Dazzler Graphics Instruction Manual

Five Dollars

CROMEMCO

DAZZLER GRAPHICS

Instruction Manual

CROMEMCO, INC.
280 Bernardo Avenue
Mountain View, CA 94040

Part No. 023-0044 October 1978

CROMEMCO DAZZLER GRAPHICS MANUAL

ABSTRACT

This manual describes the Cromemco Dazzler* graphics software package. This package is available from Cromemco on either a 5" diskette (model DGR-S) or on a 8" diskette (model DGR-L). The graphics package includes functions such as Dazzler initialization, screen erase, line generation, character generation, and point display. Several of these functions are based on a graphics standard proposed by Dr. Vincent Jones. The functions are interfaced to Cromemco Fortran IV and 16K Extended Basic as well as being fully relocatable for linking to assembly language programs. The graphics package also includes demonstration graphics programs in Fortran and Basic that use the Dazzler display.

*Dazzler is a registered trademark of Cromemco.

Table of Contents

Preface

There are two major graphics routines supplied with the DGR-S and DGR-L Dazzler Graphics diskettes. The first of these is actually a series of 9 major subroutines which perform such functions as line and area plotting, character generation, and animation. These subroutines are all included in each of the programs on the DGR disk beginning with the word "GRAPH". GRAPHZ80.REL is a relocatable program module to be used as a complete graphics interface to Z80 assembly language programs. GRAPHFOR.REL is a very similar set of relocatable modules which interfaces to CROMEMCO FORTRAN IV; however, GRAPHFOR has additions to handle the passing of parameters to and from FORTRAN. Finally, the program GRAPHX.COM is a complete interface of the standard subroutines to CROMEMCO 16K BASIC. These programs and how they work are fully described in Part I of this Manual. Chapter 3 of Part I describes the protocol for calling these routines from each of the various languages. Chapter 5 gives a discussion of the various sample programs which are provided to illustrate features of the graphics programs.

The second major graphics routine supplied on the DGR diskettes provides a more interactive method of interface with an emphasis on plotting abilities. This routine is supplied under the name DAZZPLOT.COM and it provides a stand-alone graphics package with full interface to the console keyboard as well as the CROMEMCO Joysticks. Because this program is implemented in a somewhat different way from those of Part I, it is described separately in Part II of this manual along with a description of a sample program.

Those users who do not presently have the CROMEMCO FORTRAN, BASIC, or Assembler disks should not be overly concerned. These disks may be purchased at some time in the future to provide additional graphics capability. In the meantime, DAZZPLOT.COM will provide a stand-alone version with which to gain familiarity with Dazzler Graphics.

PART I - STANDARDIZED DAZZLER GRAPHICS

Chapter 1: INTRODUCTION TO DAZZLER GRAPHICS

A major stumbling block in making good software available to a large number of users in the computer market has been a lack of standardization. Manufacturers and writers of machine programs tend to establish internal standards of software which are incompatible with those of other manufacturers. In the interest of minimizing this problem, this document proposes a graphics interface standard protocol for computers. CROMEMCO has attempted to retain the graphics standard as much as possible in the development of this graphics package. Although there are differences in the way the "GRAPH" programs interface to their respective languages, all have in common the set of seven subroutines originally outlined (Chapter 2) in a largely unchanged form, and all have the CROMEMCO Dazzler as their graphics output device. References in the following chapters to the "proposed graphics standard" refer to this original outline.

In addition, the interfaces to FORTRAN IV and 16K BASIC implemented here have been adopted as a CROMEMCO standard graphics protocol and will be supported by future graphics releases.

This graphics 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, and the functions available and what they do. For example, a request for a red line from the center of the screen to the bottom right corner would always require the 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. For example, a black and white display would draw a white line instead. To allow for possible differences in the hardware 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, either 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 if full color were 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, while if only a black and white display were 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. If necessary, it also defines where in memory the routines are loaded and the locations of their calling vectors. Returning again to the example of drawing a red line, a Z80 assembly language program would use the instruction sequence:

```
LD     A,11H          ;Code for Red
CALL   CHAR$          ;Vector for CHAR
LD     HL,8080H       ;X = 128, Y = 128
CALL   CURSR$         ;Vector for CURSOR
LD     HL,0FF00H      ;X = 255, Y = 0
CALL   LINE$          ;Vector for LINE
```

Similarly, a CROMEMCO BASIC program would read:

```
10 REM - Open the Device Driver
20 OPEN\1\"$DG"
30 REM - Select Color mode and Red
40 PUT\1\"ZR"
50 REM - Position cursor at center of screen
60 PUT\1,128,128\"P"
70 REM - Draw the Line to corner
80 PUT\1,255,0\"L"
```

Chapter 2: THE STANDARD DISPLAY AND ITS FUNCTIONS

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 2-1, 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. Those of you with a mathematical background will recognize this as the first quadrant of the standard Cartesian Coordinate System. Each picture element (pixel) may be black, white, red, green, blue, yellow, cyan, or magenta; that is, it may be any combination of the three primary colors.

The display to be used is programmed to imitate the standard to the best of its ability. To facilitate this, four standard display modes are defined. Mode zero requests the maximum possible resolution while mode one requests the maximum choice of colors. This is to allow for displays which offer a tradeoff between resolution and color such as the CROMEMCO DAZZLER^{tm}. Two additional modes provide the ability to deliberately select larger pixels. Mode two is 128 by 128 resolution and mode three 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, the range of colors available, whether the display is black/white or color, whether individual points can be erased, and if dual-buffered animation is available.

