Technical Forum

A Proposed Graphics Software Standard, Part 1

Vincent C Jones, 1913 Sheely Dr, Ft Collins CO 80526

A major stumbling block to making good software available in the personal computer market is the lack of standardization. Each manufacturer and software developer establishes internal standards for software and hardware interfaces, and they are usually incompatible with one another. Reasons for this vary from the experimenter's attempts to save 1 byte of memory in a 14 K byte program, to the mainframe manufacturer seeking to protect a development investment. The net result is the same. Extensive modifications are typically required to run software on any machine that differs from the original development's hardware and software configuration.

In an effort to prevent this fragmenting effect from overwhelming graphics applications programming, the following graphics interface software protocol is proposed as a standard.

This two-part article presents a complete microcomputer-oriented graphics software protocol and the algorithms required to implement it on typical raster scan graphics displays. The functions of hardware initialization, screen erase, point display, line generation, character generation, and animation are defined, and their implementation is demonstrated with a sample 8080/Z80 assembly language version for the Cromemco Dazzler. The power of a standard protocol is illustrated by a diagnostic demonstration program using the proposed 1 K byte 8080 assembly language protocol standard.

The standard actually proposes two separate but dependent protocols. The top-level protocol is machine independent. It defines a standard display coordinate system, several standard display modes, the available functions, and what these functions do. For example, a request for a red line from the center of the screen to the bottom right corner would always require the following command sequence:

CHAR (RED)	Set the current color to RED
CURSOR (128,128)	Move to the center of the screen
LINE (255,0)	Draw the line

Obviously, not all displays are capable of color; a black and white display would draw a white line instead. To compensate for any deficiencies in the hardware that is being used, a feedback path is included to inform the user program of the available capabilities. General-purpose programs can check to verify that the display being used is suitable and, if necessary, display an error (or warning) message, or use a different algorithm to accomplish the task at hand. For example, a TV tennis game could check to see if full color was available. If so, it could use red paddles, a yellow ball, a green court, and white boundaries. If only three colors were available, the paddles and ball could be the same color. If only a black and white display was available, all markings could be in white with a black court and background.

The lower-level protocol defines the calling sequences used in a particular programming language. When necessary, it also defines where the routines are loaded in memory, and the addresses of their calling vectors. Returning to the example of drawing a red line, an 8080 (or Z80) assembly language program would use the instruction sequence:

```
MVI     A,11H       ;Code for Red
CALL    0113H       ;Vector for CHAR
LXI     H,8080H     ;X = 128, Y = 128
CALL    010AH       ;Vector for CURSOR
LXI     H,FF00H     ;X = 255, Y = 0
CALL    0110H       ;Vector for LINE.
```

Similarly, a BASIC program would read:

```
REM — Set the current color to RED
CHA 17
REM — Move to the center of the
      screen
CUR 128,128
REM — Draw the line down to corner
LIN 255,0.
```

Suitable standards for other languages remain to be developed. Reader suggestions are welcome.

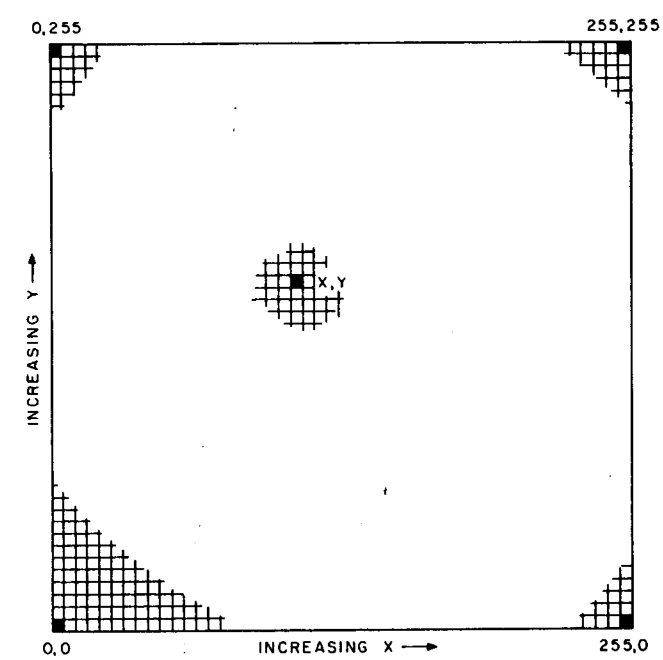


Figure 1: *Standard coordinate system used in the proposed graphics software standard.*

The Standard Display

The protocol defines a standard display device to circumvent hardware differences. The standard device displays 256 lines with 256 points on each line. As shown in figure 1, the origin (X = 0, Y = 0) is defined as the bottom leftmost point on the display. X increases to a maximum value of 255 as you move to the right, Y increases to 255 as you rise to the top. This defines the first quadrant of the standard Cartesian coordinate system. Each picture element (pixel) may be black, white, red, green, blue, yellow, cyan, or magenta (any combination of the three primary colors).

The display to be used is programmed to imitate the standard. To facilitate this procedure, four standard display modes are defined. Mode 0 requests the maximum possible resolution while mode 1 requests the maximum choice of colors. This allows for displays, such as the Cromemco Dazzler, which offer a trade-off between resolution and color. Two additional modes provide the ability to deliberately select larger pixels. Mode 2 is 128 by 128 resolution and mode 3 is 64 by 64 resolution. Regardless of the resolution actually used, the coordinate system remains at 256 by 256, as defined above. General-purpose applications programs can check to determine the available resolution and range of colors, whether the display is black and white or color, whether or not individual points can be erased, and if dual-buffered animation is available.

The Standard Functions

A five command repertoire is generally considered to be the bare minimum for a general-purpose graphics display. These commands provide all the output capabilities normally found on commercial nonintelligent graphics terminals, such as the Tektronics 4010. The routines are:

PAGE:	Next page, ie, erase the entire screen.
CURSOR (X,Y):	Position the cursor at the point X,Y.
DOT:	Set the pixel defined by the cursor position to the currently selected color.
LINE (X,Y):	Set the pixels along the line connecting the current cursor position to the point X,Y to the currently selected color.
CHAR (VAL):	Display the character whose ASCII value is VAL at the current cursor position using the currently selected color.

To facilitate matching the hardware requirements of many displays, an initialization command is also required:

INITG: Initialize the graphics subsystem.

Finally, a 2-buffer animation command is included for interactive graphics and game playing:

ANIMAT: Display the refresh buffer currently being filled and open a second refresh buffer for filling.

Display mode and current color selection are provided by the routine CHAR through ASCII control characters. Standard carriage control characters are also recognized. Display description parameters are returned by the routine INITG.

Let us now examine the function of each of the seven routines in detail.

INITG

The INITG function serves three primary functions. As an aid to the user, the display software is initialized to a standard configuration; the cursor is positioned at X = 0, Y = 0, the current color is set to white, the display is cleared, animation is disabled, and the display mode is set for maximum resolution (mode 0). Special options peculiar to the particular display are also disabled so that general-purpose programs do not have to be aware of them to function correctly. Secondly, this routine performs any initialization functions required by the display hardware. For those displays which refresh from program memory, the routine establishes the refresh buffers. If the display is under program control, it is turned on. Finally, INITG sets the display description variables to the appropriate values. Failure to initialize the display before using any of the other functions may lead to unpredictable and potentially disastrous results.

PAGE

The PAGE function clears the display screen. No other changes are made to the state of the display: the cursor is not moved, the current color is not changed, and the display mode is unaffected.

CURSOR

The CURSOR function sets the display cursor to a particular pixel on the screen. This establishes the initial location for the display functions which affect individual pixels on the screen. Coordinates are always interpreted on the 256 by 256 pixel matrix regardless of the actual resolution of the display. This is true even when the display mode is deliberately set to a lower resolution mode.

When in a lower resolution mode, the low-order bits of the position requested are ignored. For example, when in 128 by 128 resolution mode (mode 2), the points (8,4), (8,5), (9,4), and (9,5) will all be interpreted as the same pixel (the low-order bit in each coordinate has no effect).

Mnemonic	ASCII	Hexadecimal	Standard Function
			Display Mode Selection
MAXR	NUL	00	Maximum resolution
MAXC	SOH	01	Maximum colors
R128	STX	02	128 by 128
R64	ETX	03	64 by 64
RXXX	EOT	04	Undefined
			Carriage Control
BS	BS	08	Backspace (optional)
HT	HT	09	Horizontal tab (optional)
LF	LF	0A	Line feed
VT	VT	0B	Vertical tab (optional)
FF	FF	0C	Form feed
CR	CR	0D	Carriage return
			Character Style
SO	SO	0E	Undefined
SI	SI	0F	Undefined
			Current Color Selection
BLK	DLE	10	Black
RED	DC1	11	Red
BLU	DC2	12	Blue
MAG	DC3	13	Magenta
GRN	DC4	14	Green
YEL	NAK	15	Yellow
CYN	SYN	16	Cyan
WHI	ETB	17	White
NONE	ETX to GS	18 to 1F	Eight optional colors

Table 1: *Standard control character functions.*

When changing between display modes, cursor position is not required to be maintained by the interface software. To avoid erroneous results, all changes to display mode should be followed by a cursor positioning command.

DOT

The DOT function sets the display pixel indicated by the cursor to the currently selected color. With some displays in low-resolution mode, several physical pixels may be affected. For example, the Matrox ALT-256**2 turns on (or off, as selected) sixteen hardware pixels for every "dot" when in a 64 by 64 resolution mode.

LINE

The LINE function generates the line connecting the pixel defined by the cursor to the pixel requested. Both endpoints are included in the line. Therefore, a line of zero length is logically equivalent to a call to DOT. Care must be exercised when erasing or otherwise changing the color of a line, since the pixels in a line from pixel A to pixel B may differ from those used when the line is drawn from pixel B to pixel A. When lines are drawn in lower resolution modes, the pixels used are the size made by the DOT function at that resolution.

CHAR

The CHAR function provides the capability to display alphanumeric as well as graphical data. In addition, control characters provide limited cursor positioning and control over display mode and current color as shown in table 1. Control characters that are not recognized are ignored. Note that form feed positions the cursor only—it does not erase the screen.

Characters are positioned so that the cursor defines the lower left corner of a normal character (characters with descenders will extend below the cursor position). The cursor is left at the next character position. No check is made to detect characters off the edge of the screen. Parity is ignored. Lowercase characters, if not supported, are converted to uppercase.

ANIMAT

The function ANIMAT provides for flicker-free changes in the display by permitting the user to load one refresh buffer while displaying another. Each call to ANIMAT displays the buffer which is being filled, and opens another buffer for filling. This buffer exchange is performed at the start of the next vertical blanking period. Those displays without the ability to utilize multiple buffers but which *do* allow the erasing of individual pixels (such as the Matrox ALT-256**2) will just delay until the start of the next vertical blanking period. In either case, no changes are made to either buffer, and the cursor position is maintained. The ANIMAT function does nothing on those displays which support neither double buffering nor selective erase. To return to normal mode where updates are displayed in real time, it is necessary to reinitialize with INITG.

Standard Calling Sequences

To encourage maximum software interchange, two standard programming language protocols are currently defined. The first protocol is for 8080 and Z80 assembly language users, the second is for BASIC programs. By following one of these protocols, a program written for one display will work with any other display of sufficient resolution and color flexibility. The standard display and function definitions described previously are common to both protocols.

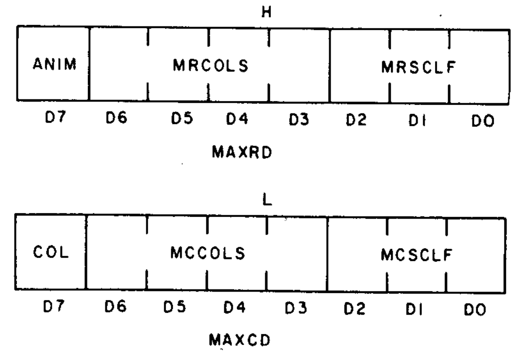
8080 Assembler Protocol

The 8080 assembly language interface is loaded into hexadecimal memory locations 0104 to 04FF. This provides a standard location for the package, regardless of memory size. To avoid conflict with programs requiring use of the restart (RST) instruction and most popular 8080 monitors, a lower starting address is not used. The first 21 bytes (hexadecimal 0104 to 0118) are the entry points to the different routines, as indicated in table 2. All arguments are passed to the called routine in register pair HL, except for the CHAR routine, which uses register A. The contents of all registers and flags are preserved, except for the INITG routine.

Routine INITG is called with the address of the first unused memory location above the program, to indicate

Routine	Vector Address (hexadecimal)	Parameters
INITG	104	HL = first free address Returns display description in HL
PAGE	107	None
CURSOR	10A	H = X coordinate; L = Y coordinate
DOT	10D	None
LINE	110	H = X end coordinate; L = Y end coordinate
CHAR	113	A = ASCII value of character
ANIMAT	116	None

Table 2: *8080 assembly language standard vector addresses.*



ANIM = 0 - Delay to start of vertical blanking.
1 - Double buffered animation supported.

COL = 1 - Display is in color.
0 - Display is black and white.

MRCOLS - Colors (grey shades) in MAXR mode.

MCCOLS - Colors (grey shades) in MAXC mode.

MRSCLF - $\frac{256}{\text{Display resolution}}$ in MAXR mode.

MCSCLF - $\frac{256}{\text{Display resolution}}$ in MAXC mode.

Figure 2: *8080 assembly language standard display parameter fields.*

available space for refresh buffers. While some displays do not require this information, it should always be included for compatibility. The address in HL is replaced by INITG with a 2-byte description of the display being used (all other registers and flags are left undisturbed). The format for these bytes is given in figure 2. The colors and scale factor fields which are available in register H describe the display when maximum resolution is selected; the same fields in register L describe the maximum color selection mode.

