $32 (84 32), STA $32 (85 32), or STX $32 (86 32). Without much difficulty, we can locate the following code:

```
F273-   A9 FF     LDA #$FF
F275-   D0 02     BNE F279
F277-   A9 3F     LDA #$3F
F279-   A2 00     LDX #$00
F27B-   85 32     STA $32
F27D-   86 F3     STX $F3
F27F-   60        RTS
F280-   A9 7F     LDA #$7F
F282-   A2 40     LDX #$40
F284-   D0 F5     BNE $F27B
```

It looks like everything needed is right here. Changing $F278 from $3F to $7F should fix the inverse problem. Further, it appears that address $F3 must be used for the flashing mode. We could store a zero in $F3 all the time, but this would waste a valuable page 0 location. A better solution is to eliminate its use in the output routine. Looking for ORA $F3 (05 F3) quickly shows the route:

```
DB62-   05 F3     ORA $F3
DB64-   20 ED FD  JSR $FDED
```

Replace the ORA instruction with a pair of NOP instructions and the flashing mode disappears. The flashing and inverse modes both yield inverses of the desired characters.

These changes to Applesoft must be made on two different disk tracks. Load tracks 7 and 8 in addresses $4000 through $5FFF. Go to the Monitor and type 5878:7F and then 587D:EA EA to make the first change. Type 4162:EA EA to eliminate the ORA instruction, and write both tracks back on the Emulation disk. With these changes, the Applesoft inverse and flash commands both produce the inverse of the desired character. Since Integer Basic doesn't have these commands, no more changes are required. The lower-case Emulation disk is complete.

With these changes, the control-character codes, 0 through 31 and 128 through 159, have no role in normal character displays. You can create your own special characters to go with these codes. You can use the inverse flag, *poke 50,31*, to print the inverse control characters, but you will need your own routine to place the normal ones on the screen.

Many Apple II editors, both for Basic (such as *PLE* or *GPLE*) and in editor/assembler packages, generate the flashing cursor internally and will show inverse lower case when the cursor goes over numbers. If you use one frequently, you will probably want to change the cursor routine to match the changes made on the Emulation disk. It is usually easy to find the responsible routine. Just search for the ORA #$40 (09 40) command that sets the flashing mode in the normal character set. Most of these programs use the Monitor KEYIN subroutine to read the keyboard, so they will automatically accept lower-case characters after you have modified the Emulation disk.

Exotic Emulation. Let's look at the hardware setup that accompanies the Emulation mode and see what changes might produce useful results. Here is the instruction sequence that turns on the Emulation mode:

```
              ;$FFD0 is the zero-page control register.
A56F- A9 00     LDA #$00      ;Select zero page = 0.
A571- 8D D0 FF  STA $FFD0

              ;$FFDF is the environment control register.

A574- A9 FC     LDA #$FC      ;Select environment—
A576- 8D DF FF  STA $FFDF     ;discussed below.

              ;$FFEF selects memory bank and I/O status.

A579- AD EF FF  LDA $FFEF     ;Retain same
A57C- 8D EF FF  STA $FFEF     ;memory bank.

              ;$FFE3 is the data direction register for the
              ;A port of the E VIA. Set the
              ;Emulation bit to output status, so the
              ;STA $FFEF will turn on Emulation.

A57F- AD E3 FF  LDA $FFE3     ;Set the Emulation mode
A582- 09 40     ORA #$40      ;bit in data direction
A584- 8D E3 FF  STA $FFE3     ;control register.
A587- AD EF FF  LDA $FFEF     ;Reselect memory
A58A- 29 B0     AND #$B0      ;bank 0, and turn on
A58C- 8D EF FF  STA $FFEF     ;Emulation mode.
```

Table 1 describes the uses of the bits in the environment register ($FFDF). It originally appeared in "III Bits: John Jeppson's Guided Tour of Highway III" (May 1983 *Softalk*). That article is an excellent reference concerning the Apple III hardware features.

Value	Bit	Function	Bit=0	Bit=1
01	0	$F000–$FFFF	RAM	ROM
02	1	ROM#	ROM#2	ROM#1
04	2	stack	alternate	normal
08	3	$C000–$FFFF	read/write	read only
10	4	RESET key	disabled	enabled
20	5	video	disabled	enabled
40	6	$C000–$CFFF	RAM	I/O
80	7	clock speed	2 MHz	1 MHz

Table 1. Environment register ($FFDF) description.