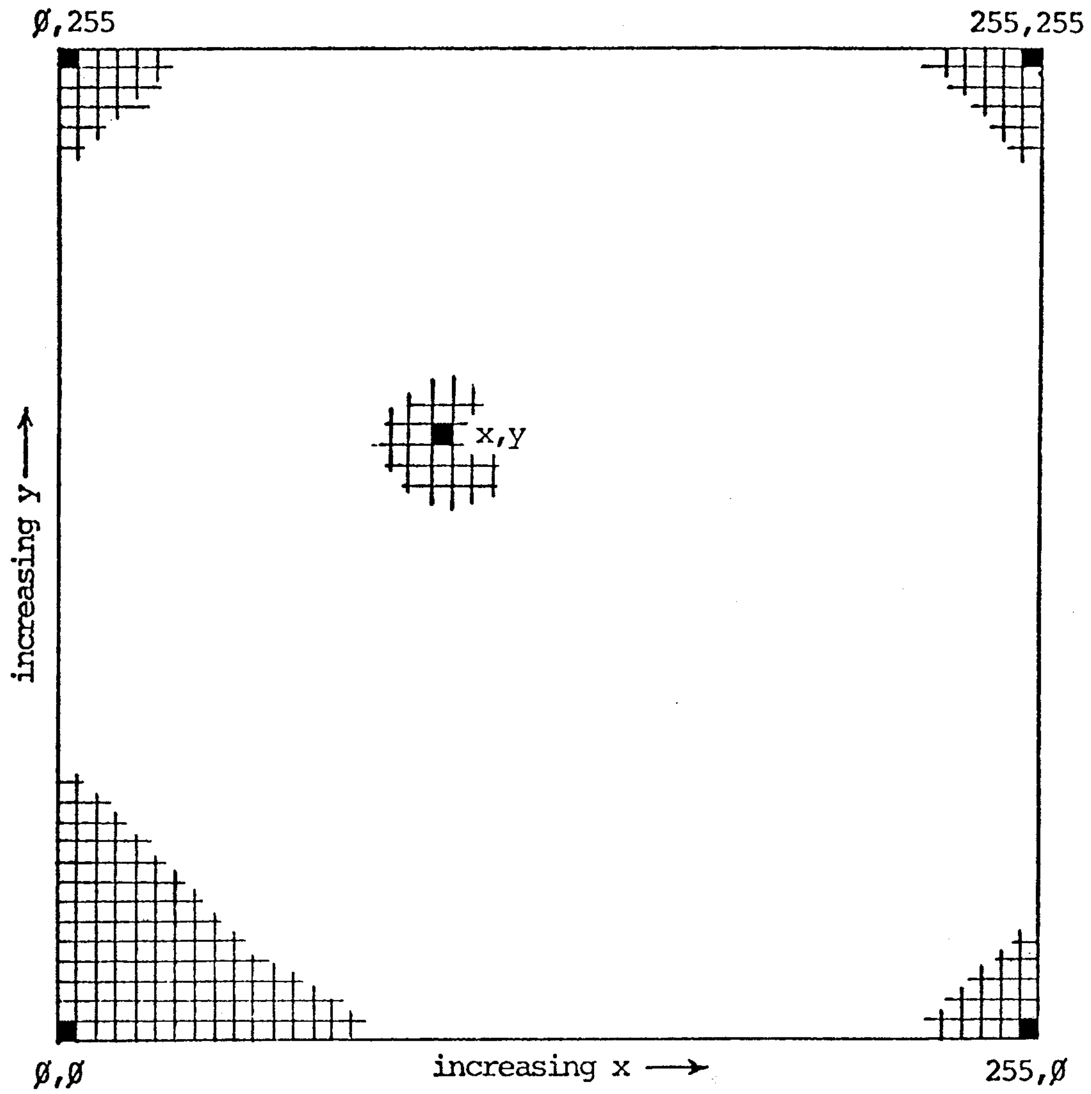


Figure 2-1. The Standard Coordinate System.

Standard Functions

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

PAGE	-Next page, i.e. 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 two 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, animation is disabled, and the display mode is set for maximum resolution (mode 0). Any special options peculiar to the particular display are also disabled so that general purpose programs do not need to know about them to function correctly. Secondly, this routine performs any initialization functions required by the display hardware. For those displays which refresh from program memory, it establishes the refresh buffers. If the display is under program control, it is turned on. Finally, this routine causes the display description variables to be set to the appropriate values. Failure to initialize the display before using any of the other functions may lead to unpredictable results.

* PAGE *

The PAGE function clears the display screen. No other changes are made to the state of the display. In particular, 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). 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. Note that this has no effect on cursor position.

* LINE *

The LINE function generates the line connecting the pixel presently defined by the cursor to the pixel requested. Both endpoints are included in the line. A line of zero length therefore is logically equivalent to a call to DOT. When erasing or otherwise changing the color of a line, care must be exercised as the pixels making up the line from pixel A to pixel B may differ from the pixels used when the line is drawn from pixel B to pixel A. When lines are drawn in various 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 2-1. Control characters not recognized are ignored. Note that form feed only positions the cursor, 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). Following display, the cursor is left positioned at the next character position. No check is made to detect characters off the edge of the screen. The parity bit is ignored. Lower case characters, if not supported, are converted to upper case.

* 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 causes the buffer which is being filled to be displayed and another buffer to be opened 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 erasing individual pixels 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 will do nothing on displays which support neither double buffering nor selective erase. The CROMEMCO DAZZLER fully supports double buffering. To return to normal mode where updates are displayed in real time, it is necessary either to reinitialize with INITG or to use one of the control character calls to CHAR to stop-animation in buffer 1 or buffer 2 (i.e., select buffer 1 or buffer 2 in real-time mode).

MNEMONIC	ASCII	CONTROL CHARACTER	HEX	STANDARD FUNCTION
				Display Mode Selection
MAXR	NUL	CTRL-@	00	Maximum resolution
MAXC	SOH	CTRL-A	01	Maximum colors
R128	STX	CTRL-B	02	128 by 128
R64	ETX	CTRL-C	03	64 by 64
RXXX	EOT	CTRL-D	04	Undefined
				Carriage Control
BS	BS	CTRL-H	08	Backspace (optional)
HT	HT	CTRL-I	09	Horizontal tab (optional)
LF	LF	CTRL-J	0A	Line feed
VT	VT	CTRL-K	0B	Vertical tab (optional)
FF	FF	CTRL-L	0C	Form feed
CR	CR	CTRL-M	0D	Carriage return
				Character Style
SO	SO	CTRL-N	0E	Undefined
SI	SI	CTRL-O	0F	Undefined
				Current Color Selection
BLK	DLE	CTRL-P	10	Black
RED	DC1	CTRL-Q	11	Red
GRN	DC2	CTRL-R	12	Green
YEL	DC3	CTRL-S	13	Yellow
BLU	DC4	CTRL-T	14	Blue
MAG	NAK	CTRL-U	15	Magenta
CYN	SYN	CTRL-V	16	Cyan
WHI	ETB	CTRL-W	17	White
NONE	CAN to VS	CTRL-X to CTRL-_	18 to 1F	Eight optional colors

Table 2-1. Standard Control Character Functions.

Chapter 3: USING THE GRAPHICS PROGRAMS

As described in the Introduction to this Part, there are three implementations of the Dazzler Graphics subroutines: one for each of the languages, BASIC, FORTRAN, and Z80 Assembler. Each of these is described briefly in this chapter along with an exact description of the ways they are called and used.

The subroutines described throughout Part I of this manual comprise a standard Z80 assembly language protocol which has been proposed to encourage maximum software interchange. The programs which have been developed to interface this graphics standard to FORTRAN and BASIC do so leaving the original routines in unchanged form. It is only the passing of parameters and the calling procedure itself which has been simplified. The procedures adopted for FORTRAN IV and 16K BASIC will be supported as far as is possible by any subsequent graphics releases.

Z80 Assembly Protocol

Z80 programmers may write graphics software in standard Zilog Z80 mnemonics into a file with the .Z80 extension. (Use EDIT.COM, supplied on all CROMEMCO disks, to create this file.) The names which must be used in calling the various graphics routines are given in Tables 3-1 and 3-2. In addition there must be a statement somewhere in the .Z80 file of the form:

```
            EXT  INITG$, CHAR$, etc.
```

to declare the routines which were used to be EXTernal. The Z80 program should have a labeled statement at the desired start of the program such as:

```
    START:  LD   SP,1000H
```

and an END statement such as:

```
            END  START
```

An END statement alone is not sufficient as this declares the program a sub-module rather than the main module (refer to the Macro Assembly Manual for more information). Except for the CHAR and PAUSE routines which use the A register, all arguments are passed to the called routine in register pair HL. Except for the INITG routine (which returns display data in HL), the contents of all registers and flags are preserved. It is up to the programmer to set up the HL registers with the correct parameters. The desired routine is then called with an instruction such as:

```
          CALL CHAR$
```

(Either the global names or the standard names may be used in calling the routines.) The routines return with a simple RET instruction. Much of this information along with the parameters required (in some cases, none) is given in Tables 3-1 and 3-2. Generally H contains the X value and L contains the Y value. However, for INITG if H is 0 (regardless of L), the routine will search for the highest 4K block of RAM <u>below</u> <u>CDOS</u> to locate the two picture buffers. This is generally the preferred location because it is well above the typical graphics program. On the other hand, if H contains a non-zero value, the 2 buffers will be put in the first 4K block above the end of the program. When this method is used, the graphics routines (GRAPHZ80.REL) must be LINKed last. Be sure when writing a graphics program to call INITG before any other graphics operations to initialize the picture buffers and other variables. Also be sure to initialize the stack pointer. 1000H is a good value because the graphics program will be linked starting at 1000H.