The available colors field gives the number of colors, other than white, to which a point can be written. If the field is zero, it means that the way to erase what has been written is to page the display. The scale factor field indicates the physical size of display points in standard coordinates. If the X and Y scale factors differ, the larger of the two is used. For example, if the display had 64 lines with 100 points on each, the scale factor would be four, based on the Y axis resolution.

The animation and color fields apply to all display modes. If the animation field is one, the display supports double buffered animation. If this field is zero, it is impossible to build one display scene while another is displaying. In this case the ANIMAT routine is a delay until the start of vertical blanking. The color/black and white field is self-explanatory: if it is one, the display is in color; otherwise it is black, grey, and white. Note that this field has no real meaning if the number of available colors is zero or one.

BASIC Protocol

For maximum flexibility and machine independence, a BASIC language usage protocol is also defined. Table 3 summarizes the commands and their arguments. Display initialization (IGR command) sets the variables A1

Mnemonic	Function	Arguments
IGR	INITG	None
PAG	PAGE	None
CUR	CURSOR	<X>, <Y>
DOT	DOT	None
LIN	LINE	<X>, <Y>
CHA	CHAR	<numeric ASCII value>
ANM	ANIMAT	None
TXT	PRINT	Equivalent to print except on display

Variable Name	Display Parameter
A1	X scale factor, high-resolution mode
A2	Y scale factor, high-resolution mode
A3	Available colors, high-resolution mode
A4	X scale factor, maximum color mode
A5	Y scale factor, maximum color mode
A6	Available colors, maximum color mode
A7	Animation support
A8	Grey scale

Table 3: *BASIC standard protocols.*

through A8 to reflect the display parameters. The scale factors A1, A2, A4, and A5, normally given exactly, are permitted to be rounded off to the nearest integer. These variables are ordinary BASIC variables and may be used and set as desired by the program.

The additional command TXT provides the user with the full flexibility of the BASIC PRINT command. Text and variables are displayed using the formats requested in the TXT statement starting at any location on the screen by using CUR to position the cursor. All characters are displayed using the current color.

Function Algorithms

To facilitate development of this standard, the algorithms used to produce the Matrox ALT-256**2 and the Cromemco Dazzler implementations of the 8080 assembly language standard are provided here. Of particular interest to most readers will be the line and character generation algorithms, which are independent of the hardware configuration of the display used.

For those readers not familiar with Nassi-Schneiderman design charts, a brief explanation is in order. More detailed information can be found in the original article published in the *SIGPLAN Notices* (August 1973). The Nassi-Schneiderman chart is a stylized flowchart for structured programming. By supporting only standard structured programming constructs (see figure 3) and not GOTOs and off page connectors, the chart forces the software designer to avoid the convolutions and obscurities in logic which make programs excruciating to debug and impossible to maintain.

The INITG and DOT routines are the only routines which normally require extensive adaptation to suit different displays. Since the Matrox ALT-256**2 is the only currently available low-cost display which is not direct memory access (DMA) refreshed from program memory and an enhanced 8080 assembly language package that is compatible with this standard is available from Matrox, the special considerations required to program I/O port driven displays are not included in this article. For direct memory access displays, the only other adaptations normally required are the refresh memory size parameter in

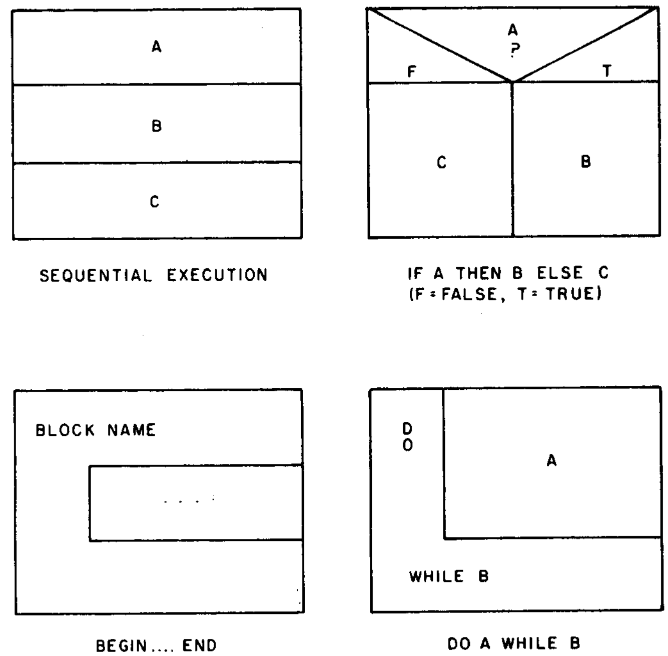


Figure 3: *Nassi-Schneiderman charts, a system of stylized flowcharts which are designed for use with structured programming techniques. Each of the charts physically resembles the program section it emulates. The charts are read from top to bottom.*

PAGE, the color and mode select controls in CHAR, and the scale factors used by the internal subroutine SCALE.

INITG Logic

Initialization is normally required for both hardware and software (see figure 4). The first step is to establish the refresh buffer. This requires taking the address which defines the top of the user program and moving up to the first address legal for refresh buffers. This address is needed by other routines, as well as for starting the display hardware. The different variables and flags are then set to the required values, and the page routine is called to clear the screen. The appropriate display

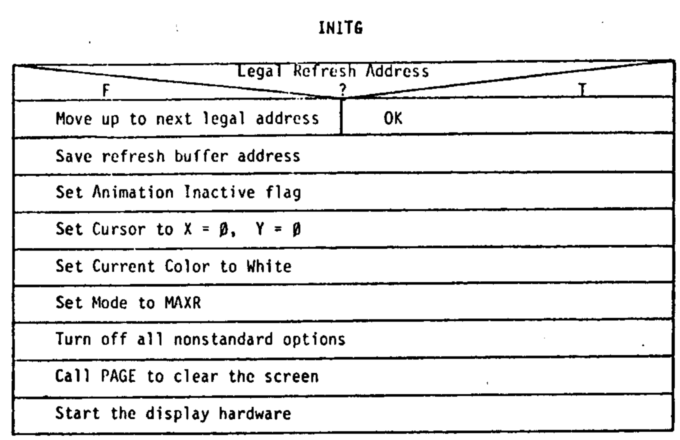


Figure 4: *The INITG function. INITG serves three purposes as an aid to the user: it initializes the system, performs any initialization functions required by the display software, and sets the display description variables to the appropriate values.*

PAGE

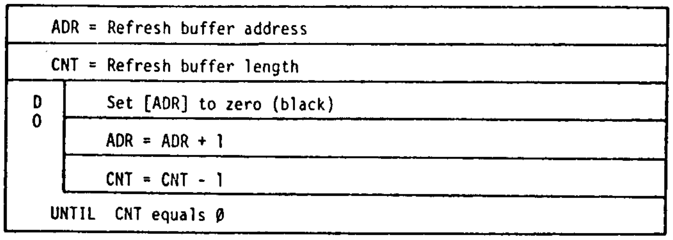


Figure 5: *The PAGE function. PAGE is used to clear the display screen.*

CURSOR

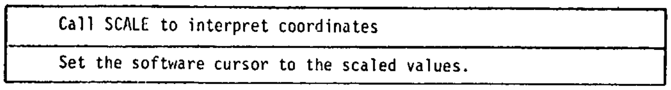


Figure 6: *The CURSOR function which sets the display cursor to a particular pixel on the screen.*

description is generated, and control is returned to the calling program.

PAGE Logic

The PAGE command clears all the memory used for display refresh (see figure 5). The most general algorithm, and the one that is charted, is clear byte, increment address, decrement byte count, and test for done. In machines with indexed addressing, the byte count can double as an index register. In machines with a memory-to-memory block transfer instruction, it is usually possible to clear one byte and transfer it to all of the display refresh memory.

CURSOR Logic

The CURSOR routine must convert from standard coordinates to software coordinates (see figure 6). Software coordinates are required by the LINE and CHAR algorithms to have a one-to-one correspondence with the actual display pixels being used. CHAR further requires X coordinates to increase to the right and Y coordinates to increase to the top. Since LINE must also scale its arguments, CURSOR and LINE can usually share the same internal scaling routine for efficiency.

DOT Logic

DOT is the only routine (other than PAGE) which actually modifies the refresh memory (see figure 7). Both LINE and CHAR use it to modify the desired pixels in the display. This routine is extremely hardware-dependent. Indeed, one of the primary reasons for defining this protocol was protection from differing display idiosyncracies. The DOT routine must translate the coordinates in the software cursor to the actual corresponding bits in memory. Remember that the software cursor is scaled so that a unit change in a coordinate is equivalent to the adjacent pixel. The logic presented here assumes a linear scan through refresh memory to generate the entire display, a line at a time, with the top line displayed first. Note that this algorithm is not adequate for the Dazzler, nor is it suitable for self-refreshed displays like the

DOT

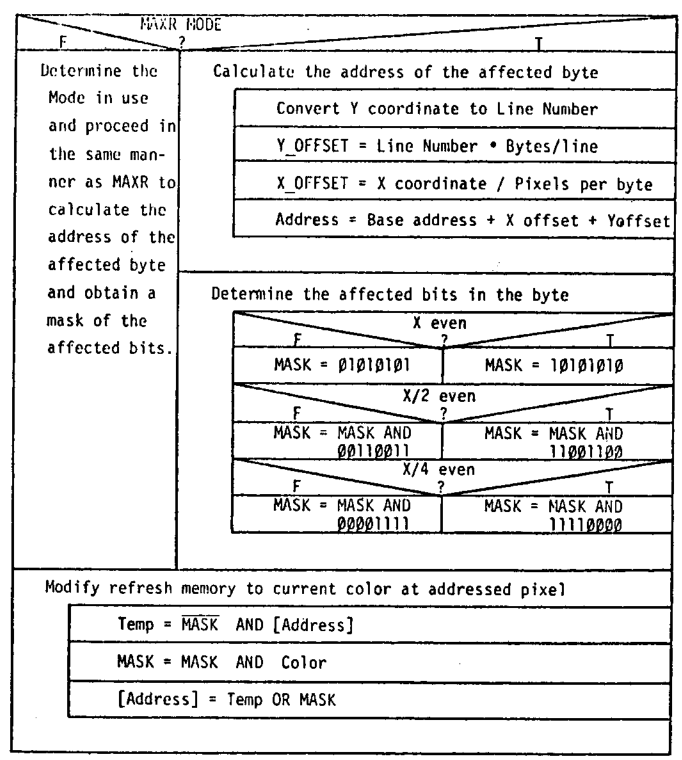


Figure 7: *The DOT function which sets the display pixel indicated by the cursor to the currently selected color.*

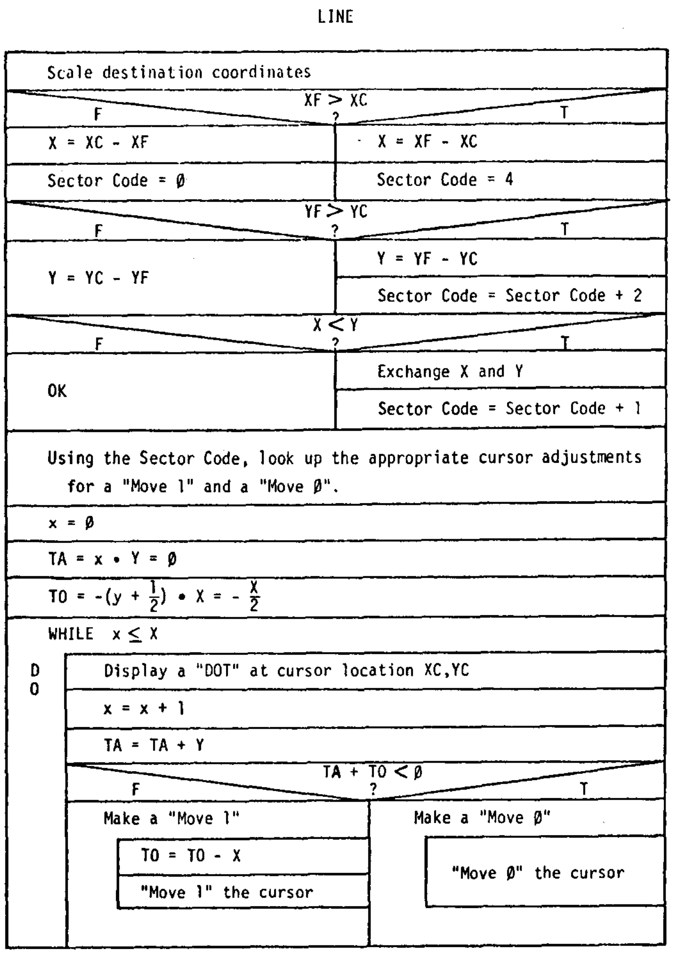


Figure 8: *The LINE function which generates the line connecting the pixel defined by the cursor to the pixel requested.*

Matrox ALT-256**2. The former divides the display into four quadrants, each in its own block of memory with every byte describing points on more than one line. The modifications to the algorithm are explained in the sample implementation, and need not concern the non-Dazzler owner. The Matrox's refresh memory is directly addressed by X,Y coordinates and no conversion is required.

The first step is to determine the address of the byte which contains the requested point. The cursor Y coordinate is converted to a display line number which, when multiplied by the number of bytes per line, gives the offset into the refresh buffer of the first byte on the line. The X coordinate corresponds directly to the desired point along the line. Dividing the X coordinate by the number of points in each byte gives the offset from the first byte in the line. Taking the base address of the refresh buffer (set up by INITG) and adding the offsets to the desired line in the buffer and the desired point on the line yields the address of the byte which requires modification.