When the environment is set to $FC, the clock speed is 1 MHz; the I/O, video, and reset key are all enabled; the memory in $C000 through $FFFF is write-protected to behave as if it were ROM; the true ($0100) stack is used, and the ROM is deselected. Two changes look tempting immediately. One is to install RAM in the $C000 through $FFFF memory space, and the other is to run the "Apple II" at 2 MHz. Both are feasible, with the important limitation that most Apple III I/O requires the 1 MHz clock, so we can't set the Emulation mode clock permanently at 2 MHz.

The installation of RAM in high memory allows the use of a limited selection of language-card software. There are two restrictions:

1. There is no bank selection in the $D000 through $DFFF memory range, so programs that use the extra memory bank can't be used.
2. Programs that switch back and forth between the language card and ROM memories won't work. This includes the language-card DOS, for example.

To use the all-RAM Emulation mode, you must disable a nasty function in DOS 3.3. During the boot process, DOS stores a zero in $E000. The zero is supposed to be in the language card, but the language-card control instructions don't do anything in the Apple III. In normal Emulation mode, the memory is write-protected, so the store instruction has no effect.

A disk that already contains DOS can be fixed with the aid of the *Trackmover* program. Read track 0 into memory at address $5000. The Monitor command 56D3:EA EA EA replaces the unwanted instruction with three NOPs. Rewrite track 0 on any DOS 3.3 disk that you want to use with RAM in high memory. Disks initialized by the modified DOS will have the same DOS changes. Since the language-card initialization is useless in Emulation mode, it can be eliminated from any disk you expect to use exclusively on the Apple III.

After you have a disk that will boot DOS without destroying Basic, a one-byte change of the Emulation program will select Emulation mode without write-protecting the high memory. Read track 0 of the Emulation disk into memory at $4000. Use the Monitor command 4575:F4 to change the value in the environment register from $FC to $F4 and rewrite track 0 on the disk. The change eliminates the high-memory write protection without changing any other part of the Emulation setup.

As a finale, we consider a Hybrid mode, in which Apple II software has full control of the Apple III environment. Advantages include number crunching, access to the 6502 and system clocks, experimentation with Apple III hardware, and some novelty features. Applesoft at 2 MHz is faster than Business Basic, because there is no memory management overhead. The system clock normally cannot be read in Emulation mode, because the zero-page register switches the output bytes. Experimentation with routines to control Apple III hardware is easier in the simpler Apple II software environment than under SOS. The color text display mode is a delightful novelty. It would be welcome on the real Apple II.

There are some important limitations to the Hybrid mode, too. It's not an environment for big software projects or for most commercial programs. DOS 3.3 won't select drive 2 in the Hybrid mode, and only three of the Apple III video modes are usable with Apple II software.

A two-byte change in the Emulation program eliminates the Apple II switch to begin the production of a Hybrid Emulation disk. Use an Emulation disk already configured for lower case, because we will use base page address $F3 for color text utilities. Read track 0 into memory at $4000. The Monitor command 4582:EA EA replaces the ORA #$40 instruction with two NOP instructions. Restore the modified track 0 to the Emulation disk. If you boot an Apple II disk at this stage, the screen fills with a checkerboard of miscellaneous colors (or shades of gray). The Monitor doesn't work at all. We have to chase down some problems to build a working computer.

The checkerboard screen display occurs because the Apple II Monitor INIT routine ($FB2F) sets the display switches for the Apple III color text mode. In this mode, each character in text page one ($400 through $7FF) is affected by the contents of the corresponding address in text page two ($800 through $BFF). The most significant nibble determines the character color, and the least significant nibble determines the background color. On a monochrome monitor, $F0 produces the normal light-on-dark display, and $0F yields an inverse display. On a color monitor, the colors are those listed on page 41 of the *Standard Device Drivers Manual*.

The Monitor program doesn't work because it expects to find data tables in $FFD0 through $FFEF, where there are Apple III control registers. We will move the data tables into parts of the Monitor that can't be used in the Hybrid mode. Listing 3 illustrates subroutines that make the color text mode a usable addition to Applesoft programs, read the system clock, and control the 6502 clock and video output. All are shown on memory page three, but they could be installed permanently in portions of Applesoft and the Monitor used by the cassette load and save commands and the lo-res graphics commands.