Once your .Z80 file is completed, it must be assembled using the CROMEMCO Relocatable Assembler (ASMB.COM, provided on the Assembler disk) and linked to the graphics routines (LINK.COM, provided on this disk) with a command of the following form:

```
          LINK  MYPROGRM,GRAPHZ80/G
```

to execute the program directly, or /E to exit to CDOS. If exiting to CDOS, the program may be saved using the CDOS SAVE command (see CDOS manual).

ROUTINE	GLOBAL NAME	PARAMETERS
INITG	INITG$	Returns display description in HL.
PAGE	PAGE$	None
CURSOR	CURSR$	H = X coordinate; L = Y coordinate.
DOT	DOT$	None
LINE	LINE$	H = X end coordinate; L = Y end coordinate.
CHAR	CHAR$	A = ASCII value of character to be printed or control character.
ANIMAT	ANIMT$	None

Table 3-1. Z80 Assembly Language Standard Subroutines.

The parameter passed in HL when calling INITG has already been described. However, if it is desired to make your Z80 program graphics standard compatible, a good value to load into HL prior to calling INITG is 5800H. The picture buffers will then be placed at the end of the current program. The address in HL is replaced by INITG with a two-byte description of the display being used (all other registers and flags are left undisturbed). The format for these bytes is given in Figure 3-1. The available colors and scale factor fields in the H register describe the display when maximum resolution is selected; the same fields in the L register describe the maximum color selection mode.

The available colors field gives the number of colors a point can be written to other than white. If this field is zero, it means 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, then the display supports double buffered animation. If this field is zero, then 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/BW field is self-explanatory. If one, the display is in color; otherwise it is black, grey, and white.

This two-byte description returned in HL is provided only for graphics standard compatibility. Presumably, the programmer already knows the number of available colors, etc. of the Dazzler, and will therefore not need to examine these two bytes in a program unless the program is of a "universal" graphics nature.

H								L							
ANIM	MRCOLS				MRSCLF			COL	MCCOLS				MCSCLF		
D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
MAXRD								MAXCD							

ANIM =	1 - Double buffered animation supported. 0 - Delay to start of vertical blanking.
COL =	1 - Display is in color. 0 - Display is black and white.
MRCOLS	- Colors (gray shades) in MAXR mode.
MCCOLS	- Colors (gray shades) in MAXC mode.
MRSCLF	- $\frac{256}{\text{Display resolution}}$ in MAXR mode.
MCSCLF	- $\frac{256}{\text{Display resolution}}$ in MAXC mode.

(8AFCH is value returned in HL for the Dazzler.)

Figure 3-1. Z80 Assembly Language Standard Display Parameter Field

For convenience several additional subroutines have been defined in this graphics package which were not a part of the original proposed standard. These have been implemented in the FORTRAN, BASIC, and Z80 versions and are described in the following sections as well as in Table 3-2.

AREA Routine

This routine is designed to fill a horizontal, rectangular area on the screen with the current color. The square or rectangle plotted is defined by four points, its four corners. However, it is equally well-defined by its two diagonals. The parameter passed to AREA in HL is the (X,Y) coordinates of any one of the corners, and the AREA routine will define a square or rectangle one of whose diagonals is drawn between this point and the point at which the cursor was positioned upon entering AREA. Either diagonal may be used and HL may contain the coordinates of any of the four corners. For example, the square defined by the four points (0,0), (0,40), (40,40), and (40,0) may be plotted by first positioning the cursor at any of the four corners using the CURSOR routine and then passing AREA the coordinates of the diagonally opposite corner. The cursor will be left positioned at this second set of coordinates.

ROUTINE	GLOBAL NAME	PARAMETERS
AREA	AREA$	H=X corner coordinate; L=Y corner coordinate.
STRING	STRIN$	HL=address of start of string.
PAUSE	PAUSE$	A=delay time in tenths of a second.
CYCLE	CYCLE$	None.

Table 3-2. Additional Non-Standard Graphics Subroutines.

The AREA routine is implemented internally by multiply-calling the LINE routine (which in turn calls DOT). All registers and flags are preserved. If the present cursor position and that passed to AREA agree, a dot will be plotted. If either their X or their Y coordinates agree, a vertical or horizontal line will be plotted, respectively.

STRING Routine

The STRING routine is used to plot a string of characters on the screen by means of a single call (rather than by multiple calls to CHAR). The string may contain any of the control characters described (for example CR and LF) and thus may be used as a fast way of setting display parameters as well as for character plotting. The parameter passed in HL is the address of the start of the desired string; the string is terminated by the parity bit set in the last character to be printed. (The relocatable format of the Assembler and Linker will take care of creating an absolute address if the address you use is symbolic.) Here is a sample implementation of STRING:

```
         LD   HL,STRIN1
         CALL STRING
          :
STRIN1:  DM   'GRAPHICS'
```

The DM pseudo-op automatically sets the parity bit of the last character in the object code. The word "GRAPHICS" will be plotted on the TV screen by this routine call. The STRING routine is implemented by multiple calls to the CHAR routine. All registers are preserved except HL, which is returned pointing to the next character following the last character of the string.

PAUSE Routine

This routine may be used to provide pauses of 0.1 second increments. This is needed because during animation the display may be updated far faster than is desired. The number of 0.1 second pauses desired is passed to the routine in the A register, up to a maximum of 25.5 sec. (A=255 or FFH). If A is 0, a negligible pause is produced.

The routine is implemented by a carefully timed software loop. The 0.1 second time assumes a 4MHz ZPU speed. If 2MHz is being used, all times will be doubled. No registers are changed by this routine.

CYCLE Routine

The CYCLE routine first examines several internal parameters and determines the currently-set color, resolution, intensity, and whether Color or B/W mode. It then changes the display to 128 X 128 mode in the color red (or lowest grey scale), and cycles through all colors (in Color mode) or through all shades of grey (in B/W mode) at tenth-second increments. The display is then returned to its entry settings of color and resolution prior to return to the main program. All registers are preserved. The CYCLE routine is implemented by calls to CHAR to generate the desired display modes and by calls to PAUSE for the required delays. This routine is provided for special color effects.

CHAR Routine Additions

The control characters which the graphics standard CHAR routine is able to interpret and implement are described in Table 2-1. However, since not all available control characters were used, several additional features for this graphics program were assigned to the extra ones. These are summarized in Table 3-3, and described here in the following. Note that Table 3-3 defines all those control characters which were undefined in Table 2-1.

