NO. 48
NOVEMBER 1986

USA $3.50
CANADA $4.75

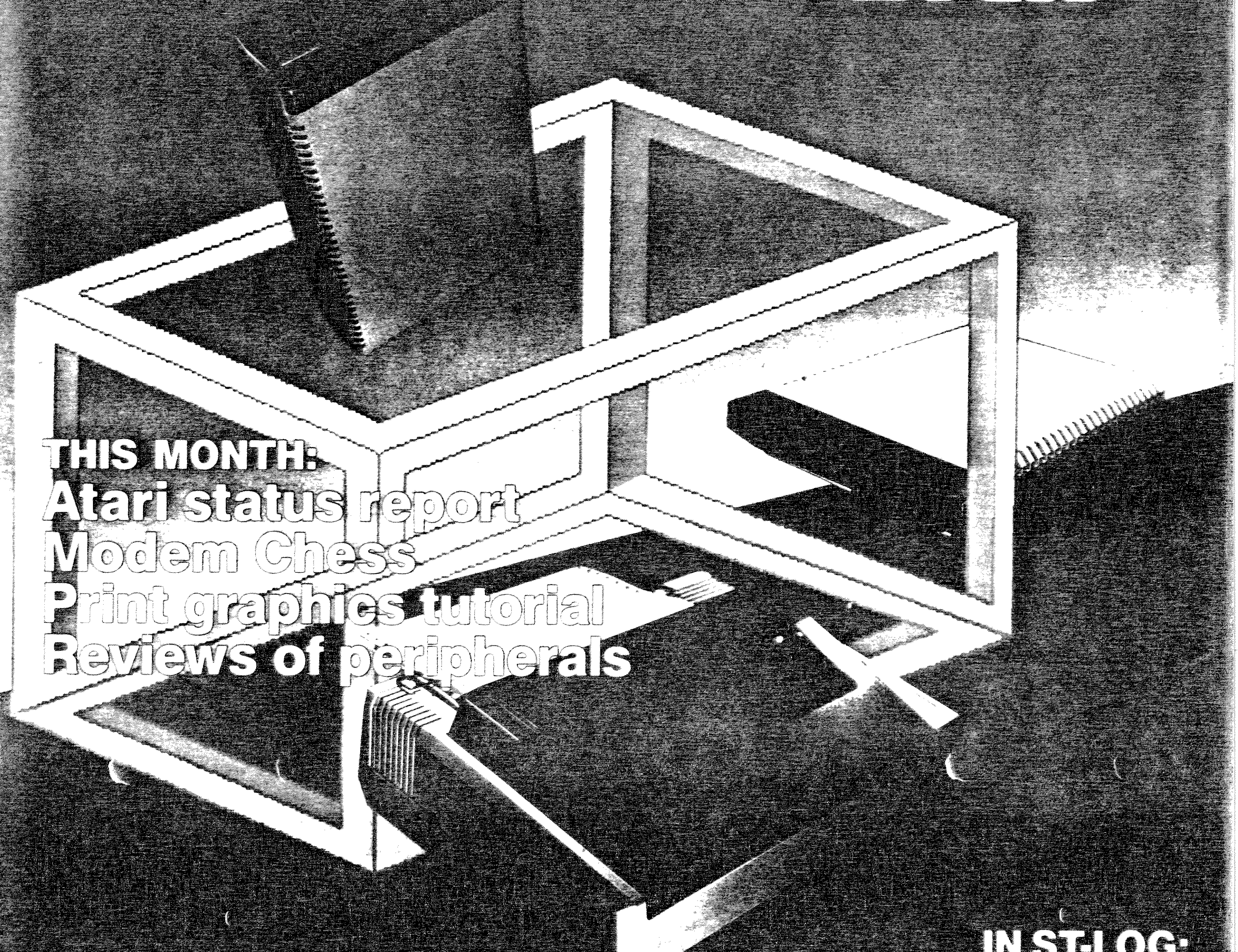
THE #1 MAGAZINE FOR ATARI® COMPUTER OWNERS

A.N.A.L.O.G. COMPUTING

INCLUDING

THE ATARI ST OPERATOR'S MAGAZINE

ST-LOG



THIS MONTH:

Atari status report
Modem Chess
Print graphics tutorial
Reviews of peripherals

IN ST-LOG:

Alert boxes
C compiler reviews

NO. 48

NOVEMBER 1986

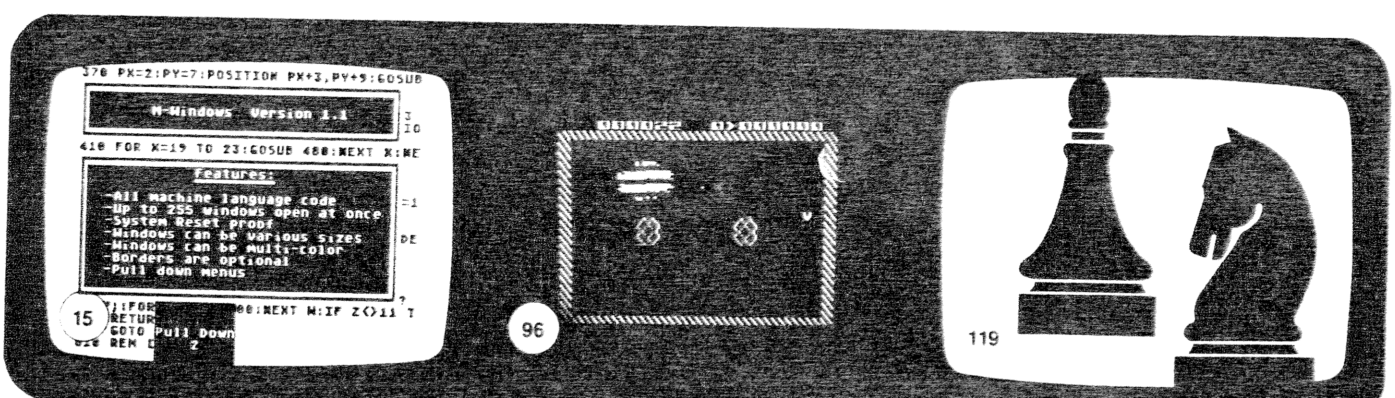
THE #1 MAGAZINE FOR ATARI® COMPUTER OWNERS

ANALOG COMPUTING

FEATURES

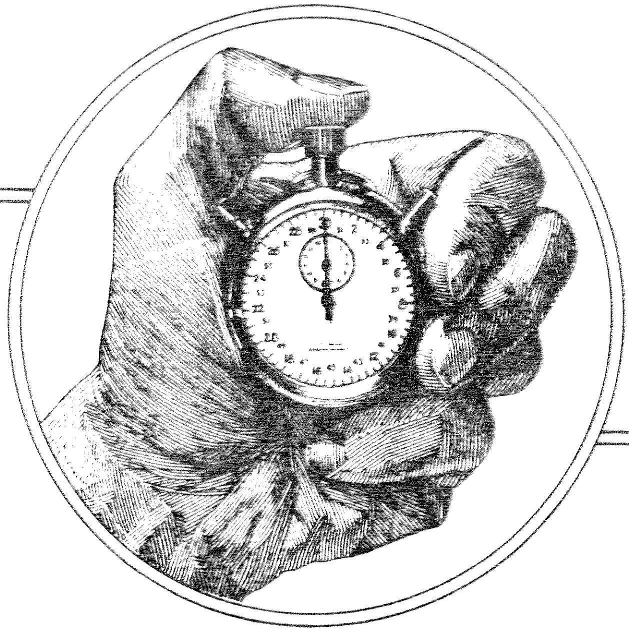
REVIEWS

COLUMNS



ANALOG Computing (ISSN 0744-9917) is published monthly for $28 ($36 in Canada, $39 foreign) per year by ANALOG 400/800 Corp., 565 Main St., Cherry Valley, MA 01611. Second class postage paid at Worcester, MA and additional mailing offices. POSTMASTER: Send address changes to **ANALOG Computing**, P.O. Box 625, Holmes, PA 19043.

DLIs



Another minute to learn

by Jonathan David Farley

Last month was the easy part. Now, you can control the (almost) omnipotent display list. Good for you. But don't start thinking about new ideas for your next arcade video game just yet! You still don't know (at least, not from me) what a DLI or—dare I say it?—display list interrupt is. Now that you know about displays lists, read on and find out about DLIs; the concept isn't as complex as you might think.

DLI talk.

A DLI is like a subroutine. The display list command equivalent of GOSUB is *128*; add 128 to the display list byte and you have it.