Initializing the color text mode requires filling text page two with $F0 bytes to give a normal display. This usually destroys the first $400 bytes of an Applesoft program, so the initializing routine beginning at $300 relocates the start of Basic programs to $C00 and calls the Applesoft new program instruction. Because of this, the color-text initialization, call 768, must be the last instruction in any Basic program in which it is used.

To restore the screen to a normal display without destroying Basic programs, call 787. The color-text routines use the base page address that we eliminated from the Applesoft flashing mode ($F3) to store the value used to fill text page two. Using C for the character color and B for the background, each with a range 0 to 15, VAL = B + 16*C. Then poke 243,VAL : call 791 to color the screen and characters to your liking. To fill just one line poke 243,VAL: poke 64,line : call 807. This feature has real utility. If VAL = 0, the line disappears from view, but the characters are still in screen memory. Set VAL=240 and the characters reappear in normal display. For inverse, use VAL=15. The Emulation program uses this technique to change menu fields from normal to inverse and to erase and restore lines of text.

Only three of the Apple III display modes can be used easily with Apple II software. Turn on full-screen, hi-res, black-and-white graphics with LDA $C057 or peek(49239). Return to text mode with LDA $C056 or peek(49238). The forty-column color-text switch is LDA $C051 or peek(49233). Return to normal text mode and normal use of the $800 through $BFF address range with LDA $C050 or peek(49232). John Jeppson described the soft switches for all of the Apple III display modes in the article mentioned earlier.

Here are the steps needed to complete the Hybrid Emulation disk after track 0 has been modified as described:

1. Using *Trackmover* and an Emulation disk already configured for lower case, load track 5 at $5000 and tracks 8 and 9 at $6000.
2. Use the Monitor to install the subroutine table in its new location with 59CE < FFE3.FFFFM and 74CE < FFE3.FFFFM. Move the character table with 5319 < FFCC.FFE2M and 6E19 < FFCC.FFE2M.
3. References to the Monitor data tables will be corrected with the following entries:

```
5048: 19 F8
5A7E: 19 F8
757E: 19 F8
```

```
5AC2:CE FE
75C2:CE FE
```

4. Disable the Applesoft lo-res-graphics commands gr, hlin, and vlin with

```
683E:60 EA EA
6990:60 EA EA
684C:60 EA EA
```

5. The disk will boot in forty-column, black-and-white text mode with two additional changes: 563A:50 and 313A:50.
6. Restore the three tracks to their proper locations on the Emulation disk.

Perhaps the most intriguing aspect of the Hybrid mode is the use of the 2-MHz processor speed with Applesoft programs. The 6502 clock can be set to 2 MHz for numerical processing but should be returned to 1 MHz for I/O operations. Since there is no SOS in this mode, you have the responsibility of taking care of it. Using the utilities shown in listing 3, call 842 to set the clock to 2 MHz and call 851 to return to 1 MHz. For even faster computations, you can turn off the video screen with call 860 and turn it back on with call 869.

As an example of the gains possible with this novel variation of Emulation, let's compare the performance of Applesoft at 2 MHz with that of the Basics tested on Jerry Pournelle's 20-by-20 matrix multiplication benchmark program described in the October 1982 issue of *Byte* magazine ("User's Column"). Table 2 shows the results obtained with several versions of Basic.

Business Basic, Video on	3:16
MBASIC (Softcard III)	3:13
Applesoft, 1 MHz (normal)	3:09
Business Basic, Video off	2:35
Applesoft, 2 MHz, video on	2:10
Applesoft, 2 MHz, video off	1:41

Table 2. Execution times for the 20-by-20 matrix multiplication benchmark program.

The results in the normal operating modes were surprising, because Business Basic is usually a little faster than Applesoft in numerical benchmarks. Business Basic has a definite advantage if there is much I/O. The Applesoft advantage at 2 MHz was expected, because it has none of the software overhead needed to manage the full Apple III memory.

All of the benchmark programs used the Apple III system clock as a stopwatch. Listing 3 includes a clock routine that can be used with Applesoft programs in the Hybrid mode. It reads all eight clock bytes into a buffer and then prints the hour, minute, and second on-screen without a carriage return. Each byte is encoded as two BCD nibbles, so the printed hex value appears to be the correct decimal number. For other uses, the BCD should be decoded to binary or ASCII.