MNEMONIC	ASCII	CONTROL CHARACTER	HEX	STANDARD FUNTION	DAZZLER FUNCTI	
				Display Mode Selection		
MAXR	NUL	CTRL-@	00	Maximum resolution	128X128 B/W	
MAXC	SOH	CTRL-A	01	Maximum colors	64X64 Color	
R128	STX	CTRL-B	02	128 by 128	128X128 Color	
R64	ETX	CTRL-C	03	64 by 64	64X64 Color	
RXXX	EOT	CTRL-D	04	Undefined	64X64 B/W	
				Additional Functions		
NONE	ENG	CTRL-E	05	Undefined	Select Picture Buffer 1	
	ACK	CTRL-F	06	Undefined	Select Picture Buffer 2	
	BEL	CTRL-G	07	Undefined	Toggle Dazzler Off/On	
				Carriage Control		
BS	BS	CTRL-H	08	Backspace (optional)	X=X-6	
HT	HT	CTRL-I	09	Horizontal tab (opt.)	X=(X+32) MOD 32	
LF	LF	CTRL-J	0A	Line feed	Y=Y-8	
VT	VT	CTRL-K	0B	Vertical tab (opt.)	Y=(Y MOD 32)-6	
FF	FF	CTRL-L	0C	Form feed	X=0;Y=-6	
CR	CR	CTRL-M	0D	Carriage return	X=0	
				Character Style		
SO	SO	CTRL-N	0E	Undefined	Fixed	
SI	SI	CTRL-O	0F	Undefined	Proportional	
				Current Color Selection	Color Mode	B/W Mode
BLK	DLE	CTRL-P	10	Black	Black	Black
RED	DC1	CTRL-Q	11	Red	Red	Gray 1
GRN	DC2	CTRL-R	12	Green	Green	Gray 2
YEL	DC3	CTRL-S	13	Yellow	Yellow	Gray 3
BLU	DC4	CTRL-T	14	Blue	Blue	Gray 4
MAG	NAK	CTRL-U	15	Magenta	Magenta	Gray 5
CYN	SYN	CTRL-V	16	Cyan	Cyan	Gray 6
WHI	ETB	CTRL-W	17	White	White	Gray 7
NONE	CAN	CTRL-X	18	NONE	Black	Gray 8
	EM	CTRL-Y	19		Dim Red	Gray 9
ASSIGNED	SUB	CTRL-Z	1A	ASSIGNED	Dim Green	Gray 10
	ESC	CTRL-[	1B		Dim Yellow	Gray 11
	FS	CTRL-\	1C		Dim Blue	Gray 12
	GS	CTRL-]	1D		Dim Magenta	Gray 13
	RS	CTRL-↑	1E		Dim Cyan	Gray 14
	VS	CTRL-_	1F		Gray	White

Table 3-3. Dazzler Graphics Control Character Functions.

The following control characters are now defined:

CTRL-@ through CTRL-D (00-04)

Display mode selection. Although only four of these are described under the Standard Functions, all five are implemented for the Dazzler to cover all possible combinations of color and resolution. Note that CTRL-A (01) and CTRL-C (03) are the same.

CTRL-E (05H)

Select picture buffer 1. This may be used to return to the first picture buffer in real-time display (i.e., the buffer being displayed is the buffer being filled) after a number of calls to ANIMAT, for example. (ANIMAT does not display in real time, i.e., the buffer being displayed is <u>not</u> the buffer being filled.) This control may also be used in conjunction with CTRL-F to switch back and forth between the two picture buffers.

CTRL-F (06H)

Select picture buffer 2. The use and features of this control character are the same as for CTRL-E above.

CTRL-G (07H)

Toggle Dazzler on/off. There is a conflict when doing disk accesses while the Dazzler is turned on (because the Dazzler does a DMA of memory and holds the bus for approximately 15% of the CPU time). This control character is provided for turning off the Dazzler during disk accesses. No flags or internal variables are changed; therefore, the next time this character is used, the Dazzler will be turned back on with the same color/resolution and pointing to the same picture buffer. (INITG turns on the Dazzler initially, pointing to the first picture buffer.)

CTRL-N (0EH)

Fixed character spacing. This control selects a fixed character width for all characters. It is useful if it is desired to have characters line up in vertical columns on the screen or if the backspace control character (CTRL-H 08) is to be used, which always backs up the cursor one fixed character width.

CTRL-O (0FH)

Proportional character spacing. This is the default mode set by INITG, and provides for greater legibility and the ability to type more characters per line than with fixed character spacing. In this mode the letter "I" for example occupies a much smaller width than the letter "W".

CTRL-X through CTRL-_ (18H-1FH)

Eight additional colors. These eight control characters are used to provide the eight Dazzler low-intensity colors which correspond to the eight high-intensity colors, CTRL-P through CTRL-W. In black and white mode, CTRL-X through CTRL-_ correspond to the eight <u>higher</u> intensity shades of gray on the gray scale. See Table 3-3, where the higher the number on the gray scale, the brighter the shade selected.

FORTRAN IV Graphics Protocol

The standard graphics routines have also been included on this diskette in an interface to CROMEMCO FORTRAN IV in the file GRAPHFOR.REL. This is a relocatable object program very similar to GRAPHZ80.REL and, in fact, includes all of the routines from that file in unchanged form. However, additions have been made to allow FORTRAN to call the routines directly with complete parameter passing ability. FORTRAN passes the addresses of parameters in the register pairs of the Z80. Therefore, when a routine requiring X and Y is called from FORTRAN, for example, the address pointing to X will be found in the HL register pair and the address pointing to Y in the DE register pair. The called routine then loads X and Y from these addresses into H and L, respectively, prior to calling the graphics routine desired.

The global names of Tables 3-1 and 3-2 are used exactly the same in FORTRAN, and the parameters required by the various routines are also the same. (Refer to the sections on SUBPROGRAMS and Calling Machine Language Subroutines of the FORTRAN IV Instruction Manual for more information on the way parameters are passed.) An example FORTRAN program passing integer X and Y values to CURSOR might be:

```
IXVAL=128
IYVAL= 40
CALL CURSR$(IXVAL,IYVAL)
```

Whereas for the Z80 program the Routine names and Global names of Tables 3-1 and 3-2 could be used interchangeably, they cannot be for FORTRAN. The Global names (those ending with $) are now the names which must be used when calling these routines from FORTRAN. The other names still correspond to the Z80 subroutines themselves. Thus, a FORTRAN program may either call a machine language subroutine which in turn calls a graphics subroutine by its non-global name, or FORTRAN may call the graphics routine itself directly by its global name.

Refer to the FORTRAN IV User's Manual for more information on the correct method to create, compile, link and load CROMEMCO FORTRAN programs. Also, refer to Chapter 5 of Part I of this manual for some examples.

16K BASIC Graphics Protocol

This interface between the graphics standard and CROMEMCO 16K BASIC is perhaps the most versatile of the programs on the DGR disk. It has been designed to be completely interactive so the user can actually see what is being created on the screen; then when the program is complete it can be run directly from the disk using BASIC's extensive disk-file-handling capabilities. The 16K BASIC-graphics interface is contained in the file GRAPHX.COM. Notice that this is a CDOS command file. The use of this will be explained further in the following.

