I should mention that you can only put one interrupt at a given line. The following program draws and fills in a square. It uses mode 4 which has only two colors. But by changing color register 0 at two interrupts, you can have a tri-colored square. Actually, you can put 8 colors plus the background on the screen.

```
10 GRAPHICS 4%:COLOR 1%
20 PLOT 10%,5%
30 DRAWTO 60%,3%
40 DRAWTO 60%,45%
50 DRAWTO 10%,45%
60 FILL 10%,5%
70 SETINT@ 0%,16%,53270%,7%*16%+8%
80 SETINT@ 1%,32%,53270%,12%*16%+8%
90 GOTO 90
```

Lines 10 through 60 set the graphics mode and draw a rectangle. Lines 70 and 80 set two display list interrupts. For all of the graphic modes, the first three display list lines (0, 1, and 2) are blanks. That is, the 16% at line 70 sets a display list interrupt at display list line 16, which is screen line 13. At that line, color register 0 is switched from red to blue. At screen line 29, color register 0 is again changed; this time to green. Note that the change does not occur until the end of the line on which the interrupt is set. This is to avoid making a change during a line. The change from red to blue occurs at the start of line 14, and the switch to green at the start of line 30.

You might wonder why you didn't need to reset the color back to reddish-orange. The reason is that during a vertical blank (at the end of a display frame), all the color registers, as well as the alternate character set locations, are reset from special memory locations.

Before trying the above example, you will need to append the display list interrupt subroutines. Like PRINT USING, there isn't enough room to fit in these special routines. To use them, insert a Master disk (or another disk with the program) and then type APPEND DLISTINT.APP.

To remove an interrupt, set the last three integerexpressions to zero. The following command removes interrupt 1:

```
SETINT@ 1%,0%,0%,0%
```

CINT@

Format:

```
CINT@ integerexpression,integerexpression
```

The first integerexpression is the identifying number of the interrupt. CINT@ is used to change the value which is stored when the interrupt occurs. You can use CINT@ only after SETINT@ has been used. The second integerexpression gives the new value to be stored by the interrupt. You could use SETINT@ itself to change this value; however, SETINT@ has to wait until just the right point in the display to put the interrupt into place. CINT@ is much faster because it can make an immediate change.