The Hybrid mode provides full access to all of the Apple III hardware features, including memory bank switching, extended indirect addressing, and read/write RAM in $C000 through $FFFF, including the area normally used for I/O. SOS was designed to free the user from memory management details, but an imaginative hobbyist could create a 192K RAM-based pseudodisk for the Emulation Apple.

The extensions of the Emulation mode discussed in these three articles arose from a variety of intended applications. A solution to the games problem was required to please the younger members of the author's family when the Apple III replaced an Apple II. Lower-case character display and entry were needed for compatibility with Apple II software already on hand. The development of the Hybrid mode was spurred by a data-taking application that demanded use of the clock, fast computations, and usable software in just a few days. Taken together, these projects emphasize that the Emulation mode offers much more than a partial imitation of the Apple II. For those who care to explore it, the Emulation program provides a route to the very heart of the Apple III.

Listing 1. The character-set editing program, *Charedit*, in Integer Basic.

```
100 GOTO 5000
120 PRINT "START(HEX): ";: INPUT A$: GOSUB 2040
140 IF A=0 THEN END
160 L=A+1024
180 FOR I=A TO L
200 P= PEEK (I)
220 FOR J=1 TO 8
240 TAB J+1: IF P MOD 2 THEN PRINT "+";
260 P=P/2
280 NEXT J
300 GOSUB 1560: TAB 20: PRINT R$
320 X= PEEK (V)
340 IF X>128 THEN 440
360 A=0
380 NEXT I
400 A=0
420 GOTO 120
440 POKE Q,0
460 IF X=195 THEN 1000      (195 = "C")
480 IF X=211 THEN 1280      (211 = "S")
500 PRINT "EDIT WHAT ADDRESS?  ";: INPUT A$
520 GOSUB 2040
540 IF A=0 THEN 120
560 PRINT "INPUT AN 8-CHARACTER STRING"
580 PRINT "SPACE FOR BLANK"
600 PRINT "ANY OTHER CHARACTER FOR SET"
620 P= PEEK (A)
640 PRINT "OLD BYTE ->";
660 FOR J=1 TO 8
680 TAB 11+J: IF P MOD 2 THEN PRINT "X";
700 P=P/2
720 NEXT J
740 TAB 20: PRINT "<-"
760 PRINT "NEW BYTE ->";: INPUT S$
780 M=0
800 FOR K=8 TO 1 STEP -1
820 P=(S$(K,K)#" ")
840 M=2*M+P
860 NEXT K
880 I1=M: GOSUB 1580
900 PRINT "M = ";R$
920 I1=A: GOSUB 1580
940 PRINT "A = ";R$
960 POKE A,M
980 GOTO 120
1000  PRINT "CLEAR FLASHING BIT"
1020  PRINT "START ADDRESS:  ";: INPUT A$
1040  GOSUB 2040
1060  IF A=0 THEN 120
1080  S=A
1100  PRINT "LAST ADDRESS:  ";: INPUT A$
1120  GOSUB 2040
1140  IF S>A THEN 120
1160  FOR I=S TO A
1180  X= PEEK (I)
1200  IF X>=128 THEN X=X-128
1220  POKE I,X
1240  NEXT I
1260  GOTO 120
1280  PRINT "SET FLASHING BIT"
1300  PRINT "START ADDRESS:  ";: INPUT A$