An unfortunate consequence of the extended capabilities of this graphics program is that more than 32K of memory is required to run a graphics-BASIC program. The recommended procedure is to boot up with either a 48K or a 64K operating system (CDOS) prior to attempting to run GRAPHX. This allows room in memory to contain the graphics program itself, the two 2K picture buffers, all of BASIC and its associated user space, as well as all of CDOS. Since GRAPHX is a command file, its name may be typed directly from the console. When this is done, the following events occur: First, GRAPHX is loaded into memory from the disk and the sign-on message is printed. Then, the program determines the bottom address of CDOS and reserves 4K bytes below that (beginning on the next lowest 512-byte boundary) for the picture buffers. Next, GRAPHX relocates itself in memory to just below the picture buffers. Finally, the directory of the current disk is searched for the file BASIC.COM, which, if found, is then loaded into memory below the graphics program and beginning at 100H. If BASIC is not found on the current drive, drive A is also searched. (If the current drive is A, this step is skipped.) If any errors occur during the above procedure (for example, insufficient memory or BASIC not found), an error message will be printed on the console and control returned to CDOS.

Once BASIC has been successfully loaded into memory and prior to execution, the following additional steps take place: The graphics program finds the location of the Device Driver Table in BASIC, i.e., DDLIST, and creates an entry in the table called "DG" for "Dazzler Graphics". This entry points to the 16-byte address table for the DG driver. Finally, the routine checks to see if a filename was typed following the word "GRAPHX" initially. If so, this file is then LOADed into BASIC's user area and its execution begun.

The following example will serve to further illustrate the above procedures. Suppose a graphics program is created in BASIC and is then SAVEd on disk (using the BASIC SAVE command). Suppose further that both BASIC and this saved program called DEMO are on the disk in drive B and that the DGR disk is in drive A. Drive B is selected as the current drive and then the command line is typed:

```
A:GRAPHX DEMO
```

This command then loads GRAPHX, which loads and chains to BASIC, which in turn loads and begins execution of the program DEMO. Because execution can be achieved via a single command line (no intermediate steps), Dazzler Graphics programs can be executed through .CMD files in Batch (@) mode (see CDOS Manual).

Although the preceding information is not all necessary to successfully use the GRAPHX.COM program, it is provided here as an aid in understanding to the interested user. What follows now is a description of the BASIC-graphics interface driver itself.

Since Dazzler Graphics as interfaced to 16K BASIC has been implemented as a device driver, parameters and quantities are passed as they would be to any device through PUT and PRINT statements. PUT and PRINT statements allow for the passing of 0, 1, or 2 parameters within the slash marks (which also contain the device number as it was given in the OPEN statement). The BASIC driver has the name "DG"; thus, the general form of the OPEN statement is:

```
OPEN\N\"$DG"
```

where N stands for a number 1-4 in the standard production CROMEMCO BASIC. The "$" above distinguishes "DG" from a disk file. The general format of the calling sequence from BASIC is:

```
PUT\N[,X,Y]\"C1[C2C3...]"
```

where N is the file specifier used in the OPEN statement, X and Y are optional parameters corresponding to the X and Y cursor position, and C1, C2, C3, etc. are the various commands as given in Table 3-4. Note that these commands may be strung together to as many as will fit on one BASIC line. Also, the PRINT statement may be used in place of the PUT above. There are two basic differences between PUT and PRINT (at least for the purposes of Dazzler Graphics): (1) the PRINT transmits a CR-LF (carriage return-line feed) following the termination of any command string, (2) the PRINT command is somewhat slower

CHARACTER	OPERATION
+* @	Area fill routine. Fills a rectangle from the current cursor position to the (X,Y) value supplied in the call, with the currently selected color. The cursor is left at the value of (X,Y) given. The two points determining the rectangle, i.e., the current cursor position plus the (X,Y) given may lie along either diagonal of the finished rectangle.
* A	Animate the display. Causes the buffer which is being filled to be displayed and a second buffer to be opened for filling. The buffer exchange is done during the DAZZLER'S vertical blanking interval.
B	Select Blue as the current color. All subsequent points will be plotted in blue.
C	Cyan is made the current color.
* D	A Dot is plotted at the current cursor position in the currently selected color.
E	The display is placed in Extended resolution mode (128x128).
F	Select Fixed spacing between displayed characters, regardless of width.
G	Green is made the current color.
H	The current color will be plotted with High intensity.
I	The current color will be plotted with low intensity.
J	Select buffer #1. Stop animate mode and return to real-time updates displaying picture buffer #1.
* K	Clear (remember Klear) the currently displayed buffer. Entire screen will then be black.
+* L	Draw a Line (using the current color) from the current cursor position to the (X,Y) value supplied in the call. The cursor is left at the end of the line corresponding to the (X,Y) given.
M	Magenta is made the current color.
N	Black (No color) is made the current color.
O	Toggles DAZZLER picture On/Off.

CHARACTER	OPERATION
+* P	Position cursor to the supplied value of (X,Y).
* Q	Cycle through all colors (or through all shades of gray if display is in B/W mode).
R	Red is made the current color.
S	The display is set for Simple resolution (64x64).
* T	Type the next characters in the string starting at the current cursor position. The T can be omitted if the desired characters are supplied in lower case. The first upper case character next encountered in the string will then be interpreted as a command.
U	Select buffer #2. Stop animate with picture buffer #2 displayed in real-time.
V	Select Variable spacing between characters depending on character width.
W	White is made the current color.
X	The display is placed in B/W Mode.
Y	Yellow is made the current color.
Z	The display is placed in Color Mode.
* \	Re-initialize the display to default parameters. "OPEN"ing a file also does this automatically.
CR	Carriage Returns are ignored (unless sent following the "T" command as CHR$(13)).
LF	Line Feed displays both a carriage return-line feed for typed characters on the screen. LF's following the CR in a PRINT statement are ignored, however.
NUL	These are ignored if they follow the CR. (If they precede the CR, they are interpreted as a CTRL-@, or MAXR.)

NOTE: OPEN\1\"$DG" will turn on the DAZZLER, 128x128 mode, black and white, variable spacing, high intensity white, real-time mode, cursor position at (X,Y)=(0,0). The display area is <u>not</u> cleared.

CLOSE\1\ will leave all modes undisturbed, but will turn off the DAZZLER (the DAZZLER must be turned off in order to access the disk).

Table 3-4. DG Device Driver-BASIC Commands for GRAPHX.COM.

than PUT. The CR sent to the DG driver by PRINT is ignored <u>in all cases</u>. The LF (and also any NULLS sent after the LF) are also ignored <u>provided they follow the CR</u>. If the LF or NULL precedes the CR, then it is treated as one of the control characters as explained in Table 3-4.