The second step is to determine which bits in the byte correspond to the desired pixel. The hypothetical display depicted by the Nassi-Schneiderman chart has eight pixels in each byte. The selected bits are then changed to match the current color, and the refresh memory is updated to reflect the revised point. An effective procedure is to generate a mask which contains ones at bit positions corresponding to the addressed point, and zeros elsewhere in the byte. The byte of refresh memory is ANDed with the complement of the mask to delete the old contents. The mask itself is then ANDed with the bit pattern for a byte with every pixel. The current color and the result are ORed into the cleaned up byte of refresh memory.

LINE Logic

Perhaps the most crucial facet of any graphics system is its line generator (see figure 8). Before introducing the actual algorithm used, it may prove beneficial to discuss its theoretical development.

We wish to generate an arbitrary line from a point (XC, YC) to a point (XF, YF) (see figure 9). The goal is to determine those discrete points (x_n, y_n) which best approximate the desired line.

To simplify the derivation, we will only consider generating a line from point (0,0) to point (X,Y), where X is greater than or equal to Y and both are greater than or equal to 0 (figure 10). (This situation is general because any arbitrary line may be rotated and translated to match the proposed conditions.) Under these conditions, there is a point along the line for every value of x ($0 \le x \le X$), and for every value of x there is only one value of y. Closer examination reveals that for any value of x, the y value for the following point (x + 1) will either remain unchanged or increase by 1. No other value of y is possible. Furthermore, it can be shown that the decision to increment y for the next x is based solely on whether the point (x + 1, y + ½) lies above or below the line. If it lies above the line, y remains unchanged. If it lies below the line, y is incremented. In the event (x + 1, y + ½) is exactly on the line, either option is correct. For convenience, "on the line" is arbitrarily treated as equivalent to "above the line."

Assuming that we have a method to determine the position of the point (x + 1, y + ½) relative to the desired line, we can generate an optimal approximation of the line from (0,0) to (X,Y), where $X \ge Y \ge 0$, using the following algorithm:

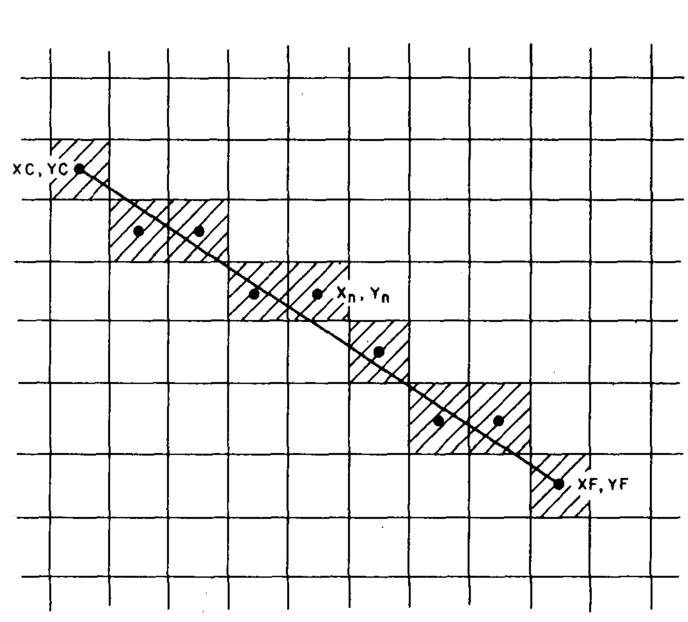


Figure 9: *Generating an arbitrary line.*

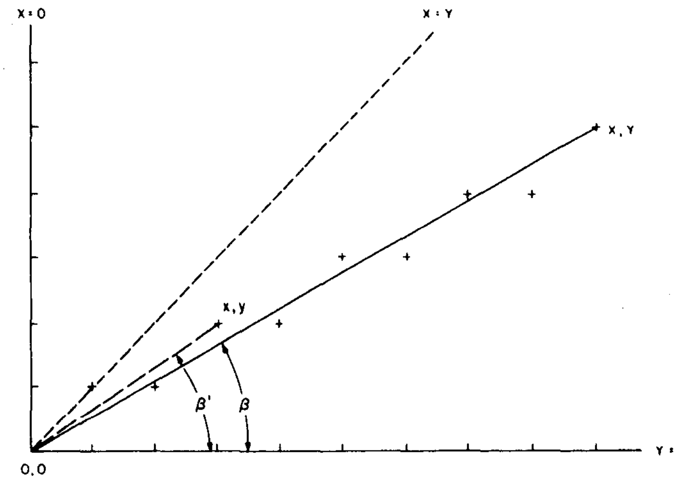


Figure 10: *Simplified line generation.*

1) Initialize x ← 0, y ← 0.
2) Display the point (x,y).
3) Test for done: x = X?
4) Calculate the position of the point (x + 1, y + ½) relative to the desired line.
5) Set dy to 1 if below the line; 0 if on or above.
6) Calculate the next point:
   x ← x + 1
   y ← y + dy
7) Go to step 2.

There are only two obstacles to overcome before implementing this algorithm: step 4 and the restrictive initial conditions. Let us examine each in turn.

A brief excursion into trigonometry is required to evaluate step 4. Referring to figure 10, if we call the angle between the desired line and the X axis θ, and the angle formed by the current point (x,y) the origin and the X axis θ', then if (x,y) lies above the desired line, $\theta < \theta'$. Conversely, if (x,y) lies below the desired line, $\theta > \theta'$. Of course, if the two coincide, $\theta = \theta'$. We know from trigonometry that for angles in the first quadrant, the greater the angle, the greater its tangent. We also know that the tangent of θ is $\frac{Y}{X}$, while that of θ' is $\frac{y}{x}$. Therefore, we can easily determine the position of any point relative to the desired line by comparing the quotients $\frac{Y}{X}$ and $\frac{y}{x}$.

Unfortunately, performing division on microcomputers is a time-consuming process. Using the properties of inequalities to eliminate the divisions, we can build a decision table (see table 4) which requires only multiplication. Returning to our original algorithm, we set dy to 1 if:

$$(x + 1) \times Y > X \times (y + ½)$$

and to 0 if it is not. Further advantage can be gained by realizing that at each iteration the product on the left side of the inequality increases by Y, while the right either remains the same or increases by X. By remembering the

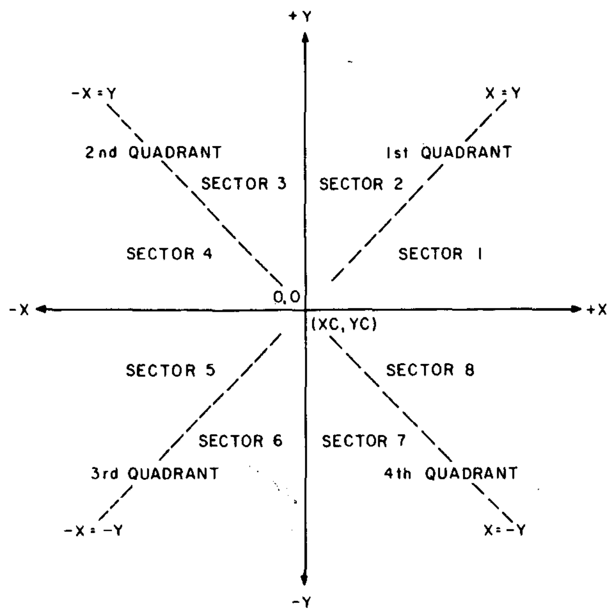


Figure 11: *Quadrant and sector definition.*

	Above	On	Below
Angle Relationship	$\theta < \theta'$	$\theta = \theta'$	$\theta > \theta'$
Tangent Relationship	$\frac{Y}{X} < \frac{y}{x}$	$\frac{Y}{X} = \frac{y}{x}$	$\frac{Y}{X} > \frac{y}{x}$
Relationship after Multiplying through by x.X	xY < Xy	xY = Xy	xY > Xy
Result of xY - Xy	Negative	Zero	Positive

Table 4: *Point position relative to a line.*

Quadrant	XM	YM
1	XF - XC	YF - YC
2	XC - XF	YF - YC
3	XC - XF	YC - YF
4	XF - XC	YC - YF

Table 5: *Component magnitudes in the four quadrants.*

Sector	Sector Weight	X	Y	Move 0 x incr	Move 0 y incr	Move 1 x incr	Move 1 y incr
1	6	XM	YM	+1	0	+1	+1
2	7	YM	XM	0	+1	+1	+1
3	3	YM	XM	0	+1	−1	+1
4	2	XM	YM	−1	0	−1	+1
5	0	XM	YM	−1	0	−1	−1
6	1	YM	XM	0	−1	−1	−1
7	5	YM	XM	0	−1	+1	−1
8	4	XM	YM	+1	0	+1	−1

Table 6: *Coordinate equivalents for each sector.*

products from the previous iteration, and whether or not y is incremented, the multiplication can be reduced to addition. For maximum efficiency, the right-hand product can be maintained negated so that the comparison can be made with a single addition.

The restriction that the line runs from (0,0) to a point (X,Y) with X ≥ Y ≥ 0 requires the use of coordinate translations, rotations, and reflections. The first step is to translate the line so that it starts at (0,0). Since the line originates at the cursor, we would traditionally subtract the cursor from the other endpoint to obtain its relative position. However, because a 256 by 256 display does not give us room for a sign-bit in an 8-bit byte, it is first necessary to rotate the line to the first quadrant and then calculate the magnitude of the endpoint displacements from the cursor.

While all these coordinate transformations may seem complicated, the actual implementation is quite simple. Consider the command to generate the line from the current cursor position (XC,YC) to a final point (XF,YF). The first step is to compare XF to XC. If XF ≥ XC then we are in the first or fourth quadrant (see figure 11); otherwise, we are in the second or third. Similarly, if YF ≥ YC, we are in the first or second quadrant; otherwise, the third or fourth quadrant. By combining the two results, the quadrant is uniquely determined, and we can proceed to determine the magnitude of the X and Y displacements, XM and YM, as shown in table 5. Finally XM and YM are compared to determine the exact sector.

The easiest technique for remembering this multiple logical decision is to weight the results of each decision and check the sum. Each sector is then assigned an equivalent weight, and the sector parameter table is reordered accordingly. Column 2 of table 6 applies a weight of 4 to (XF > XC),2 to (YF > YC) and 1 to (YP > XP).

Once the sector is determined, we have all the information required to construct any arbitrary line. Referring to step 5 of the fundamental sector 1 algorithm, we call setting dy to 0 "move 0," setting dy to 1 "move 1," and generate the equivalence chart in table 6. As the algorithm steps along in transformed coordinates, it uses the "move 0" and "move 1" to modify the cursor position using X and Y increments appropriate for the sector the line is actually in.

CHAR Logic

One of the most common formats for displaying characters is the 5 by 7 matrix of points (see figure 12). However, not many people realize why 5 by 7 is the smallest common size. The limiting width is, of course, the minimum number of points capable of displaying the three separate parallel lines required for the letters M and W. This sets the minimum possible width to 5, but why must 7 be the minimum height? The answer is, it need not be! However, human engineering studies have indicated that the average person finds it easier to read characters which are proportioned the same as in standard printing. Ratios of width to height far removed from the "normal" 0.75 increase fatigue and error rates.

To generate easily read lowercase characters, even larger matrices are required. This is a result of the greater complexity and finer detail of the lowercase characters. The full ASCII character set can be generated with a 7 by 9 matrix if provision is made for characters with descenders (g, j, p, etc). This requires the use of an extra

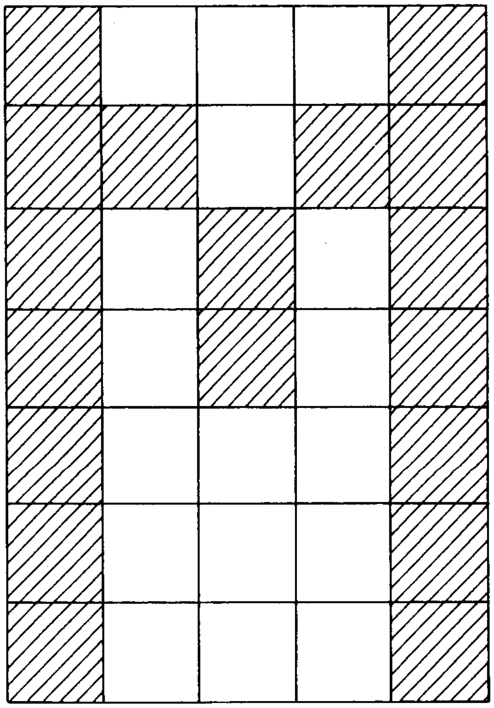


Figure 12: *Typical character generation.*

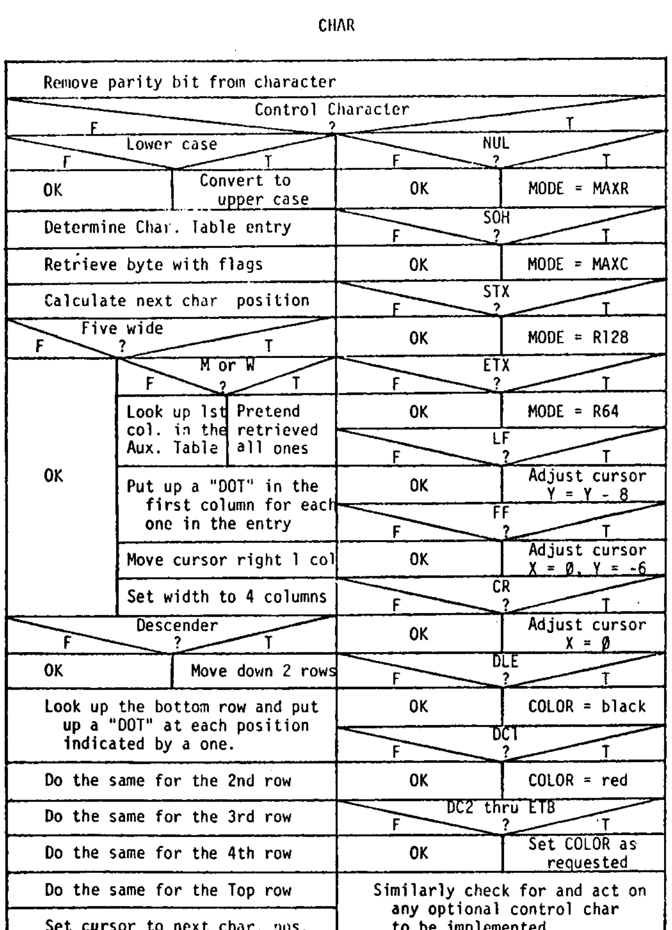


Figure 13: *The CHAR function which provides the capability to display alphanumeric as well as graphical data.*

Char Size	LC	Char/Line (256 by 256)	Lines/Page (256 by 256)	Memory For Tables (bytes)
9 x 11	Y	25	18	1200
7 x 9	Y	32	21	864
5 x 7	N	42	32	320
4 x 5*	N	64	32	192

*See text

Table 7: *Effects of differently sized character matrices.*

bit to determine if the matrix is displayed normally or shifted down two positions. As far as the display is concerned, the character uses a 7 by 11 matrix of display points. Larger display matrices can be used for greater legibility and varying character fonts, but even a 7 by 11 character matrix severely restricts the total number of characters that will fit on the low-resolution displays for which this standard is designed. If even one row of blank points is left between adjacent characters, then only sixteen 7 by 9 characters will fit across a 128-wide display. Memory requirements for large matrix character pattern storage are also severe. The table space required is directly proportional to the area of the matrix (see table 7).