1320  GOSUB 2040
1340  IF A=0 THEN 120
1360  S=A
1380  PRINT "LAST ADDRESS:   ";: INPUT A$
1400  GOSUB 2040
1420  IF S>A THEN 120
1440  FOR I=S TO A
1460  X-- PEEK (I)
1480  IF X<128 THEN X=X+128
1500  POKE I,X
1520  NEXT I
1540  GOTO 120
1560  I1=I
1580  R$=""
1600  F=0: IF I1>0 THEN 1660
1620  Z=NN*(I1< NN)
1640  I1= ABS ( ABS (I1-Z)-Z+MM):F=1
1660  FOR K=1 TO 4
1680  Z=I1/B:R=I1 MOD B
1700  N=R+1
1720  A$=H$(N,N)
1740  A$(2)=R$
```

```
1760  R$=A$
1780  IF Z=0 THEN K=4
1800  I1=Z
1820  NEXT K
1840  IF F=0 THEN RETURN
1860  M=4- LEN(R$):A$="0000"
1880  IF M=0 THEN 1960
1900  A$=A$(1, M)
1920  A$(M+1)=R$
1940  R$=A$
1960  A$=R$(1,1):R$=R$(2)
1980  C= ASC(A$)-167
2000  A$=H$(C,C):A$(2)=R$:R$=A$
2020  RETURN
2040  A=0
2060  FOR J=1 TO LEN(A$)
2080  C= ASC(A$(J,J))-176
2100  IF C>9 THEN C=C-7
2120  IF C<0 THEN 2240
2140  IF C>15 THEN 2240
2160  IF A>2047 THEN A=A-4096
2180  A=A+C+15*A
2200  NEXT J
2220  RETURN
2240  PRINT "HEX ENTRY ERROR"
2260  A=0: RETURN
3000  CALL -936
3020  PRINT "HEX TO DECIMAL CONVERSION"
3040  PRINT "ENTER 0 TO END"
3060  INPUT "HEX VALUE: ",A$
3080  GOSUB 2040
3100  IF A=0 THEN END
3120  PRINT "DECIMAL IS ",A
3140  PRINT : GOTO 3060
4000  CALL -936
4100  PRINT "DECIMAL TO HEX CONVERSION"
4120  PRINT "ENTER 0 TO END"
4140  INPUT "DECIMAL VALUE",I1
4160  IF I1=0 THEN END
4180  GOSUB 1580
4200  PRINT R$
4220  PRINT : GOTO 4140
5000  DIM A$(4),R$(4),H$(16),S$(16)
5020  B=16:H$="0123456789ABCDEF":NN=-16384:
      MM=NN+NN
5040  V=-16384:Q=-16368
5060  CALL -936
5080  VTAB 5
5100  PRINT "1 - CHARACTER SET EDITOR"
5120  PRINT "2 - HEX TO DECIMAL CONVERSION"
5140  PRINT "3 - DECIMAL TO HEX CONVERSION"
5160  PRINT "4 - QUIT"
5180  PRINT : PRINT : INPUT "CHOOSE A NUMBER",C
5200  IF C>3 OR C<1 THEN END
5220  IF C=3 THEN 4000
5240  IF C=2 THEN 3000
5260  CALL -936
5280  PRINT "      CHARACTER SET EDITOR"
5300  PRINT : PRINT
5320  PRINT "AT THE PROMPT 'START(HEX)' --": PRINT
5340  PRINT "ENTER THE HEX ADDRESS OF CHARACTERS"
5360  PRINT "      YOU WANT TO SEE DISPLAYED."
5380  PRINT "      CHARACTER BYTES AND ADDRESS"
5400  PRINT "      LABELS WILL SCROLL PAST."
5420  PRINT "ENTER 0 TO EXIT PROGRAM"
5440  PRINT : PRINT
5460  PRINT "DURING SCROLLING DISPLAY --": PRINT
5480  PRINT "HIT 'C' TO CLEAR FLASHING MODE"
5500  PRINT "             IN AN ADDRESS RANGE"
5520  PRINT "HIT 'S' TO SET FLASHING MODE"
5540  PRINT "ANY OTHER KEY TO REQUEST BYTE EDIT"
5560  PRINT
5580  PRINT "IF EDIT ADDRESS = 0, RETURN TO 'START'"
5600  PRINT : PRINT : PRINT
5620  INPUT "PRESS RETURN",A$
5640  GOTO 120
```