The X and Y parameters which are passed are converted to integers first. They may or may not be referenced by the routines called, but will be ignored if not needed. The routines requiring X and Y are listed in Tables 3-1 and 3-2. Some of the most common features of CHAR were singled out and assigned a single-letter command as in Table 3-4. These functions do not require a parameter as they generate their own. If it is desired to send specific control characters to CHAR (for example a CR which is usually ignored when coming from a PUT or PRINT statement), the "T" or Text command is put in the statement. All text of any kind which follows on that same BASIC line will then go to the CHAR routine. Note from Table 3-3 that all control characters have been assigned a function under Dazzler Graphics. It turns out in fact that nearly every character in the ASCII character set has been assigned a function here. The ASCII character set (seen in Figure 3-2) can be roughly divided into four groups as shown. Column A contains the control characters of Table 3-3. These can be output to DG by following the "T" command as already mentioned. Column B contains standard punctuation and number digits. These can be output directly to CHAR (through PUT or @ statements) and will be printed on the screen as the characters themselves in the current color. Characters in column C comprise the commands of Table 3-4. Those commands which correspond to the 7 subroutines of the graphics standard as well as the additional routines AREA and CYCLE (STRING and PAUSE are not implemented because of the ease with which they can be done through BASIC) are marked in the Table with an

		Columns			
		A	B	C	D
	HEX VALUE	High Digit of Hex Number			
		0	2	4	6
Low Digit of Hex Number	0	CTRL-@	SPACE	@	`
	1	CTRL-A	!	A	a
	2	CTRL-B	"	B	b
	3	CTRL-C	#	C	c
	4	CTRL-D	$	D	d
	5	CTRL-E	%	E	e
	6	CTRL-F	&	F	f
	7	CTRL-G	'	G	g
	8	BACKSPACE	(	H	h
	9	HORI.TAB	)	I	i
	A	LINE FEED	*	J	j
	B	VERT.TAB	+	K	k
	C	FORM FEED	,	L	l
	D	CARR.RETURN	-	M	m
	E	CTRL-N	.	N	n
	F	CTRL-O	/	O	o
		1	3	5	7
	0	CTRL-P	0	P	p
	1	CTRL-Q	1	Q	q
	2	CTRL-R	2	R	r
	3	CTRL-S	3	S	s
	4	CTRL-T	4	T	t
	5	CTRL-U	5	U	u
	6	CTRL-V	6	V	v
	7	CTRL-W	7	W	w
	8	CTRL-X	8	X	x
	9	CTRL-Y	9	Y	y
	A	CTRL-Z	:	Z	z
	B	CTRL-[	;	[	{
	C	CTRL-\	<	\	|
	D	CTRL-]	=	]	}
	E	CTRL-↑	>	↑	~
	F	CTRL-_	?	_	DEL

Figure 3-2. ASCII Character Set.

asterisk ("*"). Those requiring an (X,Y) coordinate for the cursor are also marked with a plus ("+") sign. The ASCII characters in column D of Figure 3-2 may be sent to the driver, but will be interpreted and printed on the screen as their upper-case counterparts of column C. For example, the letter "d" sent to DG through a PUT or PRINT statement would be printed on the screen as "D". This feature is provided to prevent having to use "T" to type output on the screen. (With the "T" command the entire rest of the line is taken as text, which may be considered an inconvenience.) Some of these features will be illustrated in the example at the end of this section.

There are a few more points to be made on the procedure of using the device driver "DG". The values for X and Y should be in the range 0 through 255. If they are not, the value used will be MOD 256. (For example, if X is 265, then the actual value used for X will be 9.)

The OPEN routine calls the INITG routine, thus turning on the Dazzler and initializing all parameters. The cursor is set to (0,0), color is set to white in B/W mode at high resolution and intensity. The CLOSE routine should always be called when finished with a graphics program; it serves to turn off the Dazzler (thus preventing problems with disk access). Within a graphics program the "O" command may be used to turn on and off the Dazzler.

Finally, BASIC has provisions for reading values from a driver through the IOSTAT function (which works through the GETSTAT routine of a driver). The driver DG will return up to 5 IOSTAT's, which are listed on the next page. These allow a BASIC programmer to make decisions based on such things as the current cursor position. These functions are <u>not</u> implemented in the FORTRAN and Z80 versions of Dazzler Graphics, thus making GRAPHX.COM even more versatile than is immediately apparent.

SCREEN READ FUNCTIONS

A=IOSTAT(N,1)	returns the X co-ordinate of the current cursor position.
A=IOSTAT(N,2)	returns the Y co-ordinate of the current cursor position.
A=IOSTAT(N,3)	returns a value corresponding to the screen color at the current cursor position according to the chart shown below.
A=IOSTAT(N,4)	returns a 64 if in 64x64 mode or a number from 0-15 representing the screen color or shade of gray if in 128x128 mode, according to the chart below.
A=IOSTAT(N,5)	returns a 0/1 for B&W/Color mode.

(where N stands for a file specifier, usually 1-4.)

IOSTATS 3 and 4 return one of the values in column 1 below depending on the mode and resolution selected. The mode can in turn be determined (if unknown) from IOSTAT 5 as above. IOSTAT 4 returns a value of 64 if in 64x64 resolution; it may therefore be used to test resolution as well as color (test for a value =64 for 64x64 and <64 for 128x128).

	IOSTAT NO.	MODE		
	4	128x128-COLOR	128x128-B/W	(not applicable)
	3	64x64-COLOR	64x64-B/W	128x128-COLOR or B/W
Returned Value	0	Black	Black	Black (dot off)
	1	Low Red	Gray 1	White (dot on)
	2	Low Green	Gray 2	
	3	Low Yellow	Gray 3	
	4	Low Blue	Gray 4	
	5	Low Magenta	Gray 5	
	6	Low Cyan	Gray 6	
	7	Low White	Gray 7	
	8	(not used)	Gray 8	
	9	Hi Red	Gray 9	
	10	Hi Green	Gray 10	
	11	Hi Yellow	Gray 11	
	12	Hi Blue	Gray 12	
	13	Hi Magenta	Gray 13	
	14	Hi Cyan	Gray 14	
	15	Hi White	White	

Table 3-5. Summary of IOSTAT Functions 3 and 4 Implemented for the DG Driver.

For example a graphics program which plots a function could be self-modifying to rescale the axes if the function is found to go out of range.

This chapter is now concluded with a well-commented example of some of the features just described of GRAPHX.COM:

```
10 OPEN\1\"$DG"
```

Dazzler Graphics is opened (Dazzler turned on) and assigned to file specifier 1.

```
20 PUT\1\"KSZ"
```

The screen is cleared (K), set to simple, or 64 X 64, resolution (S), and set to Color Mode (Z).

```
30 PUT\1,64,64\"RP"
```

The current color is set to red (R) and the cursor is positioned (P) at (64,64). Note the cursor parameters are ignored by the routine which sets color.

```
40 PUT\1,192,64\"L" : PUT\1,192,192\"L"
```

Two straight lines are drawn (L), the first from (64,64) to (192,64) and the second from (192,64) to (192,192).

```
50 @\1,64,192\"L" : @\1,64,64\"L"
```

Two more lines are drawn, thus completing the red square. Note that PRINT statements are used this time and that the CR-LF's following the "L's" are ignored.

```
60 PUT\1,70,120\"BPsquare";CHR$(10)
```

The color blue is selected (B), the cursor is positioned at (70,120), the word "SQUARE" is written on the screen (translated to upper case), and the cursor is moved by a CR-LF (CHR$(10) is a LF which generates both CR-LF) to be ready to plot a new line of text.

```
70 FOR I=1TO2000
80 NEXT I
```

A short pause is generated to allow viewing of the display.

```
90 CLOSE\1\
```

The DG driver is closed and the Dazzler turned off.

```
100 END
```

Chapter 4: FUNCTION ALGORITHMS

The algorithms used to produce the CROMEMCO DAZZLER implementation of the Z80 Assembly Language Standard (upon which the BASIC and FORTRAN standards are based) are provided here as an aid in understanding their function. Of particular interest to most readers will be the line and character generation algorithms, which work completely independently of the hardware configuration of the display used.