A character matrix size less than the "absolute minimum" 5 by 7 was desirable, since even 5 by 7 characters require 320 bytes for their lookup table. Readable versions of 58 of the 64 uppercase printing ASCII characters can be generated within a 4 by 5 matrix. The remaining 6 characters (#, $, &, %, M, and W) fit in a 5 by 5 matrix. Since these are normally considered wide characters, their unity width-to-height ratio is not objectionable.

To simplify table lookups and the special handling of 5 wide characters, 3 bytes are used for each character. Twenty bits are used for the 4 by 5 display matrix; the four extra bits are used as flags to define the specific parameters for each character. Two flag-bits are used to indicate the width of the character. Proportional spacing also fits the maximum number of characters into any given space. The third flag-bit is used by 5 wide characters to indicate whether the first column is all ones (M and W), or must be retrieved from an auxiliary lookup table (#, $, %, and &). The remaining flag is used to indicate descending characters (, ; and _). These characters are displayed two positions lower than their matrices indicate. Each character is therefore displayed in an n by 7 display area, where n ranges from 2 to 5.

The basic character generation algorithm (figure 13) is applicable to any size character matrix, whether the character is stored by column (more efficient for 5 by 7 and 6 by 8 matrix characters), or by row (more efficient for variable 4 by 5, 7 by 9, and 8 by 11). If the character set being used does not include lowercase, it is necessary to shift lowercase characters to their uppercase equivalents. Comparing the ASCII value of the character to 32 separates control characters for special handling.

The character table is ordered by ASCII value and lookup is done by indexing on the ASCII value requested. Since the first 32 ASCII characters are control characters,

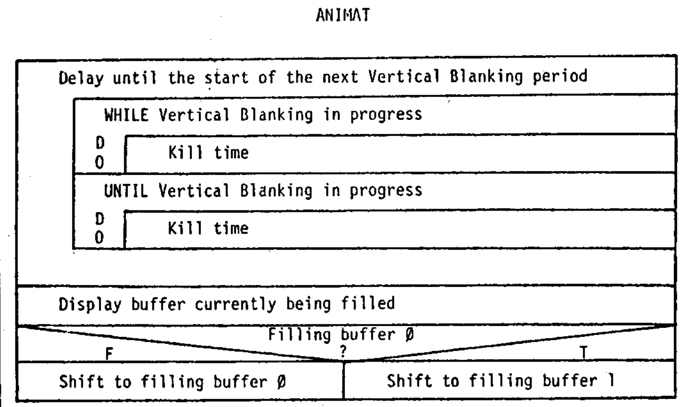


Figure 14: *The ANIMAT function which provides for flicker-free changes in the display by permitting the user to load one refresh buffer while displaying another.*

the physical contents of the table start with character 32 (blank). To index into the table, the ASCII value of the first table entry is subtracted from the value requested. This index value is then multiplied by the number of bytes per character, and the product is added to the address of the first character in the table in order to obtain the address of the first byte of the character desired. The cursor is then sequenced through the character matrix, turning on the points indicated. Only the points actually making up the character are affected, so background data is not erased and an overprint results.

Control characters are handled separately. Mode and color changes will depend on the DOT routine. Since these will be overly hardware-dependent, their implementation is left as an exercise to the reader. Carriage control characters modify the cursor position without otherwise affecting the display. Any unrecognized characters should be ignored.

ANIMAT Logic

The first requirement of the ANIMAT logic is to wait for vertical blanking to start (see figure 14). Most displays provide an input port with a status-bit which indicates when vertical blanking is in progress. By delaying until the status-bit indicates normal scan, then delaying until it indicates vertical blanking in progress, we are assured of a full vertical blanking period being available. If the display being programmed does not support changing the location of the refresh buffer by software controls, the routine is finished.

Displays in which refresh buffer locations can be changed are programmed to provide double buffering. After waiting for the vertical blanking period, the refresh buffer currently being filled is put on display. The alternate buffer is then opened for filling. Note that this algorithm is valid whether the buffer being filled is displayed (first call to ANIMAT after an INITG) or is being filled while another buffer is being displayed (all subsequent calls to ANIMAT).

In part 2 we will present an implementation of the 8080 assembly language protocol for the proposed graphics software standard, plus a series of demonstration programs. ■

Technical Forum

A Proposed Graphics Software Standard
Part 2

Dr Vincent C Jones, 1913 Sheely Dr, Ft Collins CO 80526

Sample Implementation

In part 1, the framework for a proposed graphics software standard was discussed.

An implementation of the 8080 assembly language protocol for use with the Cromemco Dazzler (listing 1) illustrates how the algorithms and standards presented translate into working software. Except for a few instances where the architecture of the 8080 or Dazzler allowed substantial simplification, the program code corresponds exactly to the Nassi-Schneiderman charts in part 1. The major deviations are in the handling of control characters in the routine CHAR, affected byte address calculation in DOT, and the termination condition in PAGE.

The software starts by defining the standard entry points. The Dazzler is assumed to be jumpered to use ports 16 and 17 (octal), the Cromemco default. If you own a Dazzler and it uses different ports, the I/O (input /output) commands in INITG, CHAR, and ANIMAT will need modification.

8080/Dazzler INITG

The first step in all these routines is to preserve any registers affected. In this case, HL is not saved because its contents will be replaced by the display description parameters.

The Dazzler requires the refresh buffer to start at an even multiple of 512. No test is made to check and see if the address provided is valid; instead, an algorithm that converts any address to a valid address and a valid address to itself is used. The refresh buffer address calculated is then stored in the two bytes labeled RBUF. Placing all the variables in a single section of memory is not only good programming practice, it also permits efficient setting of defaults by using register indirect addressing. The call to the CHAR routine with zero accumulator sets the display mode to MAXR and takes care of outputting the required controls to the Dazzler's Color/Mode port.

After calling PAGE to clear the screen, the Dazzler is finally turned on. The high-byte of the refresh buffer address is retrieved from memory and rotated into the bit position expected by the Dazzler. The OUT instruction starts the display, if it is not already on. The final step, before restoring register values, is to load the appropriate parameter description into HL. Hexadecimal 8AFC indi-

Text continued on page 176
Listing on page 84

Text continued from page 82:
cates that double buffered animation is available, MAXC mode has 15 colors and 64 by 64 resolution, the display is in color, and MAXR mode has one color and 128 by 128 resolution.

8080/Dazzler Page

The PAGE routine takes advantage of the hardware requirement that refresh buffers start only on even page boundaries and are 2 K bytes long. The low-byte of the address is used for a free zero, while the HL register is incremented until H corresponds to the high-byte of the first address beyond the buffer.

8080/Dazzler Cursor

Since the same scaling routine is used for both CURSOR and LINE, CURSOR becomes an almost empty routine. Aside from preserving registers, all it does is call CU000 with the coordinates presented, and save the scaled result as the new software cursor position XPOS, YPOS.

The MODE byte engages in some trickery to indicate the desired mode efficiently. The numeric value associated with the mode is rotated right one bit position. The resultant value can be incremented up to 126 times and still remain negative if in MAXC or R64 mode, and positive if in MAXR or R128 mode. Since MAXR on the Dazzler is 128 by 128 resolution, and MAXC is 64 by 64, we have a simple test to determine which mode is in use.

The scale routine CU000 divides X and Y by 2, checks to see if R128 or MAXR is selected, and divides again if they are not.

8080/Dazzler DOT

This routine tends to be somewhat complex due to the convoluted mapping from bits in the byte to points on the screen used by the Dazzler in 128 by 128 resolution mode, and the dividing of the screen into four quadrants. Fortunately, if the 128 by 128 coordinates are divided by 2, the address and mask generated by applying the algorithm for 64 by 64 resolution yields the four bits corresponding to the four possible 128 by 128 points. The low-order bits of the X and Y coordinates lost in the division are then used to select the single bit corresponding to the desired point.

The four quadrant problem is similarly solved by using the high-order bit of each coordinate to determine the quadrant, and the remaining lower order bits to find the location inside the quadrant. Since these problems are unique to the Dazzler, they will not be discussed further. The interested reader is invited to trace the logic in the program listing.

One final comment on the DOT routine is appropriate. The DOT register restore sequence is also used by LINE and CHAR. If it is changed, the appropriate modifications will also be required in LINE and CHAR.

8080/Dazzler LINE

The LINE routine is almost a block-for-block encoding of the LINE algorithm. The variable name correspondence table (table 8) is provided as a cross-reference guide, since some of the variable names used in the algorithm were modified.

Because the values of XP and YP are lost when the cursor adjustments for "move 0" and "move 1" are looked up, initialization of variables is moved to immediately after sector determination. TA and T0 are both 16-bit numbers because they represent the product of two 8-bit numbers. The only 16-bit arithmetic available on the 8080 is addition. To subtract X from T0, the 16-bit two's complement of X, DX, is calculated and added. Similarly, DY is the 16-bit representation of Y.

The cursor adjustments required for a "move 0" and a "move 1" are looked up in the table MXT. Entries are indexed by sector weight. Each entry is four bytes long (M0X, M0Y, M1X, and M1Y for the particular sector), so the sector weight is multiplied by 4 (two shifts left) and added to the starting address of the table. The correct cursor adjustments are then retrieved and stored where access is more convenient.

The only other significant change to the logic is the placement of the test for completion. For efficiency, x is compared to X immediately after the point is displayed. This has the added advantage of occurring at the only time the stack is free of temporary variables.

8080/Dazzler CHAR

The CHAR routine, with the exception of control character processing, also follows its Nassi-Schneiderman chart rigorously. The major change has been to convert to a SELECT construct the string of IFs used for control character processing. This avoids a multitude of tests which are guaranteed to fail once the character has been recognized. The processing of control characters with similar actions has also been consolidated to reduce redundancy.

As is obvious from its Nassi-Schneiderman chart, CHAR is really two independent routines with a common entry point. The only common code is the register saving and parity stripping. By pushing the address of the restore register routine onto the top of the stack, the return (RET) instruction will jump to the restore register sequence, restore all registers, and then return to the calling program.

The character matrix table is indexed by ASCII value minus 32, ie: the first entry is a blank. Since each entry is

Table 8: *Variable name definitions for LINE.*

8080 Software	Algorithm Description
XT	x
YT (not used)	y
XP	X
YP	Y
XPOS or XC	XC
YPOS or YC	YC
XF	XF
YF	YF
TA	TA
T0	T0
DX	−X
DY	+Y
M0X, M0Y	Cursor adjustment for a "Move 0"
M1X, M1Y	Cursor adjustment for a "Move 1"

Note: *The table numbering sequence is continued from part 1.*

three bytes long, the index must be multiplied by 3 to get the offset into the table. (The format of the character table is fully defined in the comments preceding it in the listing.) The first byte of each entry contains all the flags describing the character. The width bits are masked off and the cursor value for the next character position calculated. If the width is 6 (including a blank pixel between characters) the special subroutine to generate the first column of a 5-column wide character is executed. The descender indicator flag is then checked and the cursor is adjusted if necessary.

The normal character generation code scans the character matrix row-by-row. Whenever a 1 is encountered, the DOT routine is called to display the pixel at that location. When all five rows are completed, the cursor value is set for the next character position as calculated earlier, and control returned to the calling program.

The special subroutine used for five wide characters generates only the first column. By incrementing the cursor position, the normal character generation sequence is used to generate columns 2 through 5 instead of the normal 1 through 4.

Control character handling proceeds in three phases. Phase 1 checks for any of the four mode controls and sets MODE as required. The Dazzler hardware must also be informed so it can change mode. Phase 2 is entered if the control character is not a mode control. This is an individual check for each of the carriage control characters. Note that to get to the top line, form feed must determine what resolution is in use. Phase 3, if reached, is current color selection. The value of the control character is first checked to verify that it actually is a color select character. If it is black, the COLOR byte is set to all zeros. If any other color, a check is made to determine if the Dazzler is in a color supporting mode (MAXC or R64). If not, COLOR is set to all ones (high-resolution white). If a color mode is in use, the bright bit is set and the low-order four bits are duplicated in the high-half of the byte to yield a COLOR byte with the desired color in both pixel fields. Conveniently, the Dazzler color bit definitions match the lower three bits of the color select character.