Listing 2. KEYIN3, an Emulation Monitor patch that allows full keyboard input in Emulation mode.

```
                 *****************************************
                 *
                 *      KEYIN3: Lower-case
                 *      keyboard input routine
                 *      for Apple II Emulation
                 *      mode on the Apple III.
                 *
                 *
                 *
                 *
                 *****************************************
                 *
                 KBDA      EQU  $C000     ;KEYBOARD "A" REGISTER
                 KBDB      EQU  $C008     ;KEYBOARD "B" REGISTER
                 KBDSTRB   EQU  $C010     ;KEYBOARD RESET ADDRESS
                 A4L       EQU  $42       ;MONITOR SCRATCH LOCATION
                 A4H       EQU  $43       ;SCRATCH, HIGH BYTE
                 *
                           ORG  $FEFE
FEFE: 84 43      KEYIN3    STY  A4H       ;SAVE Y
FF00: 86 42                STX  A4L       ;SAVE X
FF02: AD 00 C0             LDA  KBDA      ;CHECK KEYBOARD "A"
                                           REGISTER
FF05: A8                   TAY            ;STORE DATA IN Y
                                           REGISTER
FF06: AD 08 C0             LDA  KBDB      ;READ KEYBOARD "B"
                                           REGISTER
FF09: AA                   TAX            ;STORE IN X
FF0A: 2C 10 C0             BIT  KBDSTRB   ;RESET KEYBOARD
FF0D: 29 08                AND  #$08      ;WAS THE ALPHA LOCK
                                           KEY SET?
FF0F: D0 06                BNE  FILTER    ;NOT = MEANS NO
FF11: 98                   TYA            ;IF ALPHA LOCK, JUST
                                           TRANSFER DATA
FF12: A6 42      RTS1      LDX  A4L       ;RESTORE X
FF14: A4 43                LDY  A4H       ;RESTORE Y
FF16: 60                   RTS
FF17: 98         FILTER    TYA            ;CHARACTERS BELOW "A"
                                           ARE OKAY
FF18: C9 C1                CMP  #$C1      ;SO JUST RETURN
FF1A: 30 F6                BMI  RTS1
FF1C: C9 DB                CMP  #$DB      ;CHARACTER ABOVE Z?
FF1E: 10 F2                BPL  RTS1      ;YES, NO CHANGE
FF20: 8A                   TXA            ;THE REST TESTS THE
                                           SHIFT KEY
FF21: 29 02                AND  #$02      ;HERE'S THE SHIFT BIT
FF23: AA                   TAX            ;SAVE THE RESULTS
FF24: 98                   TYA            ;NOW GET DATA
FF25: E0 00                CPX  #$00      ;X=0 IF SHIFT
FF27: D0 E9                BNE  RTS1      ;ELSE NO CHANGES
FF29: 18                   CLC            ;DON'T CHANGE
                                           CHARACTER BY ACCIDENT
FF2A: 69 20                ADC  #$20      ;DO THE SHIFT
FF2C: D0 E4                BNE  RTS1      ;BRANCH TO RETURN
                                           (NEVER ZERO)
FF2E: 00                   BRK
```