For instance, the computer sees a byte of *130* at DL+16 (where *DL* equals the address of the display list) and says, "Gee, here's another ANTIC mode 2 line. . . but what's this? An extra *128*? I see: the big guy out there wants me to do something. I guess I'll have to finish up what I'm doing now (which happens to be this mode line) and go to it. Then I'll come back and do the next line."

The computer needs to know where in memory the subroutine is; locations 512 and 513 handle that. You must also "turn on" the DLI by poking a *192* into the register NMIEN (decimal *54286*).

Save our registers.

DLIs are subroutines, but they're not written in BASIC. They're done in machine language. The first instructions in the DLI subroutine must save the computer's three special registers, called X and Y, and the Accumulator, or A. Why must we save the contents of these registers? When your DLI is finished and returns control to the main program, the computer expects to have everything just the way it left it. There may have been important data in those registers, and you surely changed them when you performed your DLI.

A DLI.

Sometimes, instead of waiting till the end of a line to start a DLI, ANTIC gets a little ahead of itself. If you use your DLI to change, perhaps, the background color, it will sometimes start changing the color mid-line, producing a ragged boundary (that shakes and changes position annoyingly) between one color and the next; on the same line.

By storing any nonzero value into a location named WSYNC at $D40A (the symbol for the dollar indicates a base sixteen, or hexadecimal, number) or 54282 decimal, you tell ANTIC, "Don't do this DLI until the horizontal blank."

Remember the horizontal blank? When the color is changed during the blank, you're ensured the new color will start on the following mode line.

```
*=$0600
PHA
TXA
PHA
TYA
PHA
LDA #$FF
STA $D40A
LDX #$0
STA $D017
STX $D018
PLA
TAY
PLA
```

```
TAX
PLA
RTI
```

The *=$0600 tells the computer to put the first byte of the program into location $0600 or 1536 decimal, with the rest of the program following. Commonly called page 6, this area (256 locations of RAM) is set aside for the programmer's use.

The next five mnemonic statements save the registers. The accumulator is "pushed" (saved in yet another memory area called the stack). The X- and Y-registers are, in turn, transferred to the accumulator and pushed onto the stack. And the end of the DLI, the register values are pulled back (in the reverse order of the way they were saved) and transferred back to their respective registers. (Just like logs piled onto a stack: the last log placed will be the first one taken off.) After all this, the computer returns from the interrupt and resumes going through the display list—like a BASIC RETURN.

At the start of the actual DLI, the computer loads the accumulator with *$FF* hexadecimal (255 decimal). It stores this number into memory location $D40A, WSYNC. Since this DLI changes the color of the screen, it should start doing so only after the horizontal blank for a clean switch of color. It also loads X with *$0*. It stores these registers' values in locations $D017 and $D018. You may ask, "So what?"

Poking shadows the hard way.

You may be aware of RAM locations 709 through 712, which determine the screen colors. The byte values in these locations are combinations of luminances and hues, to produce a myriad of shades and colors. These, however, are but shadows of the hardware locations the computer keeps for its own use.

Sure, you can POKE values into hardware locations, and, yes, the screen color does change. However, the screen is updated sixty times per second, and the computer gets the screen color from the shadows. End result: the screen flickers to your color, then, one-sixtieth of a second later, it's business as usual, switching back to the color in the shadow location.

Locations $D017 and $D018 are the hardware registers for locations 709 and 710. These registers keep track of the character and background colors, respectively. So. . .when the DLI routine stores the values in the A- and X-registers into these hardware locations, it's changing the color of the screen.

You may ask, "Yeah, but won't it just flicker back to the color in shadow registers?" Not at all, because you're going to keep poking that hardware register sixty times a second. The DLI is executed just as often as the computer draws the display. So the screen below the mode line with the DLI is the color you're placing in the hardware register. After the vertical blank (when this sort of thing is also done), the hardware register will again be set from its shadow, and hence the screen (up to the line with your DLI) will be normally colored—until the DLI comes around once more, that is. It changes the hardware locations, and everything starts over again (and again).

```
10 GRAPHICS 0
20 DL=PEEK(560)+256*PEEK(561)
30 FOR A=1536 TO 1539
40 READ B:POKE A,B:NEXT A
50 POKE 512,0:POKE 513,6
60 POKE DL+16,130:POKE 54286,192
70 DATA 72,138,72,152,72,169,255,141,1
0,212,162,0,141,23,208,142,24,208,104,
168,104,170,104,64
```

The *DATA* is the DLI POKEd into page 6. The DLI is in machine language, the result of assembling our assembly listing.