A word of caution is in order for anyone using the compiler hexadecimal output in the listing directly, rather than the source code. The character table contains more bytes per line than the compiler used allocates for listing purposes (hence the "D" error). One must load the character table from the source code rather than from the compiler's hexadecimal output.

8080/Dazzler ANIMAT

The ANIMAT routine's implementation is adequately described in the comments on the listing. The flag byte ANIM indicates whether the first 2 K buffer or the second (auxiliary) 2 K buffer is currently being filled. Note that if the buffer swap were made as soon as vertical blanking was detected rather than as soon as vertical blanking was detected following an absence of vertical blanking, it would be possible to swap buffers, modify the display, and swap buffers again—all during one vertical blanking period. The net result, of course, would be that the one buffer would never be displayed, a clearly undesirable circumstance.

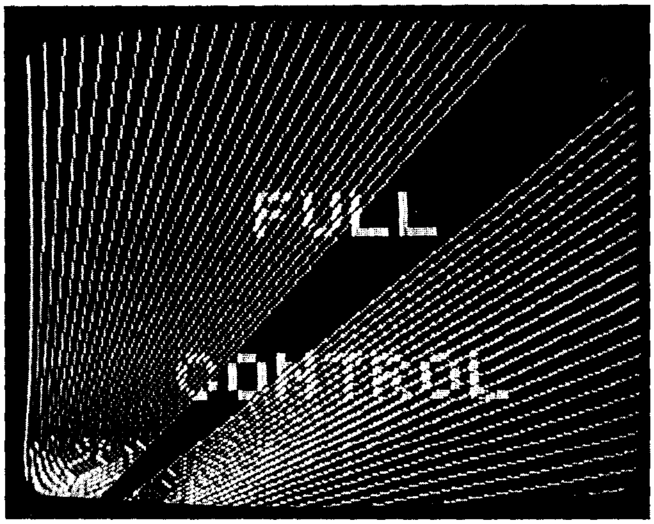


Photo 1: *Display generated by demonstration program number 2 (see listing 2).*

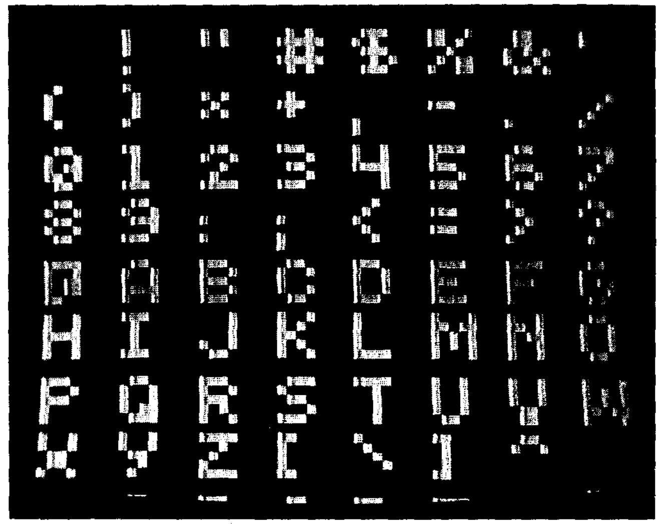


Photo 2: *Display generated by demonstration program number 3 (see listing 2).*

Demonstration Program

The demonstration program (see listing 2 on page 184) is provided for several purposes. Aside from demonstrating the power of the protocols, it serves as a tutorial in using the 8080 assembly language protocol and as a debugged, working user program for verifying successful implementation of the 8080 assembly language protocol. The photographs illustrating this article were all generated by this program and a Matrox ALT-256**2 display. The program contains four independent demonstrations and two utility subroutines. Equates are used to allow mnemonic references to the standard protocol's entry vectors, color controls, and display modes.

The first demonstration is a maximum-resolution exercise for the line generator. The identification message uses R64 resolution deliberately to get large characters. A series of maximum-length lines are drawn to generate the string art parabolas in each corner. The calculation of the endpoints of all the lines is simplified by the standard coordinate system. Their spacing is controlled by the value for MRSCLF returned by INITG. Because of the speed of generation, a variable delay utility subroutine, PAUSE, is used to give time to observe the display. These pauses may be extended indefinitely by setting the switch register to hexadecimal 01.

The second demonstration tests the generation of all 64 of the uppercase ASCII characters. Again, advantage is taken of the lowest resolution mode to display large characters. The 64 characters are drawn eight times, once in each color, in order to demonstrate the ability to vary the display dynamically. On the last iteration, the characters are drawn in black, leaving a clear screen. Rather than verify that the display is capable of selective erase, the PAGE routine is also called. The full range of available character sizes is then displayed using R64, MAXR, and R128 display modes for one line each. All mode changes are immediately followed by absolute cursor positioning commands to avoid erroneous results.

The third demonstration cycles through all available colors with the line generator. To avoid claiming Full Color Control on a monochrome display, the color bit in MAXCD is tested. MCCOLS is then checked to see how many colors or grey shades are available. All available colors are used, one at a time, as one end of each line is moved closer to (255,255). The attempt at mode RXXX, after shifting to R64, is ignored by the package in this article. The enhanced Dazzler package available from Cromemco uses it to select the Dazzler's 16-level grey scale mode.

The final demonstration is a short animation sequence. The header is inserted in both buffers. The auxiliary buffer must be cleared first, since this function is not included in the standard. If the display is not double buffered, this will also clear any warning messages generated by the graphics package.

The algorithm used to animate the figure will work with either double buffered displays or selectively erasable displays. For the former, the figure is backed up one step and drawn in black to erase it from the non-displaying buffer (PAGE would require too much time and erase the header). The figure is then advanced two steps to get to the position past the one currently being displayed and drawn in white. Finally, ANIMAT is called to display the updated buffer, and the whole procedure is repeated until the screen is traversed. If the display is not double buffered (tested using the ANIM field in MAXRD), the ANIMAT routine is called anyway to delay until the start of vertical blanking. While the display is busy with vertical blanking, the old figure is erased and the new one displayed. If all the changes can be made before the affected memory is displayed, there will not be any flicker, and the animation will be as smooth as when double buffering is used.

The STRING subroutine is a convenient utility for displaying text strings. It calls the CHAR routine with each successive character in a string of ASCII characters until an ASCII '$' (hexadecimal 24) is detected.

Conclusion

The availability of a powerful graphics protocol immensely simplifies the design and coding of graphics programs. The limitations imposed by forcing individual capabilities to meet a common protocol are more than made up by the availability of precisely defined functions and controls. Furthermore, the protocol is sufficiently flexible to allow the installation and use of unique display features without adversely affecting the ability to run programs designed to the standard. For example, the package available from Matrox for its ALT-256**2 contains such enhancements as high-resolution positioning of low-resolution DOTs, choice of fixed or proportional character spacing, and up to 8 bits (256 combinations) color and/or grey scale for each pixel.

The author would like to thank John Rogers, Gary Johnsey, and especially Bart Schwartz for their help in making these articles possible.

Graphics Interface Standard for FORTRAN

The following FORTRAN subroutine definitions extend the flexibility and hardware independence of the proposed microcomputer graphics standard to FORTRAN.

INITG (XMRSCL, YMRSCL, MRCOLS, XMCSCL, YMCSCL, MCCOLS, LANIM, LCOLOR)

Initialize graphics hardware and software to maximum resolution mode with all options disabled. The screen is cleared and the current color is set to white. Eight variables are used to return the display parameters:

*XMRSCL (REAL*4) X dimension of physical display points in standard coordinates, maximum resolution mode.*
*YMRSCL (REAL*4) — as above except Y dimension.*
*MRCOLS (INTEGER*2) — colors (grey shades) available in maximum resolution mode.*
*XMCSCL (REAL*4) — X dimension of physical display points maximum colors mode.*
*YMCSCL (REAL*4) — as above except Y dimension.*
*MCCOLS (INTEGER*2) — colors (grey shades) available in maximum color selection mode.*
*LANIM (LOGICAL*1) — TRUE if double buffered animation available.*
*LCOLOR (LOGICAL*1) — TRUE if display is in color, FALSE implies monochrome.*

PAGE

Clear the sreen

CURSOR (IX, IY)

Move the cursor to the coordinate position specified.

*IX (INTEGER*2) — X (horizontal) coordinate desired. Value is in standard display coordinates (0 through 255). Out of range values are permitted but may have unpredictable results.*

*IY (INTEGER*2) — as above except Y (vertical) coordinate desired. Lower left-hand corner of the screen is the point 0,0.*

DOT

Display a dot at the current cursor position using the current color.

LINE (IX, IY)

Display a line from the current cursor position to the coordinate position specified. IX and IY are defined as in CURSOR.

CHAR (ICHAR)

The ASCII character defined by the low-order 7 bits of ICHAR is displayed at the current cursor position. Control characters are interpreted as defined in the standard to change display mode, current color, etc.

*ICHAR (INTEGER*2) — the ASCII character to be interpreted or displayed.*

ANIM

Program execution is delayed until the start of the next vertical blanking period. If double buffered animation is supported, buffers are not switched until immediately before returning.

WRITE (10, nnn) var, var, ...

The logical unit number 10 is available for formatted output to the display. Binary output will result in an I/O (input/output) error. Input attempts will return End of File. Rewind, endfile, and backspace operations are no-ops. The display must be initialized by INITG before writing to LUN 10. The first character output on each line is interpreted as a standard FORTRAN printer control character (' ' for single space, '0' for double space, '1' for new page, and '+' to overprint the same line).

Sample Program

```
C---   Example usage of FORTRAN Standard Graphics Calls
         LOGICAL*1 LANIM, LCOLOR
C---   Initialize graphics
         CALL INITG(XMRSCL, YMRSCL, MRCOLS, XMCSCL,
       1 YMCSCL, MCCOLS, LANIM, LCOLOR)
C---   Title display
         WRITE (10, 100)
100      FORMAT(1H1, 'A SINE WAVE')
C---   Calculate and display a sine wave
C---   Move to starting point
         CALL CURSOR (0, 128)
C---   Determine distance between X values
         INCR = IFIX (YMRSCL + 0.5)
         IF(INCR.LE.0)INCR = 1
C---   Draw the actual curve
         DO 1000 IX = INCR, 255, INCR
         X = 3.14159*FLOAT(IX)/64.0
         Y = SIN(X)*100.0
         IY = IFIX (Y + 128.0)
1000     CALL LINE (IX, IY)
         END
```