Listing 3. Utility routines for use in the Hybrid Emulation mode.

```
                 *****************************************
                 *
                 *      APPLE III HYBRID
                 *       EMULATION MODE
                 *      Color-text control
                 *      1 MHz/2 MHz control
                 *      Video on/off switch
                 *
                 *
                 *
                 *
                 *****************************************
                 *
                 ENVIRON   EQU  $FFDF     ;ENVIRONMENT REGISTER
                 NEW       EQU  $D64B     ;APPLESOFT "NEW"
                 COLOR     EQU  $C051     ;COLOR SWITCH
                 INMASK    EQU  $F3       ;INTENSITY MASK
                 LINE      EQU  $40       ;LINE TO MASK
                 BASL      EQU  $42       ;LINE BASE ADDRESS
                 BASH      EQU  $43       ;HIGH BYTE
                 BASIC     EQU  $C00      ;START OF BASIC PROGS
                 ASLO      EQU  $67       ;PROGRAM START POINTER
                 ASHI      EQU  $68       ;HIGH BYTE
                 CLOCK     EQU  $C070     ;CLOCK CHIP ADDRESS
                 PRBYTE    EQU  $FDDA     ;PRINT BYTE SUBROUTINE
                 COUT      EQU  $FDED     ;CHARACTER OUTPUT
                 ZPAGE     EQU  $FFD0     ;ZERO-PAGE REGISTER
                 BUFFER    EQU  $278      ;= 760 DECIMAL
                 COLON     EQU  $BA       ;ASCII COLON
                 *
                           ORG  $0300
                 * Set Applesoft pointers so programs
                 * will start at $C00, rather than
                 * the usual $800
0300: A9 00                LDA  #$00
0302: 8D 00 0C             STA  BASIC
0305: A9 01                LDA  #$01
0307: 85 67                STA  ASLO
0309: A9 0C                LDA  #$0C
030B: 85 68                STA  ASHI
030D: 20 4B D6             JSR  NEW       ;CALL APPLESOFT NEW
                                           ROUTINE
0310: AD 51 C0             LDA  COLOR     ;TURN ON COLOR MODE
                 * Set intensity for normal screen
                 * and set to line 0.
0313: A9 F0                LDA  #$F0      ;CHAR ON, BACKGND OFF
0315: 85 F3                STA  INMASK
0317: A9 00      DOSCRN    LDA  #$00      ;DO WHOLE SCREEN
0319: 85 40                STA  LINE
031B: 20 27 03   DOLINES   JSR  CLINE     ;FROM LINE TO BOTTOM
031E: E6 40                INC  LINE
0320: A9 17                LDA  #$17
0322: C5 40                CMP  LINE
0324: B0 F5                BCS  DOLINES
0326: 60                   RTS
                 * Routine to calculate base address
                 * of line and set its intensity mask.
                 *  The routine is almost identical
                 *   with the BASCALC routine in the
                 *   Apple II Monitor, at $FBC1.
0327: A5 40      CLINE     LDA  LINE
0329: 4A                   LSR
032A: 29 03                AND  #$03
032C: 09 08                ORA  #$08      ;SELECT TEXT PAGE 2!
032E: 85 43                STA  BASH      ;ADDRESS HIGH BYTE
0330: A5 40                LDA  LINE      ;NOW DO LOW BYTE
0332: 29 18                AND  #$18
0334: 90 02                BCC  BSCLC
0336: 69 7F                ADC  #$7F
0338: 85 42      BSCLC     STA  BASL
033A: 0A                   ASL
033B: 0A                   ASL
033C: 05 42                ORA  BASL
033E: 85 42                STA  BASL
                 * Now put the value in INMASK into
                 * each position in the line.
0340: A0 27                LDY  #$27      ;CHARACTERS 0-39
0342: A5 F3                LDA  INMASK    ;VALUE TO STORE
0344: 91 42      STO       STA  (BASL),Y
0346: 88                   DEY
0347: 10 FB                BPL  STO
0349: 60                   RTS
                 *
                 * Set clock to 2 MHz
                 *
034A: AD DF FF   FAST      LDA  ENVIRON
034D: 29 7C                AND  #$7C
034F: 8D DF FF             STA ENVIRON
0352: 60                   RTS
                 *
                 * Set clock to 1 MHz
                 *
0353: AD DF FF   SLOW      LDA  ENVIRON
0356: 09 80                ORA  #$80
0358: 8D DF FF             STA  ENVIRON
035B: 60                   RTS
                 *
                 * Turn off video screen
                 *
035C: AD DF FF   VIDOFF    LDA  ENVIRON
035F: 29 DC                AND  #$DC
0361: 8D DF FF             STA  ENVIRON
0364: 60                   RTS
                 *
                 * Turn on video screen
                 *
0365: AD DF FF   VIDON     LDA  ENVIRON
0368: 09 20                ORA  #$20
036A: 8D DF FF             STA  ENVIRON
036D: 60                   RTS
                 *
                 * Subroutine to read Apple III
                 * system clock with Apple II
                 * software in Hybrid Emulation
                 *
036E: A2 07      RCLOCK    LDX  #$7       ;THERE ARE 8 BYTES
0370: 8E D0 FF   LOOP      STX  ZPAGE     ;SET CHIP REGISTER
0373: AD 70 C0             LDA  CLOCK     ;READ CLOCK
0376: 9D 78 02             STA  BUFFER,X  ;STORE DATA
0379: CA                   DEX
037A: 10 F4                BPL  LOOP      ;> =0, ANOTHER VALUE
037C: AD 7C 02             LDA  BUFFER+4  ;PRINT HOUR
037F: 20 DA FD             JSR  PRBYTE
0382: A9 BA                LDA  #COLON
0384: 20 ED FD             JSR  COUT
0387: AD 7B 02             LDA  BUFFER+3  ;MINUTE
038A: 20 DA FD             JSR  PRBYTE
038D: A9 BA                LDA  #COLON
038F: 20 ED FD             JSR  COUT
0392: AD 7A 02             LDA  BUFFER+2  ;SECOND
0395: 20 DA FD             JSR  PRBYTE
0398: 60                   RTS            ;RETURN TO CALLER
0399: 00                   BRK
```