First, the program will find display list's start, POKE the DATA, and tell the computer where to find the DLI (6*256+0 =1536). ANTIC is told to interrupt after the mode line at DL+16, about halfway down the ANTIC 2 screen, and, finally, NMIEN is set to enable the DLI.

Here's a correlation between the DATA and the assembler routine: 72=PHA, 138=TXA, 152=TYA, 169=LDA with the following byte, 141=STA into the following 2-byte address (the first byte plus 256 times the second), 162 =LDX with the following byte, 142=STX into the following 2-byte address, 104=PLA, 168=TYA, 170=TXA, 64=RTI. There you go!

Whether you exclaimed, "Awesome!" or yawned, "Quite trite," I've got some advice.

DLIs are comparable to enclitics on words, or catalysts in chemical reactions—by themselves, they're virtually useless, but, combined with other techniques, they're extremely powerful. The stars are made of atoms, a brain of cells, and a computer (essentially) of bits. What I'm trying to tell you is: learn about your Atari.

Mirror Mirror on the screen.

For now, though, you can still do some "wild and crazy" things. Did you really believe I would leave out something to pop your eyes?

The program *Mirror Mirror* is shown in BASIC (Listing 1) and assembler (Listing 2).

The DLI at $0600 changes the screen color hardware registers, makes sure the characters are right side up, and puts the characters from the bottom half of the screen onto the top. Let's see how it does this, via a detailed examination of the assembler code.

The *=*$0600* informs the computer to place our DLI program into memory, starting at location $0600 (1536 decimal). The program then saves the registers and loads them with new values (after taking care of WSYNC). These new values are stored into locations $D017, $D018, and $D401.

What does location $D401 do?

In *Mirror Mirror*, $D401 (CHACTL) is used to cause the "flip effect:" it makes the characters appear upside down. Since I changed the hardware register (only hardware locations are usually used in DLIs, as I mentioned earlier), only the area of the DLI will be affected. CHACTL normally contains the number *2*. By putting a *6* into it, the letters are flipped.

The most important of this DLI's functions: it must take all the characters on the bottom half of the screen and put them on top, overlapping whatever else may already be there. Remember, the screen RAM is composed of 960 bytes and starts at location 40000 (in a 48K machine). Furthermore, each byte represents a character; 40 bytes represent one line of text.

The bottom twelve lines of text are the last 480 bytes; to mirror them, we must put the bytes of locations 40480-40959 (the bottom twelve lines' worth of characters) in their logically corresponding positions on the top half. The byte, and hence the character, in 40959 for instance, would be flipped into 40040—the vertical opposite but horizontal equal on the screen.

All that part three of the DLI does is load a byte from the location given, plus the value in X. For each new line, X is loaded with #0 and incremented after each byte stored. After every byte, the machine compares X to 40, to see if the whole line has been copied. The computer branches if X is not equal to 40, to do another byte.

Just before the computer pulls the registers back and returns to where it was prior to the interrupt, it does one more task: the DLI stores a $0 in 512 and a hexadecimal $40 (decimal 64) into 513. Locations 512 and 513 are the address of the DLI handle, remember? The computer reads these locations to find out where the DLI is when it encounters a DLI instruction in the display list. So, the next time your computer is told to interrupt the display list (in less than one-sixtieth of a second), it goes to 64*256+0 =16384 or $40*$ff+$0=$4000.

This second DLI also stores the values of A and X into the hardware color registers; these colors are different from those in the first DLI. In CHACTL, a 6 is stored from Y. This value in $D401 makes all characters under the DLI's influence appear flipped.

As with the first, this DLI has something that makes it special. The earlier DLI moves the bottom of the screen to the top. Because of this, anything typed on the top is changed to its bilateral counterpart on the bottom sixty times every second, erasing whatever you typed. Basically, you can't put anything on the top, though nothing stops the cursor from going there. This presents some problems in cybernetic aesthetics—easily solved problems, of course.

The DLI at $4000 cuts the screen's vertical length by two, as if something just crunched the screen together. This was done so the cursor could not go into the top, where it was useless. The display is still a normal screen, but only the last 480 bytes are displayed (and mirrored).