Listing 1: *Implementation of the 8080 assembly language protocol for use with the Cromemco Dazzler. With a few exceptions, the program corresponds exactly to the Nassi-Schneiderman charts in part 1.*

```
                          ;              THE VCJ GRAPHICS PACKAGE
                          ;                8080/DAZZLER VERSION
                          ;           VERSION 3.02B  <>  AUG 25, 1977
                          ;
                          ;       ********* COPYRIGHT NOTICE *********
                          ;       *                                  *
                          ;       *                                  *
                          ;       *          COPYRIGHT (C) 1977      *
                          ;       *         DR. VINCENT C. JONES     *
                          ;       *                                  *
                          ;       *                                  *
                          ;       *   COMMERCIAL USE OR DISTRIBUTION *
                          ;       *  IS PROHIBITED WITHOUT THE       *
                          ;       *  EXPRESS WRITTEN CONSENT OF THE  *
                          ;       *  COPYRIGHT OWNER.  REPRODUCTION, *
                          ;       *  MODIFICATION OR ADAPTATION FOR  *
                          ;       *  PERSONAL USE IS PERMITTED PRO-  *
                          ;       *  VIDED THIS NOTICE IS INCLUDED.  *
                          ;       *                                  *
                          ;       ************************************
                          ;
                          ;
                          ;JUMP TABLE TO DEFINE STANDARD ENTRY POINTS
                          ;
0104                              ORG     104H    ;START OF STANDARD SPACE
0104 C31901                       JMP     INITG   ;INITIALIZE GRAPHICS
0107 C34501                       JMP     PAGE    ;CLEAR THE SCREEN
010A C35901                       JMP     CURSOR  ;GO TO A POINT ON THE SCREEN
010D C37701                       JMP     DOT     ;DISPLAY A POINT ON THE SCREEN
0110 C3F101                       JMP     LINE    ;DRAW A LINE BETWEEN POINTS
0113 C38F02                       JMP     CHAR    ;DISPLAY AN ASCII CHARACTER
0116 C38F03                       JMP     ANIMAT  ;CHANGE BUFFERS WITHOUT FLICKER

                          ;DEFINE THE DAZZLER PORTS
000E =                    DAZ0    EQU     0EH     ;CONTROL PORT
000F =                    DAZ1    EQU     DAZ0+1  ;COLOR/MODE PORT

                          ;
                          ;ROUTINE INITG
                          ;       INITIALIZE THE DAZZLER TO 128 BY 128 B/W MODE,
                          ;          X = 0, Y = 0, SCREEN CLEARED, AND
                          ;          CURRENT COLOR SET TO WHITE.
                          ;       H,L CONTAINS THE FIRST AVAILABLE ADDRESS
                          ;          FOR REFRESH BUFFERS.
                          ;       RETURNS DISPLAY CHARACTERISTICS IN H,L
                          ;
0119 F5                   INITG:  PUSH    PSW     ;SAVE A FEW REGISTERS
011A D5                           PUSH    D       ;
011B 2B                           DCX     H       ;FIX REFRESH ADDR TO LEGAL
011C 7C                           MOV     A,H     ;  BOUNDARY
011D C602                         ADI     02H     ;
011F E6FE                         ANI     0FEH    ;CLEAR 256 BIT
0121 67                           MOV     H,A     ;
0122 AF                           XRA     A       ;CLEAR A
0123 6F                           MOV     L,A     ;  AND MAKE LOW BYTE ZERO
0124 22A904                       SHLD    RBUF    ;SAVE BUFFER ADDRES
0127 21AB04                       LXI     H,ANIM  ;START OF VARIABLE AREA
012A 77                           MOV     M,A     ;FILLING BUFFER 0
012B 23                           INX     H       ;REAIM AT YPOS
012C 77                           MOV     M,A     ;   WHICH IS ZERO
012D 23                           INX     H       ;REAIM AT XPOS
012E 77                           MOV     M,A     ;  WHICH IS ZERO
012F 23                           INX     H       ;REAIM AT CURRENT COLOR
0130 36FF                         MVI     M,0FFH  ;  WHICH IS WHITE
0132 CD8F02                       CALL    CHAR    ;128 BY 128 MAX RESOLUTION MODE
0135 CD4501                       CALL    PAGE    ;AND FINALLY A CLEAR SCREEN
0138 3AAA04                       LDA     RBUF+1  ;RETRIEVE REFRESH ADDRESS
013B 37                           STC             ;FUTURE ON BIT
013C 1F                           RAR             ;NOW HAVE A DAZZLER CONTROL
013D D30E                         OUT     DAZ0    ;SO TURN IT ON
013F 21BC8A                       LXI     H,8ABCH ;DISPLAY DESCRIPTION
0142 D1                           POP     D       ;RESTORE REGISTERS
0143 F1                           POP     PSW     ;
0144 C9                           RET             ;ALL DONE

                          ;
                          ;ROUTINE PAG
                          ;       CLEAR THE SCREEN
                          ;          OR CLEAR 2K OF MEMORY COMMENCING
                          ;         AT THE ADDRESS IN 'RBUF'.
                          ;
0145 F5                   PAGE:   PUSH    PSW     ;SAVE THE USER WORLD
0146 D5                           PUSH    D       ;
0147 E5                           PUSH    H       ;
0148 2AA904                       LHLD    RBUF    ;STARTING ADDRESS
014B 3E08                         MVI     A,08H   ;(BUFFER LENGTH/256)
014D 84                           ADD     H       ;HIGH BYTE OF LAST ADDR +1
014E 5D                           MOV     E,L     ;NEED A ZERO FOR MEMORY FILL
014F 73                   P000:   MOV     M,E     ;ZAP THAT BYTE
0150 23                           INX     H       ;NEXT BYTE
0151 BC                           CMP     H       ;DONE YET?
0152 C24F01                       JNZ     P000    ;NO.  KEEP TRUCKING
0155 E1                           POP     H       ;RESTORE THE USER
0156 D1                           POP     D       ;
0157 F1                           POP     PSW     ;
0158 C9                           RET             ;ALL DONE

                          ;
                          ;ROUTINE CURSOR
                          ;       POSITION THE CURSOR AT X,Y IN H,L
                          ;          OR CONVERT THE COORDINATES IN H,L FROM
                          ;             STANDARD COORDINATES (0-255 ON EACH AXIS)
                          ;             TO THE COORDINATES CURRENTLY IN USE BY THE
                          ;             DAZZLER.
                          ;
0159 F5                   CURSOR: PUSH    PSW     ;SAVE THE WORLD
015A E5                           PUSH    H       ;  OR AT LEAST PART
015B CD6401                       CALL    CU000   ;CONVERT TO COORD IN USE
015E 22AC04                       SHLD    YPOS    ;AND SAVE FOR OTHER PEOPLE
0161 E1                           POP     H       ;RESTORE THE WORLD
0162 F1                           POP     PSW     ;
0163 C9                           RET             ;ALL DONE

                          ;INTERNAL SUBROUTINE CU000
                          ;       CONVERT THE X,Y COORDINATE PAIR IN H,L TO
```

Listing 1 continued:

```
                        ;       THE COORDINATE SYSTEM CURRENTLY IN USE
                        ;
                        ;       USES REGISTERS A, H, AND L
                        ;
0164 3AAF04     CU000:  LDA     MODE    ;WHICH MODE?
0167 F5         CU001:  PUSH    PSW     ;WILL NEED IT LATER
0168 AF                 XRA     A       ;MOVE H TO A WITH CY CLEAR
0169 B4                 ORA     H
016A 1F                 RAR             ;DIVIDE BY 2
016B 67                 MOV     H,A     ; AND SAVE IT
016C AF                 XRA     A       ;DO THE SAME FOR Y
016D B5                 ORA     L
016E 1F                 RAR
016F 6F                 MOV     L,A     ;ALL DONE IF 128 BY 128
0170 F1                 POP     PSW     ;WHAT MODE ARE WE IN?
0171 3C                 INR     A       ; 128 BY 128?
0172 F0                 RP              ;YES, ALL DONE
0173 AF                 XRA     A       ;64 BY 64, PRETEND 128
0174 C36701             JMP     CU001   ; AND DIVIDE AGAIN

                        ;ROUTINE DOT
                        ;       DISPLAY THE POINT AT THE CURSOR POSITION
                        ;
                        ;BLOCK #1:  ADDRESS CALCULATION FROM Y POSITION
                        ;
0177 F5         DOT:    PUSH    PSW     ;SAVE THE WORLD
0178 C5                 PUSH    B       ;
0179 D5                 PUSH    D       ;
017A E5                 PUSH    H       ;
017B 3AAF04             LDA     MODE    ;GET THE DISPLAY MODE BYTE
017E 4F                 MOV     C,A     ;00=MAXR,80=MAXC,01=R128,81=R64
017F 3AAC04             LDA     YPOS    ;Y ADDR IS FIRST
0182 0C                 INR     C       ;WHICH RESOLUTION
0183 FA8801             JM      D101    ;64 BY 64
0186 F5                 PUSH    PSW     ;SAVE FOR BIT MASK TIME
0187 1F                 RAR             ;DIVIDE BY 2
0188 2F         D101:   CMA             ;CONVERT TO LINE NUMBER
0189 57                 MOV     D,A     ;SAVE A COPY
018A E61F               ANI     1FH     ;EACH QUADRANT IS 32 HIGH
018C 6F                 MOV     L,A     ;MULT LINE # BY BYTES/LINE
018D 7A                 MOV     A,D     ;BUT FIRST CORRECT FOR QUADRANT
018E E620               ANI     20H     ;WHICH ARE WE IN?
0190 CA9701             JZ      D102    ;1ST OR 2ND, NO CORRECT REQ
0193 3E40               MVI     A,40H   ;MOVE DOWN TO 3RD OR 4TH
0195 85                 ADD     L       ;WHICH IS 1K AFTER SHIFTING
0196 6F                 MOV     L,A     ;END OF QUADRANT CORRECTION
0197 2600       D102:   MVI     H,00H   ;READY TO MULTIPLY BY 16
0199 29                 DAD     H       ;TIMES 2
019A 29                 DAD     H       ;TIMES 2
019B 29                 DAD     H       ;TIMES 2
019C 29                 DAD     H       ;TIMES 2 = TIMES 16
019D EB                 XCHG            ;MAKE ROOM FOR BASE ADDRESS
019E 2AA904             LHLD    RBUF    ;WHERE DO WE START?
01A1 19                 DAD     D       ;H,L IS FIRST BYTE ON LINE
                        ;
                        ;BLOCK #2:  ADDRESS CALCULATION FROM X POSITION
                        ;
01A2 3AAD04             LDA     XPOS    ;GET X CURSOR
01A5 0C                 INR     C       ;WHICH RESOLUTION?
01A6 FAAB01             JM      D201    ;64 BY 64
01A9 F5                 PUSH    PSW     ;128 BY 128, SAVE FOR BLOCK #3
01AA 1F                 RAR             ;DIVIDE BY 2
01AB 57         D201:   MOV     D,A     ;SAVE FOR LATER
01AC E620               ANI     20H     ;CHECK QUADRANT
01AE CAB501             JZ      D202    ;QUAD 1 OR 3, NO CORRECTION
01B1 3E02               MVI     A,02H   ;QUAD 2 OR 4, ADD 512
01B3 84                 ADD     H       ; TO GET CORRECT ADDRESS
01B4 67                 MOV     H,A     ;QUADRANT CORRECTION COMPLETE
01B5 7A         D202:   MOV     A,D     ;X CURSOR (NORM 64)
01B6 E61E               ANI     1EH     ;32 TO A QUADRANT
01B8 0F                 RRC             ;2 POINTS PER BYTE
01B9 5F                 MOV     E,A     ;  ADD TO LINE ADDRESS
01BA 7A                 MOV     A,D     ;  BUT SAVE COPY OF 64 VALUE
01BB 1600               MVI     D,00H   ;CLEAR D FOR A DOUBLE ADD
01BD 19                 DAD     D       ;NOW HAVE THE BYTE ADDRESSED
                        ;
                        ;BLOCK #3:  BIT MASK GENERATION
                        ;
01BE 1F                 RAR             ;EVEN OR ODD?
01BF 3E0F               MVI     A,0FH   ;ASSUME EVEN
01C1 D2C501             JNC     D301    ;CORRECT ASSUMPTION
01C4 2F                 CMA             ;WAS ODD, SWITCH HALVES
01C5 0C         D301:   INR     C       ;128 BY 128?
01C6 FAE201             JM      D400    ;64 BY 64!
01C9 57                 MOV     D,A     ;SAVE MASK
01CA F1                 POP     PSW     ;XPOS
01CB E601               ANI     01H     ;EVEN OR ODD?
01CD 5F                 MOV     E,A     ;SAVE LOW BIT
01CE F1                 POP     PSW     ;YPOS
01CF 1F                 RAR             ;EVEN OR ODD?
01D0 3E33               MVI     A,33H   ;ASSUME ODD
01D2 DAD601             JC      D302    ;VALID ASSUMPTION
01D5 2F                 CMA             ;NO, EVEN
01D6 A2         D302:   ANA     D       ;DOWN TO 2 BITS
01D7 1D                 DCR     E       ;FINALLY DO X
01D8 CAE001             JZ      D303    ;X WAS ODD
01DB E655               ANI     55H     ;X WAS EVEN
01DD C3E201             JMP     D400    ;
01E0 E6AA       D303:   ANI     0AAH    ;X WAS ODD
                        ;
                        ;BLOCK #4:  REFRESH MEMORY MODIFICATION
                        ;
01E2 57         D400:   MOV     D,A     ;SAVE THE BIT MASK
01E3 2F                 CMA             ;FIRST ERASE ORIGINAL CONTENTS
01E4 A6                 ANA     M       ;
01E5 5F                 MOV     E,A     ;SAVE CLEANED VERSION
01E6 3AAE04             LDA     COLOR   ;GET COLOR DESIRED
01E9 A2                 ANA     D       ;DOWN TO DESIRED POINT
01EA B3                 ORA     E       ;ADD TO ORIGINAL CONTENTS
01EB 77                 MOV     M,A     ;AND STUFF INTO MEMORY
01EC E1         D402:   POP     H       ;RESTORE THE WORLD
01ED D1                 POP     D       ; NOTE: THIS SERIES IS
01EE C1                 POP     B       ;   ALSO USED BY LINE
01EF F1                 POP     PSW     ;   AND CHAR.
01F0 C9                 RET             ;ALL DONE
                        ;
                        ;
```