The algorithms which follow are illustrated by the use of Nassi-Shneiderman design charts. For those readers who are not familiar with Nassi-Shneiderman charts, a brief explanation is in order. More detailed information can be found in their original article published in the August 1973 SIGPLAN Notices. The Nassi-Shneiderman chart is a stylized flow chart for structured programming. By supporting only standard structured programming constructs (see Figure 4-1) and not supporting GOTO's and off-page connectors, it 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. Other adaptations which may be required are the refresh memory size parameter in PAGE, the color and mode select controls in CHAR, and the scale factors used by the internal subroutine SCALE.

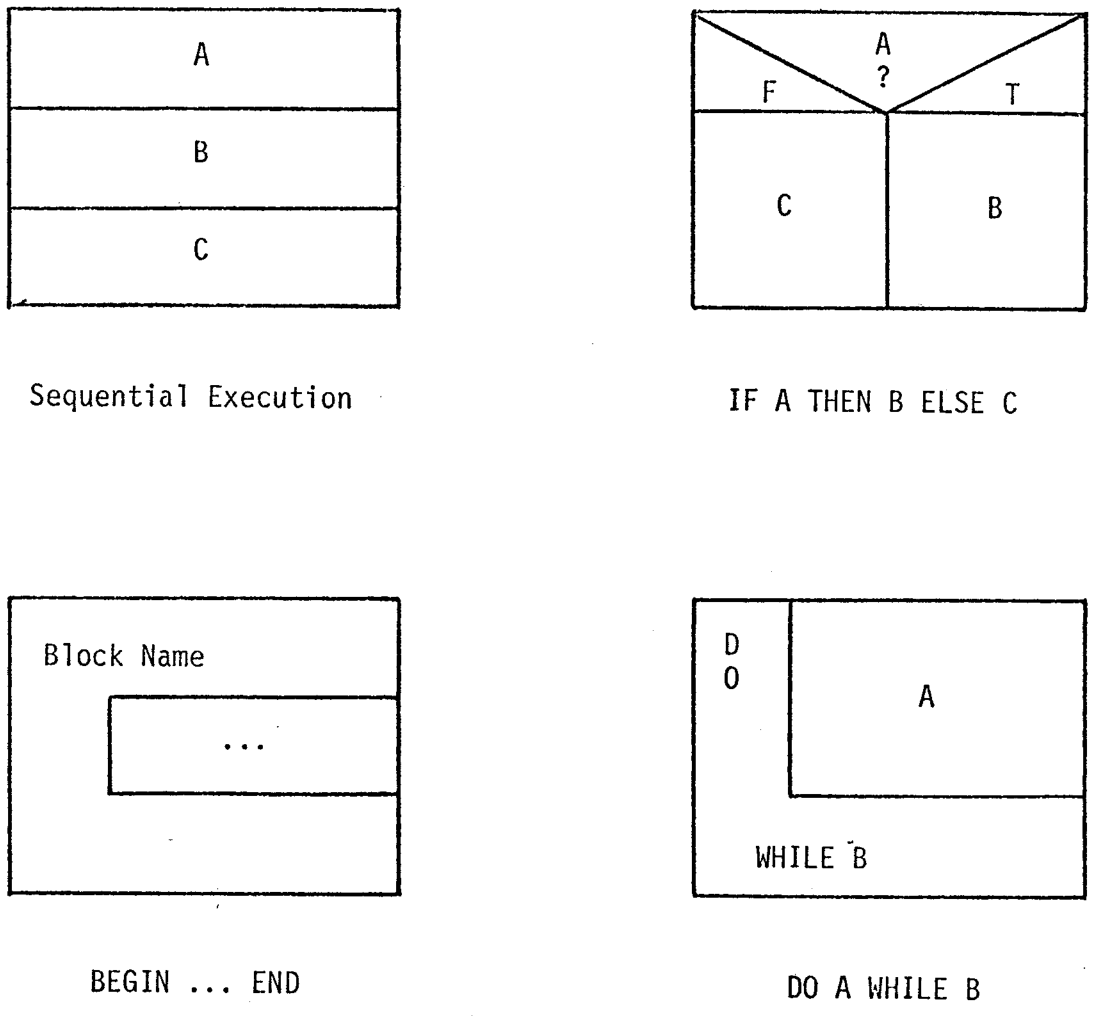


Figure 4-1. Sample Nassi-Shneiderman Constructs

INITG Logic

Initialization is normally required for both hardware and software. The first step is to establish the refresh buffer. This requires taking the address defining 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. The PAGE routine is not called to clear the screen so that the program may be re-entered at a later time without fear of erasing what has already been plotted in the picture buffers. To override this feature, PAGE may be called immediately after calling INITG. The appropriate display description is generated and control is returned to the calling program.

PAGE Logic

The PAGE command clears all the memory used for display refresh. Probably the most general algorithm and the one charted is some variety of 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 just 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. 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 be increasing to the right and Y