A certain memory location, $54 (84), has a value equal to the vertical position of the cursor. When humans count, they say, "One, two, . . . ten." Your logical Atari, though, says, "Zero, one, . . . nine." If PEEK(84)=0, the cursor's in the topmost row of the screen; if it is equal to 23, it's at the bottom. When you use the CTRL-ARROW key, the cursor goes to Line 23 when you go "over" the screen past Line 0, and to Line 0 when you go "below" Line 23. Location $54 changes accordingly.

Your friendly DLI at $4000 is on the lookout for $54. It constantly loads it into the accumulator and compares its value, to see if it's in the dreaded top half. It is, as I said, compared to $B (11 decimal). If the value in A (which is the value in $54) is equal to $B, a branch similar to a BASIC GOTO is taken, to TRTN. At TRTN, the accumulator is loaded with 0 and stored in $54. Since the value in $54 is the screen row of the cursor, the result is that, when the cursor gets to row 11, it's put in row 0. The next statement loads A with the ATASCII value of the ↑ function.

ATASCII values of characters are used in a system subroutine that starts at $F6A4 (Note: this address was changed on the XL and XE computers. For that reason, the cursor routine in the program won't function on these computers, unless a translator disk i loaded.) This subroutine prints the character whose ATASCII value is in the A-register. In this case, the character printed is the one that moves the cursor upward—since it's on Line 0, it would, after this, be on Line 23.

Why didn't I just put a 23 in $54 as soon as it got to #$B? One of the problems that had to be overcome was that, until a key is pressed, the cursor cannot be seen right after location $54 is changed. I solved it by "pressing" a key with the subroutine at $F6A4.

At the end of the DLI, the locations 512 and 513 are changed to $00 and $06, so the next time the computer encounters a "go to DLI" instruction in the display list, it will go to the one at $0600.

The program here gets the location of the display list from 560 and 561, and puts it in 203 and 204. It loads A with *$F0*—that's 112+128=240. It also loads Y with 2. The *STA (203),Y* means, "Add the value in Y to the address stored in 203 and 204, and store the value of A in the resultant address." We know the resulting memory location to be 2 bytes from the first byte in the display list, or DL+2. Since the first three instructions (DL+0, DL+1, and DL+2), each create eight blank lines, we must store *240* into the last one. Normally, DL+2 is *112*, but a *128* added to it gives *240*. When ANTIC comes to that, it knows it should execute a DLI. A second interrupt at DL+16 is staged. Since this location is normally an ANTIC mode line of two, it's now *130*. Finally, this program tells the computer where to go for the first interrupt (the location is stored in 512 and 513). After that, a *$C0* (or 192) is stored in MNIEN, *54286*. When that's done, it goes like clockwork!

The end—at long last.

Finally, you know all about DLIs. You've even explored some genuine assembler programs. It may have taken a bit longer than a minute to learn, but you have a lifetime to master it. ■

> The two-letter checksum code preceding the line numbers here is *not* a part of the BASIC program. For further information, see the *BASIC Editor II* in issue 47.