Listing 1 continued on page 168

Listing 1 continued from page 85:

```
                        ;ROUTINE LINE
                        ;       GENERATE THE LINE FROM THE CURRENT CURSOR
                        ;          POSITION TO THE POINT X,Y IN H,L.
                        ;       USES DOT TO ACTUALLY DISPLAY THE POINTS.
                        ;
                        ;BLOCK 1:  PRELIMINARIES
                        ;
                        ;   1.1--SECTOR DETERMINATION
                        ;
01F1 F5                 LINE:   PUSH    PSW     ;SAVE THE WORLD
01F2 C5                         PUSH    B       ;   NOTE: ORDER IS SET BY
01F3 D5                         PUSH    D       ;   RESTORE IN DOT
01F4 E5                         PUSH    H       ;
01F5 CD6401                     CALL    CU000   ;COORDINATES NEED CHANGING
01F8 3AAD04                     LDA     XPOS    ;GET CURRENT CURSOR POSITION
01FB BC                         CMP     H       ;WHICH IS BIGGER?
01FC DA0502                     JC      L100    ;XF
01FF 94                         SUB     H       ;NEED A-H
0200 0600                       MVI     B,00H   ;SET SECTOR CODE TO ZERO
0202 C30A02                     JMP     L101    ;AND CONTINUE
0205 2F                 L100:   CMA             ;NEED H-A
0206 3C                         INR     A       ;  WHICH REQUIRES 2'S COMP
0207 84                         ADD     H       ;  AND AN ADD
0208 0604                       MVI     B,04H   ;SECTOR CODE GETS 4
020A 57                 L101:   MOV     D,A     ;XP GOES IN D
020B 3AAC04                     LDA     YPOS    ;DO THE SAME FOR Y
020E BD                         CMP     L       ;WHICH IS LARGER
020F DA1702                     JC      L102    ;YF IS
0212 95                         SUB     L       ;YC IS
0213 5F                         MOV     E,A     ;SAVE IT
0214 C31F02                     JMP     L103    ;AND CONTINUE
0217 2F                 L102:   CMA             ;AGAIN, GET 2'S COMPLIMENT
0218 3C                         INR     A       ;
0219 85                         ADD     L       ;  TO FIND YF-YC
021A 5F                         MOV     E,A     ;  AND SAVE IT
021B 3E02                       MVI     A,02H   ;INCR SECTOR CODE BY 2
021D 80                         ADD     B       ;
021E 47                         MOV     B,A     ;NEW SECTOR VALUE
021F 7A                 L103:   MOV     A,D     ;IS XP < YP?
0220 BB                         CMP     E       ;IF SO THEY NEED EXCHANGING
0221 D22702                     JNC     L104    ;  OK AS THEY ARE
0224 53                         MOV     D,E     ;XP = YP
0225 5F                         MOV     E,A     ;  AND YP = OLD XP
0226 04                         INR     B       ;AND SECTOR CODE GETS ONE MORE
                        ;
                        ;   1.2--PARAMETER INITIALIZATION
                        ;
0227 2E00               L104:   MVI     L,00H   ;XT = 0
0229 62                         MOV     H,D     ;XP
022A E5                         PUSH    H       ;XP, XT
022B 65                         MOV     H,L     ;0,0
022C E5                         PUSH    H       ;TA = 0
022D 6B                         MOV     L,E     ;H,L = YP
022E 22A104                     SHLD    DY      ;DY = +YP
0231 7A                         MOV     A,D     ;DETERMINE DX
0232 2F                         CMA             ;  WHICH IS 2'S COMPLIMENT
0233 6F                         MOV     L,A     ;  OF XP
0234 26FF                       MVI     H,0FFH  ;  I.E. DX = -XP
0236 23                         INX     H       ;
0237 22A304                     SHLD    DX      ;SAVE FOR LOOOP
023A 37                         STC             ;T0 = 1/2 DX
023B 7C                         MOV     A,H     ;ARITH SHIFT RIGHT
023C 1F                         RAR             ;  OF H,L
023D 67                         MOV     H,A     ;HIGH  BYTE DONE
023E 7D                         MOV     A,L     ;NOW DO LOW BYTE
023F 1F                         RAR             ;
0240 6F                         MOV     L,A     ;ALL DONE
0241 E5                         PUSH    H       ;SAVE T0
                        ;
                        ;   1.3--SET UP COORDINATE TRANSFORMATION TABLE
                        ;
0242 21BD03                     LXI     H,MXT   ;CALCULATE CORRECT MOVES
0245 78                         MOV     A,B     ;OFFSET INTO TABLE
0246 07                         RLC             ;EACH ENTRY IS FOUR BYTES
0247 07                         RLC             ;
0248 5F                         MOV     E,A     ;ADD TO BASE ADDRESS
0249 1600                       MVI     D,0
024B 19                         DAD     D       ;H,L IS NOW ADDRESS OF M0X
024C 5E                         MOV     E,M     ;GET M0X
024D 23                         INX     H       ;AIM AT M0Y
024E 56                         MOV     D,M     ;AND GET IT TOO
024F EB                         XCHG            ;SHIFT TO H,L
0250 22A504                     SHLD    M0X     ;AND STORE IN MOVE ZERO
0253 EB                         XCHG            ;NOW GET 'ONE' MOVE
0254 23                         INX     H       ;WHICH ARE THE NEXT 2 ENTRIES
0255 5E                         MOV     E,M     ;M1X
0256 23                         INX     H       ;
0257 56                         MOV     D,M     ;M1Y
0258 EB                         XCHG            ;GET SET
0259 22A704                     SHLD    M1X     ;  AND STORE
                        ;
                        ;BLOCK #2:  THE ACTUAL LINE GENERATION LOOP
                        ;
                        ;   2.1--DISPLAY THE CURRENT POINT
                        ;
025C CD7701             L200:   CALL    DOT     ;DISPLAY THE CURRENT POINT
                        ;
                        ;   2.2--TEST FOR DONE
                        ;
025F C1                         POP     B       ;B,C=T0
0260 D1                         POP     D       ;D,E = TA
0261 E1                         POP     H       ;H,L = XP, XT
0262 7D                         MOV     A,L     ;XT
0263 BC                         CMP     H       ;XP
0264 D2EC01                     JNC     D402    ;ALL DONE, GO RESTORE
0267 2C                         INR     L       ;XT = XT + 1
0268 E5                         PUSH    H       ;SAVE FOR NEXT ITERATION
                        ;
                        ;   2.3--DETERMINE NEXT MOVE
                        ;
0269 2AA104                     LHLD    DY      ;GET DY
026C 19                         DAD     D       ;TA = TA + DY
026D E5                         PUSH    H       ;SAVE FOR NEXT ITERATION
026E 09                         DAD     B       ;TA + T0
026F DA7902                     JC      L240    ;IF POSITIVE
                        ;
                        ;   2.4--MAKE THE REQUIRED MOVE
                        ;
```

Listing 1 continued on page 170

Listing 1 continued:

```
0272 C5         L242:   PUSH    B        ;T0 UNCHANGED WITH MOVE ZERO
0273 2AA504             LHLD    M0X      ;M0X IN L; M0Y IN H
0276 C38102             JMP     L241     ;  GO MOVE
0279 2AA304     L240:   LHLD    DX       ;MOVE ONE INCREMENTS T0
027C 09                 DAD     B        ;T0 = T0 + DX
027D E5                 PUSH    H        ;SAVE FOR NEXT ITERATION
027E 2AA704             LHLD    MIX      ;MIX IN L; MIY IN H
0281 EB         L241:   XCHG             ;MAKE ROOM FOR AN ADDRESS
0282 21AC04             LXI     H,YPOS   ;UPDATE Y FIRST
0285 7A                 MOV     A,D      ;M?Y
0286 86                 ADD     M        ;  IS ADDED TO YPOS
0287 77                 MOV     M,A      ;NEW YPOS
0288 23                 INX     H        ;DO THE SAME FOR XPOS
0289 7B                 MOV     A,E      ;
028A 86                 ADD     M        ;
028B 77                 MOV     M,A      ;
028C C35C02             JMP     L200     ;END OF LINE GENERATION LOOP
                ;
                ;ROUTINE CHAR
                ;       GENERATE THE ASCI CHARACTER IN REGISTER A.
                ;       CHARACTERS ARE BASED ON A VARIABLE WIDTH
                ;          4 BY 5 MATRIX.
                ;       THE CURSOR DEFINES THE LOWER LEFT CORNER
                ;          OF THE DOT MATRIX.
                ;       CURSOR IS MOVED TO THE NEXT CHARACTER POSITION.
                ;       LOWER CASE IS CONVERTED TO UPPER CASE.
                ;       PARITY IS IGNORED.
                ;       THE FOLLOWING CONTROL CHARACTERS ARE RECOGNIZED:
                ;MNEMONIC ASCII HEX     FUNCTION
                ;MAXR    NUL     00     DISPLAY MODE - 128 BY 128 COLOR
                ;MAXC    SOH     01     DISPLAY MODE - 64 BY 64 COLOR
                ;R128    STX     02     DISPLAY MODE - 128 BY 128 COLOR
                ;R4     ETX     03     DISPLAY MODE - 64 BY 64 COLOR
                ;
                ;BS      BS      08     BACKSPACE:  XPOS=XPOS-6
                ;HT      HT      09     HOR. TAB:  XPOS=(XPOS+32)MOD 32
                ;LF      LF      0A     LINE FEED:  YPOS=YPOS-8
                ;VT      VT      0B     VERT. TAB:  YPOS=((YPOS-32) MOD
32)-6
                ;FF      FF      0C     FORM FEED:  XPOS=0, YPOS=MAX-6
                ;CR      CR      0D     CAR. RET.:  XPOS=0
                ;
                ;BLK     DLE     10     BLACK (ERASE)
                ;RED     DC1     11     RED
                ;BLU     DC2     12     BLUE
                ;MAG     DC3     13     MAGENTA
                ;GRN     DC4     14     GREEN
                ;YEL     NAK     15     YELLOW
                ;CYN     SYN     16     CYAN
                ;WHI     ETB     17     WHITE
                ;
                ;
                ;BLOCK 1:  CHARACTER TYPE DETERMINATION
                ;
028F F5         CHAR:   PUSH    PSW      ;SAVE THE WORLD
0290 C5                 PUSH    B        ;NOTE: ORDER IS SET BY
0291 D5                 PUSH    D        ;   RESTORE IN DOT
0292 E5                 PUSH    H        ;
0293 01EC01             LXI     B,D402   ;FAKE A CALL FROM THE
0296 C5                 PUSH    B        ;  REGISTER RESTORE SEQUENCE
0297 E67F               ANI     7FH      ;CLEAR PARITY BIT
0299 FE20               CPI     20H      ;COMPARE TO A BLANK
029B DA1C03             JC      C500     ;  CONTROL CHARACTER
029E FE60               CPI     60H      ;COMPARE TO ACCENT GRAVE
02A0 DAA502             JC      C100     ;  UPPER CASE
02A3 E65F               ANI     5FH      ;CONVERT LOWER CASE TO UPPER
02A5 2AAC04     C100:   LHLD    YPOS     ;GET CURRENT CURSOR POSITION
02A8 EB                 XCHG             ;  BUT IN D,E
                ;
                ;BLOCK 2:  CALCULATE THE CHARACTER MATRIX ADDRESS
                ;       A = ASCII CHARACTER     D,E = XPOS, YPOS
                ;
02A9 21DD03             LXI     H,CHRX   ;BASE ADDRESS OF CHAR TABLE
02AC D620               SUI     20H      ;ZEROTH ENTRY IN TABLE IS BLANK
02AE 4F                 MOV     C,A      ;3 BYTES PER ENTRY
02AF 0600               MVI     B,00H    ;  SO MULTIPLY OFFSET BY3
02B1 09                 DAD     B        ;ONCE
02B2 09                 DAD     B        ;  TWICE
02B3 09                 DAD     B        ;    THRICE
02B4 7E                 MOV     A,M      ;GET BYTE 0 WITH FLAGS
02B5 E603               ANI     03H      ;ISOLATE WIDTH FIELD
02B7 FE03               CPI     03H      ;FIVE WIDE?
02B9 CCF302             CZ      C400     ;YES, TAKE CARE OF IT
02BC 42                 MOV     B,D      ;SAVE STARTING XPOS
02BD C603               ADI     03H      ;WIDTH OF CHAR + 1
02BF 82                 ADD     D        ;XPOS OF NEXT CHARACTER
02C0 57                 MOV     D,A      ;D,E IS NEXT CHAR POSITION
02C1 D5                 PUSH    D        ;SAVE UNTIL DONE
02C2 50                 MOV     D,B      ;RESTORE CURRENT POSITION
02C3 7E                 MOV     A,M      ;ONE LAST FLAG TO TEST
02C4 07                 RLC              ;IS THIS A DESCENDING CHAR
02C5 D2CA02             JNC     C300     ;  NO.  GO GENERATE IT
02C8 1D                 DCR     E        ;  YES.  DOWN TWO ON Y
02C9 1D                 DCR     E        ;
                ;
                ;BLOCK 3:  GENERATE THE ACTUAL CHARACTER
                ;       A = MASK FOR BOTTOM ROW
                ;       D,E = XPOS, YPOS
                ;       H,L = ADDRESS OF FIRST BYTE OF CHAR TABLE ENTRY
                ;
02CA EB         C300:   XCHG             ;GET REGISTERS IN POSITION
02CB CDE102             CALL    C310     ;DO BOTTOM ROW OF CHAR
02CE CDDF02             CALL    C305     ;SECOND ROW
02D1 CDE102             CALL    C310     ;THIRD ROW
02D4 CDDF02             CALL    C305     ;FOURTH ROW
02D7 CDE102             CALL    C310     ;AND TOP ROW
02DA E1                 POP     H        ;RETRIEVE PRECALCULATED CURSOR
02DB 22AC04             SHLD    YPOS     ;AND UPDATE CURSOR
02DE C9                 RET              ;ALL DONE
02DF 13         C305:   INX     D        ;NEXT BYTE IN TABLE
02E0 1A                 LDAX    D        ;  GOES IN A
02E1 0604       C310:   MVI     B,04H    ;COLUMNS PER ROW
02E3 E5                 PUSH    H        ;SAVE STARTING POSITION
02E4 07         C311:   RLC              ;SHOULD POINT BE ON?
02E5 22AC04             SHLD    YPOS     ;UPDATE CURSOR
02E8 DC7701             CC      DOT      ;PUT UP THE POINT IF REQUIRED
02EB 24                 INR     H        ;NEXT X
02EC 05                 DCR     B        ;COUNT DOWN
```