Listing 1.
BASIC listing.

```
RL 0 POKE 559,0:DATA 0,1,2,3,4,5,6,7,8,9,
   0,0,0,0,0,0,0,10,11,12,13,14,15
BZ 10 FOR A=1536 TO 1790:POKE A,0:NEXT A:
   FOR A=16384 TO 16639:POKE A,0:NEXT A:F
   OR A=24576 TO 24831:POKE A,0
OJ 20 NEXT A:DIM A(23),A$(1000),B$(120)
TE 30 FOR A=1 TO 23:READ B:A(A)=B:NEXT A
KH 40 FOR A=1 TO 5:READ B$:A$(LEN(A$)+1)=
   B$:NEXT A:B=1536:GOSUB 10000
PA 50 READ B$:A$=B$:READ B$:A$(LEN(A$)+1)
   =B$:B=16384:GOSUB 10000
QD 60 READ A$:B=24576:GOSUB 10000:POKE 55
   9,34:A=USR(24576)
SG 70 ? "K":POSITION 10,12:? "************
   ***********":POSITION 10,16:? "********
   ***************"
KA 80 POSITION 10,13:? "*  MirROR     MiRR
   Or *":POSITION 10,14:? "*           by
        *":POSITION 10,15
VL 90 ? "* Jonathan  Farley  *":STOP
SP 1000 DATA 488A489848A9808D0AD4A2FFA002
   8D17D08E18D08C01D4A200BDD89F9D409CE8E0
   28D0F5A200BDB09F9D689CE8E028D0
PW 1010 DATA F5A200BD889F9D909CE8E028D0F5
   A200BD609F9DB89CE8E028D0F5A200BD389F9D
   E09CE8E028D0F5A200BD109F9D089D
RS 1020 DATA E8E028D0F5A200BDE89E9D309DE8
   E028D0F5A200BDC09E9D589DE8E028D0F5A200
   BD989E9D809DE8E028D0F5A200BD70
CV 1030 DATA 9E9DA89DE8E028D0F5A200BD489E
   9DD09DE8E028D0F5A200BD209E9DF89DE8E028
   D0F5A9008D0002A9408D010268A868
HE 1040 DATA AA6840
VT 2000 DATA 488A489848A9FF8D0AD4A280A006
   8D17D08E18D08C01D4A554C90BF010C900D015
   A90B8554A91D20A4F64C3640A90085
RX 2010 DATA 54A91C20A4F6A9008D0002A9068D
   010268A868AA6840
SM 3000 DATA 68AD300285CBAD310285CCA9F0A0
   0291CBA982A01091CBA9008D0002A9068D0102
   A9C08D0ED460
XQ 10000 FOR A=1 TO LEN(A$) STEP 2:D1=A(A
   SC(A$(A,A))-47):D0=A(ASC(A$(A+1,A+1))-
   47):D=D1*16+D0:POKE B,D
DK 10010 B=B+1:NEXT A:RETURN
BK 10020 REM
BO 10030 REM
BS 10040 REM
DL 10050 REM This program turns off ANTIC
   so it may RUN faster without worrying
   about what's on the screen.
GK 10060 REM It loads the DLI's in $6000
   (1536) and $4000 (16384), and the main
   program in $6000 (24576).
YI 10070 END
```

●

Listing 2.
Assembly listing.

```
;********************
;*  MirROR   MiRROr  *
;*        by         *
;* Jonathan  Farley  *
;********************
;
;
; This DLI places the registers
;on the stack, LoaDs "A" with
;the characters' color and
;STores it in WSYNC for a clean
;change.  It also gets the
;characters' background color in
;"X" and the "no-flip" value in
;"Y," which are STored
;appropriately.  Then, line by
;line, byte by byte, it copies
;the characters on the bottom
;to the top.  It makes sure the
;next DLI the computer goes to
;is at $4000, and it gets back
;the registers.
      *=   $0600
      PHA
      TXA
      PHA
      TYA
      PHA
      LDA #$80
      STA $D40A
      LDX #$FF
      LDY #$02
      STA $D017
      STX $D018
      STY $D401
      LDX #$00
ONNE  LDA 40920,X
      STA 40000,X
      INX
      CPX #40
      BNE ONNE
      LDX #$00
TWWO  LDA 40880,X
      STA 40040,X
      INX
      CPX #40
      BNE TWWO
      LDX #$00
THRE  LDA 40840,X
      STA 40080,X
      INX
      CPX #40
      BNE THRE
```

(continued on page 130)

```
DT 850 DATA 21,53,50,50,48,32,63,32,35,67
   ,54,59,34,131,131,131,131,131,131,131,
   131,34
NQ 860 DATA 31,53,50,52,48,32,63,32,35,67
   ,54,59,34,132,133,134,139,138,134,133,
   132,34,58,71,79,84,79,32,53,51,52,48
XI 870 DATA 21,53,50,55,48,32,63,32,35,67
   ,54,59,34,132,133,134,138,139,134,133,
   132,34
JJ 880 DATA 21,53,50,57,48,32,63,32,35,67
   ,54,59,34,131,131,131,131,131,131,131,
   131,34
GH 890 DATA 21,53,51,49,48,32,63,32,35,67
   ,54,59,34,3,3,3,3,3,3,3,3,34
JH 900 DATA 21,53,51,51,48,32,63,32,35,67
   ,54,59,34,4,5,6,10,11,6,5,4,34
JT 910 DATA 108,53,51,57,48,32,88,61,85,8
   3,82,40,65,68,82,40,34,216,104,104,133
   ,204,104,133,203,104,133,206
ZC 920 DATA 104,133,205,104,133,208,104,1
   33,207,160,0,132,212,132,213,177,203,1
   45,207,230,212,165,212,208,2
HB 930 DATA 230,213,230,207,165,207,208,2
   ,230,208,230,203,165,203,208,2,230,204
   ,165,212,197,205,208,222,165
XZ 940 DATA 213,197,206,208,216,96,34,41,
   44,65,68,82,40,83,36,41,44,76,69,78,40
   ,83,36,41,44,83,84,65,82,84,41
```