Listing 1 continued on page 172

Listing 1 continued:

```
02ED C2E402          JNZ     C311      ;MORE TO GO
02F0 E1              POP     H         ;RESTORE X
02F1 2C              INR     L         ;UP ONE ON Y
02F2 C9              RET               ;END OF LOCAL SUBROUTINE
                ;
                ;BLOCK 4:  GENERATE FIRST COLUMN OF 5 WIDE CHARACTERS
                ;        A = 03H         C = CHAR - 32
                ;        D,E = XPOS, YPOS
                ;        H,L = ADDR OF 1ST BYTE OF CHAR TABLE ENTRY
                ;
02F3 7E         C400:  MOV     A,M       ;GET FLAGS
02F4 D5                PUSH    D         ;SAVE STARTING CURSOR
02F5 E604              ANI     04H       ;AUXILIARY LOOKUP REQUIRED
02F7 C21003            JNZ     C410      ;  YES.   GO DO IT
02FA 2F                CMA               ;1ST COLUMN ALL ONES (M & W)
02FB 0605       C411:  MVI     B,05H     ;5 POINTS TO A COLUMN
02FD 07         C401:  RLC               ;SHOULD THE POINT BE ON?
02FE EB                XCHG              ;GET X,Y IN H,L
02FF 22AC04            SHLD    YPOS      ;CURRENT CURSOR POSITION
0302 DC7701            CC      DOT       ;DISPLAY AS REQUIRED
0305 EB                XCHG              ;BACK TO NORMALCY
0306 1C                INR     E         ;NEXT YPOS
0307 05                DCR     B         ;TEST FOR DONE
0308 C2FD02            JNZ     C401      ;NOT YET
030B D1                POP     D         ;ORIGINAL CURSOR POSITION
030C 14                INR     D         ;FIX UP TO DO COLUMNS 2-5
030D 3E02              MVI     A,02H     ; AS A 4 WIDE CHAR
030F C9                RET
0310 E5         C410:  PUSH    H         ;SAVE CHAR TABLE ENTRY
0311 219A04            LXI     H,CHRA-3  ;AUXILIARY TABLE ADDR
0314 0600              MVI     B,00H     ;  FOR CHARS #, $, %, AND &
0316 09                DAD     B         ;NOTE: C HAS CHAR - 20H
0317 7E                MOV     A,M       ;GET THE FIRST COLUMN
0318 E1                POP     H         ;AND RESTORE TABLE ENTRY
0319 C3FB02            JMP     C411      ;DISPLAY THE RETRIEVED COLUMN
                ;
                ;BLOCK 5:  CONTROL CHARACTERS
                ;        A = ASCII CONTROL CHARACTER
                ;
031C FE00       C500:  CPI     00H       ;MAXR?
031E CA2603            JZ      C501      ;YES
0321 FE02              CPI     02H       ;R128?
0323 C23003            JNZ     C503      ;NO
0326 067F       C501:  MVI     B,7FH     ;128 BY 128 WHITE
0328 0F         C502:  RRC               ;CONVERT TO MODE BYTE
0329 32AF04            STA     MODE      ; AND SAVE NEW MODE
032C 78                MOV     A,B       ;GET DESIRED DAZZLER MODE
032D D30F              OUT     DAZ1      ;AND TELL THE DAZZLER
032F C9                RET
0330 FE01       C503:  CPI     01H       ;MAXC?
0332 CA3A03            JZ      C504      ;YES
0335 FE03              CPI     03H       ;R64?
0337 C23F03            JNZ     C505      ;NO
033A 063F       C504:  MVI     B,3FH     ;64 BY 64 FULL COLOR
033C C32803            JMP     C502      ;REST IS SAME AS 128
033F FE0A       C505:  CPI     0AH       ;LINE FEED?
0341 C24C03            JNZ     C506      ;NO
0344 21AC04            LXI     H,YPOS    ;YPOS = YPOS - 8
0347 7E                MOV     A,M
0348 D608              SUI     08H
034A 77                MOV     M,A
034B C9                RET
034C FE0D       C506:  CPI     0DH       ;CARRIAGE RETURN?
034E C25603            JNZ     C508      ;NO
0351 AF         C507:  XRA     A         ;XPOS = 0
0352 32AD04            STA     XPOS
0355 C9                RET
0356 FE0C       C508:  CPI     0CH       ;FORM FEED?
0358 C26D03            JNZ     C510      ;NO
035B 3AAF04            LDA     MODE      ;WHAT RESOLUTION?
035E 47                MOV     B,A
035F 3E7A              MVI     A,7AH     ;ASSUME 128 BY 128
0361 04                INR     B         ;IS IT?
0362 F26703            JP      C509      ;SURE IS
0365 3E3A              MVI     A,3AH     ;BAD ASSUMPTION
0367 32AC04     C509:  STA     YPOS      ;SO MUCH FOR YPOS
036A C35103            JMP     C507      ;TAKE CARE OF XPOS
036D FE10       C510:  CPI     10H       ;BLACK?
036F CA8303            JZ      C512      ;SURE IS
0372 D8                RC                ;NOT EVEN A COLOR
0373 FE18              CPI     18H       ;ABOVE WHITE?
0375 D0                RNC               ;YES.   FORGET IT
0376 F608              ORI     08H       ;USE BRIGHT COLORS
0378 47                MOV     B,A       ;SAVE WHERE SAFE
0379 3AAF04            LDA     MODE      ;128 BY 128 CAN ONLY
037C 3C                INR     A         ;  BE WHITE
037D FA8203            JM      C511      ;OK.   64 BY 64
0380 060F              MVI     B,0FH     ;FORCE IT TO BE WHITE
0382 78         C511:  MOV     A,B       ;MAKE BOTH HALFS
0383 E60F       C512:  ANI     0FH       ;  THE SAME
0385 47                MOV     B,A       ;SAVE ONE COPY
0386 0F                RRC
0387 0F                RRC
0388 0F                RRC
0389 0F                RRC
038A B0                ORA     B         ;COMPLETE COLOR BYTE
038B 32AE04            STA     COLOR
038E C9                RET

                ;
                ;ROUTINE ANIMAT
                ;        SWAP DISPLAY BUFFERS
                ;        BUFFER CURRENTLY BEING FILLED IS DISPLAYED
                ;        BUFFER INDICATED BY ANIM IS FILLED
                ;           ANIM=0   STARTS FILLING RBUF+2K
                ;           ANIM=-1  STARTS FILLING RBUF-2K
                ;
038F F5         ANIMAT: PUSH   PSW       ;SAVE REGISTERS USED
0390 C5                PUSH    B         ;
0391 E5                PUSH    H         ;D IS NOT TOUCHED
0392 DB0E       AN002: IN      DAZ0      ;VERTICAL BLANKING ON?
0394 E640              ANI     40H       ;IF SO
0396 CA9203            JZ      AN002     ;  WAIT FOR NEXT ONE
0399 21AB04            LXI     H,ANIM    ;ADDRESS OF IN USE FLAG
039C 7E                MOV     A,M       ;  BUFFER IN USE
039D 2F                CMA               ;SET FOR NEXT TRY
039E 77                MOV     M,A       ;  AND SAVE FOR NEXT TIME
039F 47                MOV     B,A       ;  ALSO SAVE FOR LATER
```

Listing 1 continued on page 174

Listing 1 continued:

```
03A0 2B                   DCX     H           ;NOW LOOKING AT HIGH BYTE
03A1 7E                   MOV     A,M         ;   OF RBUF
03A2 37                   STC                 ;FUTURE GO BIT
03A3 1F                   RAR                 ;ALL) SET TO DISPLAY
03A4 4F                   MOV     C,A         ;SAVE WHILE WAIT FOR VERT BLANK
03A5 DB0E        AN000:   IN      DAZ0        ;WHAT'S THE DAZZLER DOING?
03A7 E640                 ANI     40H         ;VERTICAL) BLANKING ON?
03A9 C2A503               JNZ     AN000       ;  NOT YET.  GIVE IT TIME
03AC 79                   MOV     A,C         ;NEW BUFFER WITH NO FLICKER
03AD D30E                 OUT     DAZ0        ;  IS NOW DISPLAYING
03AF 7E                   MOV     A,M         ;CALCULATE NEW FILL
03B0 C608                 ADI     08H         ;  ASSUME BUFFER 0
03B2 04                   INR     B           ;WHICH IS IT?
03B3 CAB803               JZ      AN001       ;ZERO IT IS
03B6 D610                 SUI     10H         ;THE SECOND, SO COME BACK
03B8 77          AN001:   MOV     M,A         ;UPDATE THE FILL ADDRESS
03B9 E1                   POP     H           ;AND RESTORE THE WORLD
03BA C1                   POP     B           ;
03BB F1                   POP     PSW         ;
03BC C9                   RET                 ;
                 ;
                 ;                    END OF EXECUTABLE PROGRAM CODE
                 ;
                 ;                    ******************************
                 ;
                 ;                            LOOKUP TABLES
                 ;
                 ;
                 ;MOVE TABLE FOR THE LINE GENERATOR
                 ;
03BD FF00FFFF    MXT:     DB      0FFH,000H,0FFH,0FFH    ;SECTOR 5
03C1 00FFFFFF             DB      000H,0FFH,0FFH,0FFH    ;SECTOR 6
03C5 FF00FF01             DB      0FFH,000H,0FFH,001H    ;SECTOR 4
03C9 0001FF01             DB      000H,001H,0FFH,001H    ;SECTOR 3
03CD 010001FF             DB      001H,000H,001H,0FFH    ;SECTOR 8
03D1 00FF01FF             DB      000H,0FFH,001H,0FFH    ;SECTOR 7
03D5 01000101             DB      001H,000H,001H,001H    ;SECTOR 1
03D9 00010101             DB      000H,001H,001H,001H    ;SECTOR 2
                 ;
                 ;
                 ;CHARACTER MATRIX TABLE
                 ;        EACH ENTRY IS 3 BYTES
                 ;        BIT >>    7    6    5    4    3    2    1    0
                 ;BYTE    0         U2   Q    R    S    T    V5   V2   V1
                 ;BYTE    1         M    N    O    P    I    J    K    L
                 ;BYTE    2         E    F    G    H    A    B    C    D
                 ;
                 ;        A  B  C  D                  FLAGS
                 ;        E  F  G  H        U2:  DESCENDERS, MOVE DOWN 2
                 ;        I  J  K  L        V2, V1:  WIDTH OF CHARACTER - 2
                 ;        M  N  O  P        V5:  FOR FIVE WIDE FIGURES
                 ;        Q  R  S  T             0 - FIRST COLUMN ALL ONES
                 ;                               1 - FIRST COLUMN FROM CHRA
                 ;        REPRESENTS 2ND THRU 5TH COLUMNS
                 ;           OF FIVE COLUMN WIDE CHARACTERS
                 ;
                 CHRX:
D03DD 0200004008   DB     02H,00H,00H,40H,08H,88H,01H,00H,0AAH       ;  !  "
03E6 57FAFA775E    DB     57H,0FAH,0FAH,77H,5EH,4EH,1FH,0B4H,0A9H    ;#  $  %
03EF 6F25CC0000    DB     6FH,25H,0CCH,00H,00H,88H,20H,88H,84H       ;&  ' (
03F8 40444801A4    DB     40H,44H,48H,01H,0A4H,0A0H,01H,4EH,40H      ;)  *  +
0401 808800010E    DB     80H,88H,00H,01H,0EH,00H,40H,00H,00H        ;,  -  .
040A 42421032DB    DB     42H,42H,10H,32H,0DBH,96H,71H,44H,4CH       ;/  0  1
0413 7A42967216    DB     7AH,42H,96H,72H,16H,1EH,0AH,1FH,99H        ;2  3  4
041C 721E8F329E    DB     72H,1EH,8FH,32H,9EH,86H,42H,42H,1FH        ;5  6  7
0425 3296967217    DB     32H,96H,96H,72H,17H,96H,40H,08H,00H        ;8  9  :
042E 8088081148    DB     80H,88H,08H,11H,48H,42H,01H,0E0H,0E0H      ;;  <  =
0437 4142482202    DB     41H,42H,48H,22H,02H,96H,62H,0BBH,9FH       ;>  ?  @
0440 4AF996729E    DB     4AH,0F9H,96H,72H,9EH,9EH,32H,98H,96H       ;A  B  C
0449 72999E7A8E    DB     72H,99H,9EH,7AH,8EH,8FH,42H,8EH,8FH        ;D  E  F
0452 329B864A9F    DB     32H,9BH,86H,4AH,9FH,99H,71H,44H,4EH        ;G  H  I
045B 3291114AAC    DB     32H,91H,11H,4AH,0ACH,0A9H,7AH,88H,88H      ;J  K  L
0464 0B55B14A9B    DB     0BH,55H,0B1H,4AH,9BH,0D9H,32H,99H,96H      ;M  N  O
046D 428E9E3AB9    DB     42H,8EH,9EH,3AH,0B9H,96H,4AH,0AEH,9EH      ;P  Q  R
0476 7216871222    DB     72H,16H,87H,12H,22H,2FH,32H,99H,99H        ;S  T  U
047F 3269990BB5    DB     32H,69H,99H,0BH,0B5H,51H,4AH,66H,99H       ;V  W  X
0488 2226997A84    DB     22H,26H,99H,7AH,84H,2FH,60H,88H,8CH        ;Y  Z  [
0491 0212486044    DB     02H,12H,48H,60H,44H,4CH,02H,08H,96H        ;/  ]  ↑
049A F80000        DB     0F8H,00H,00H                               ;←
                 ;AUXILIARY LOOKUP TABLE
                 ;        FIRST COLUMN OF #, $, %, AND &
                 ;
049D 50109860    CHRA:    DB      50H,10H,98H,60H ;# $ % &
                 ;
                 ;                END OF ROMABLE SEGMENT OF PROGRAM
                 ;
                 ;                ************************************
                 ;
                 ;                START OF RAM (VARIABLE) STORAGE AREA
                 ;
                 ;SCRATCH PAD STORAGE FOR THE LINE GENERATOR
                 ;        THESE LOCATIONS MAY BE ALTERED AT ANY TIME A
                 ;           LINE IS NOT ACTUALLY BEING GENERATED
                 ;        VARIABLES MUST BE IN THE ORDER GIVEN.
04A1             DY:      DS      2       ;+YP
04A3             DX:      DS      2       ;- XP
04A5             M0X:     DS      1       ;X INCR FOR A ZERO MOVE
04A6             M0Y:     DS      1       ;Y INCR FOR A ZERO MOE
04A7             M1X:     DS      1       ;X INCR FOR A ONE MOVE
04A8             M1Y:     DS      1       ;Y INCR FOR A ONE MOVE
                 ;
                 ;
                 ;GLOBAL) STORAGE AREA FOR THE GRAPHICS PACKAGE
                 ;        THESE LOCATIONS MUST BE PRESERVED BETWEEN
                 ;           CALLS TO THE GRAPHICS ROUTINES.
                 ;        THEY ARE INITIALIZED BY INITG.
                 ;        VARIABLES MUST BE IN THE ORDER GIVEN.
                 ;
04A9             RBUF:    DS      2       ;REFRESH BUFFER ADDRESS
04AB             ANIM:    DS      1       ;BUFFER IN USE FLAG FOR ANIMATIO
                                           N
04AC             YPOS:    DS      1       ;Y CURSOR VALUE
04AD             XPOS:    DS      1       ;X CURSOR VALUE
04AE             COLOR:   DS      1       ;CURRENT COLOR BYTE
04AF             MODE:    DS      1       ;DISPLAY MODE
04B0                      END
```