DLIs

continued from page 111

```
0630      LDX #$00
0640 FOUR LDA 40800,X
0650      STA 40120,X
0660      INX
0670      CPX #40
0680      BNE FOUR
0690      LDX #$00
0700 FIVE LDA 40760,X
0710      STA 40160,X
0720      INX
0730      CPX #40
0740      BNE FIVE
0750      LDX #$00
0760 SIXX LDA 40720,X
0770      STA 40200,X
0780      INX
0790      CPX #40
0800      BNE SIXX
0810      LDX #$00
0820 SEVN LDA 40680,X
0830      STA 40240,X
0840      INX
0850      CPX #40
0860      BNE SEVN
0870      LDX #$00
0880 EGHT LDA 40640,X
0890      STA 40280,X
0900      INX
0910      CPX #40
0920      BNE EGHT
0930      LDX #$00
0940 NINE LDA 40600,X
0950      STA 40320,X
0960      INX
0970      CPX #40
0980      BNE NINE
0990      LDX #$00
1000 TENN LDA 40560,X
1010      STA 40360,X
1020      INX
1030      CPX #40
1040      BNE TENN
1050      LDX #$00
1060 LEVN LDA 40520,X
1070      STA 40400,X
1080      INX
1090      CPX #40
1100      BNE LEVN
1110      LDX #$00
1120 TWLV LDA 40480,X
1130      STA 40440,X
1140      INX
1150      CPX #40
1160      BNE TWLV
1170      LDA #$00
1180      STA 512
1190      LDA #$40
1200      STA 513
1210      PLA
1220      TAY
1230      PLA
1240      TAX
1250      PLA
1260      RTI
1270 ; Here, the registers are placed
1280 ;onto the stack, and the colors
1290 ;and flip value are chosen and
1300 ;STored.  "A" LoaDs the cursor
1310 ;row, and if it is #$B, it is
1320 ;changed to #$0 and moved up a
1330 ;row.  If the cursor row is not
1340 ;#$B or #$0, one row above or
1350 ;below the unmirrored portion
1360 ;of the screen, nothing happens
1370 ;at all.  If it is #$0, it is
1380 ;turned into #$B and moved
1390 ;downward.  The next DLI is
1400 ;made to be the one at $0600,
1410 ;and the registers are regained
1420 ;for a final departure.
1430      *=  $4000
1440      PHA
1450      TXA
1460      PHA
1470      TYA
1480      PHA
1490      LDA #$FF
1500      STA $D40A
1510      LDX #$00
1520      LDY #$06
1530      STA $D017
1540      STX $D018
1550      STY $D401
1560      LDA $54
1570      CMP #$0B
1580      BEQ TRTN
1590      CMP #$00
1600      BNE FRTN
1610      LDA #$0B
1620      STA $54
1630      LDA #'↓
1640      JSR $F6A4
1650      JMP FRTN
1660 TRTN LDA #$00
1670      STA $54
1680      LDA #'↑
1690      JSR $F6A4
1700 FRTN LDA #$00
1710      STA 512
1720      LDA #$06
1730      STA 513
1740      PLA
1750      TAY
1760      PLA
1770      TAX
1780      PLA
1790      RTI
1800 ; This is the true program:  It
1810 ;STores #$F0 and #130 in bytes
1820 ;2 and 16 of the display list.
1830 ;The first DLI is selected
1840 ;($0600) and NMIEN is secured.
1850      *=  $6000
1860      LDA 560
1870      STA 203
1880      LDA 561
1890      STA 204
1900      LDA #$F0
1910      LDY #$02
1920      STA (203),Y
1930      LDA #130
1940      LDY #16
1950      STA (203),Y
1960      LDA #$00
1970      STA 512
1980      LDA #$06
1990      STA 513
2000      LDA #$C0
2010      STA 54286
2020 END
```