INITG

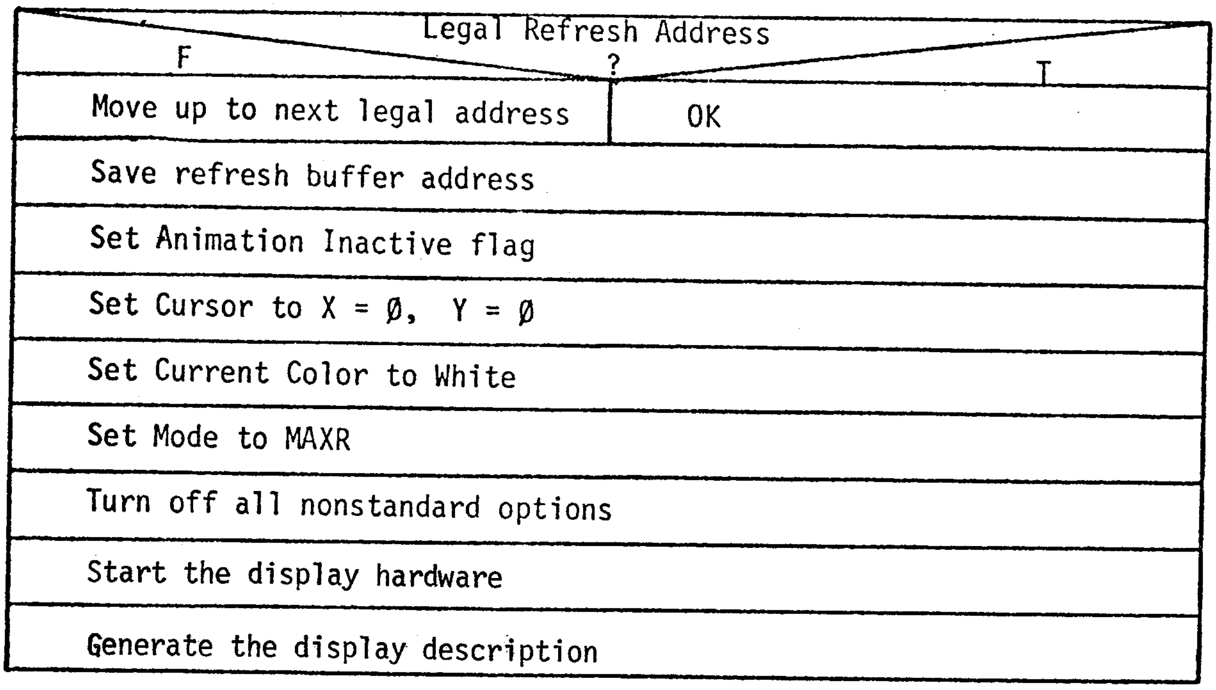

Legal Refresh Address
?
F
T
Move up to next legal address
OK
Save refresh buffer address
Set Animation Inactive flag
Set Cursor to X = ∅, Y = ∅
Set Current Color to White
Set Mode to MAXR
Turn off all nonstandard options
Start the display hardware
Generate the display description

Figure 4-2. Nassi-Shneiderman Chart for INITG Routine.

PAGE

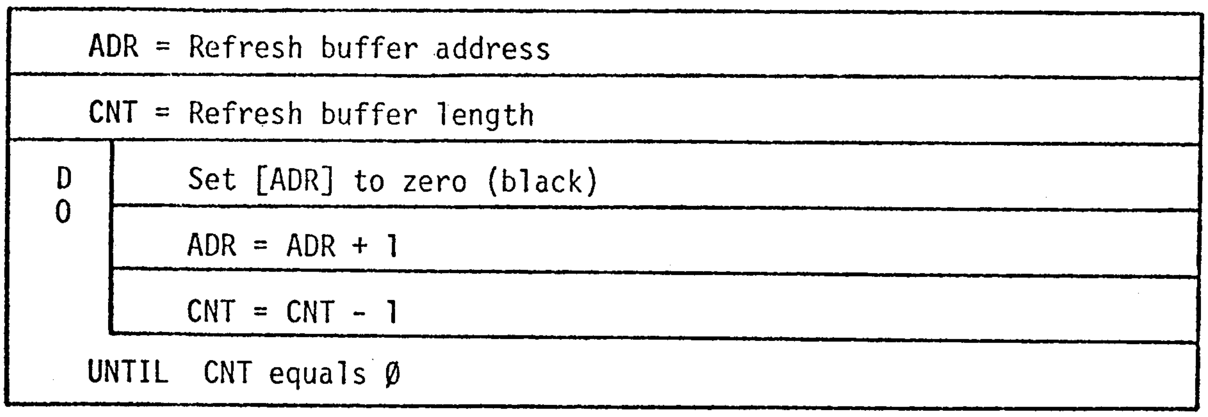

ADR = Refresh buffer address
CNT = Refresh buffer length
DO
Set [ADR] to zero (black)
ADR = ADR + 1
CNT = CNT - 1
UNTIL CNT equals ∅

Figure 4-3. Nassi-Shneiderman Chart for PAGE Routine.

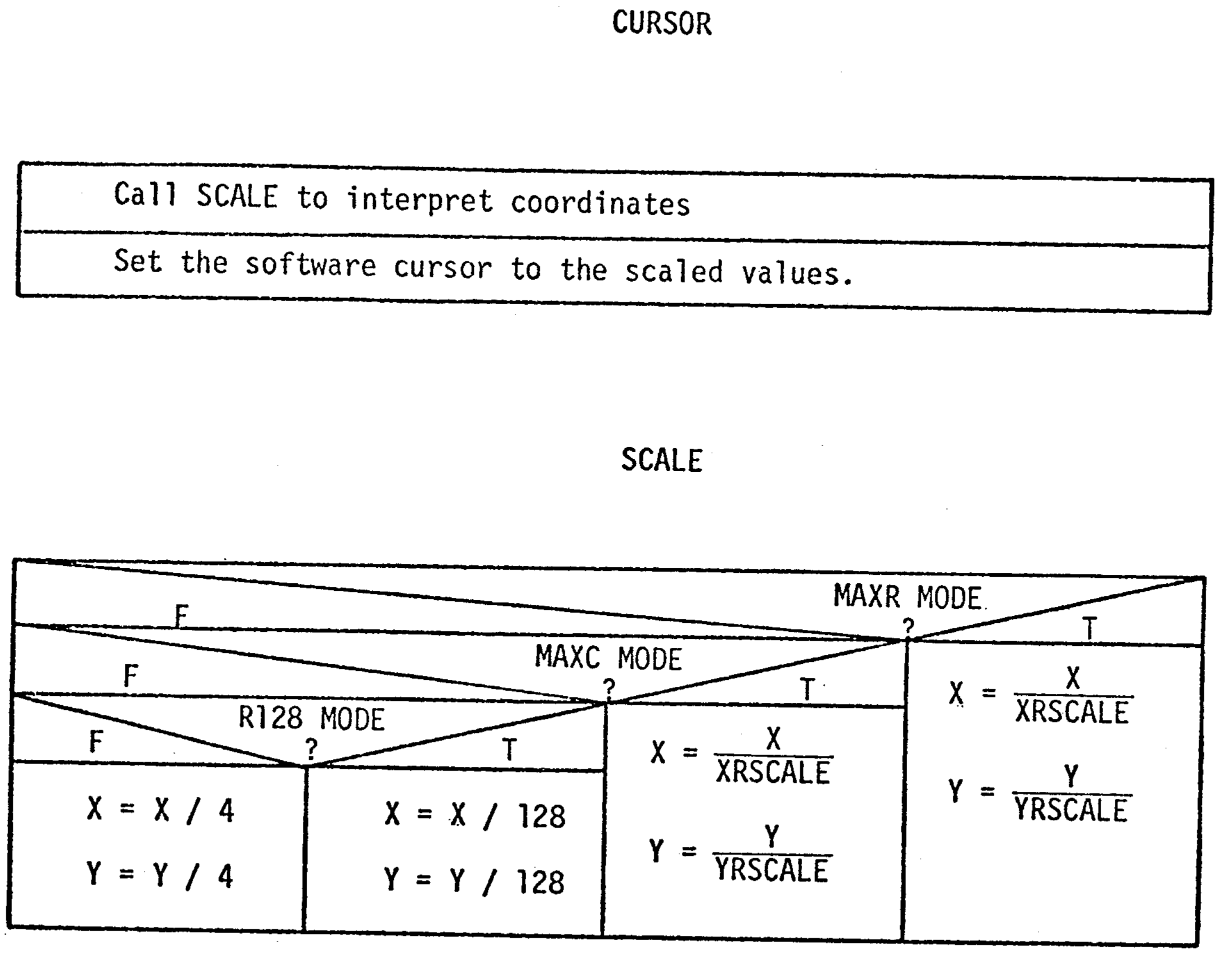


Figure 4-4. Nassi-Shneiderman Chart for CURSOR Routine.

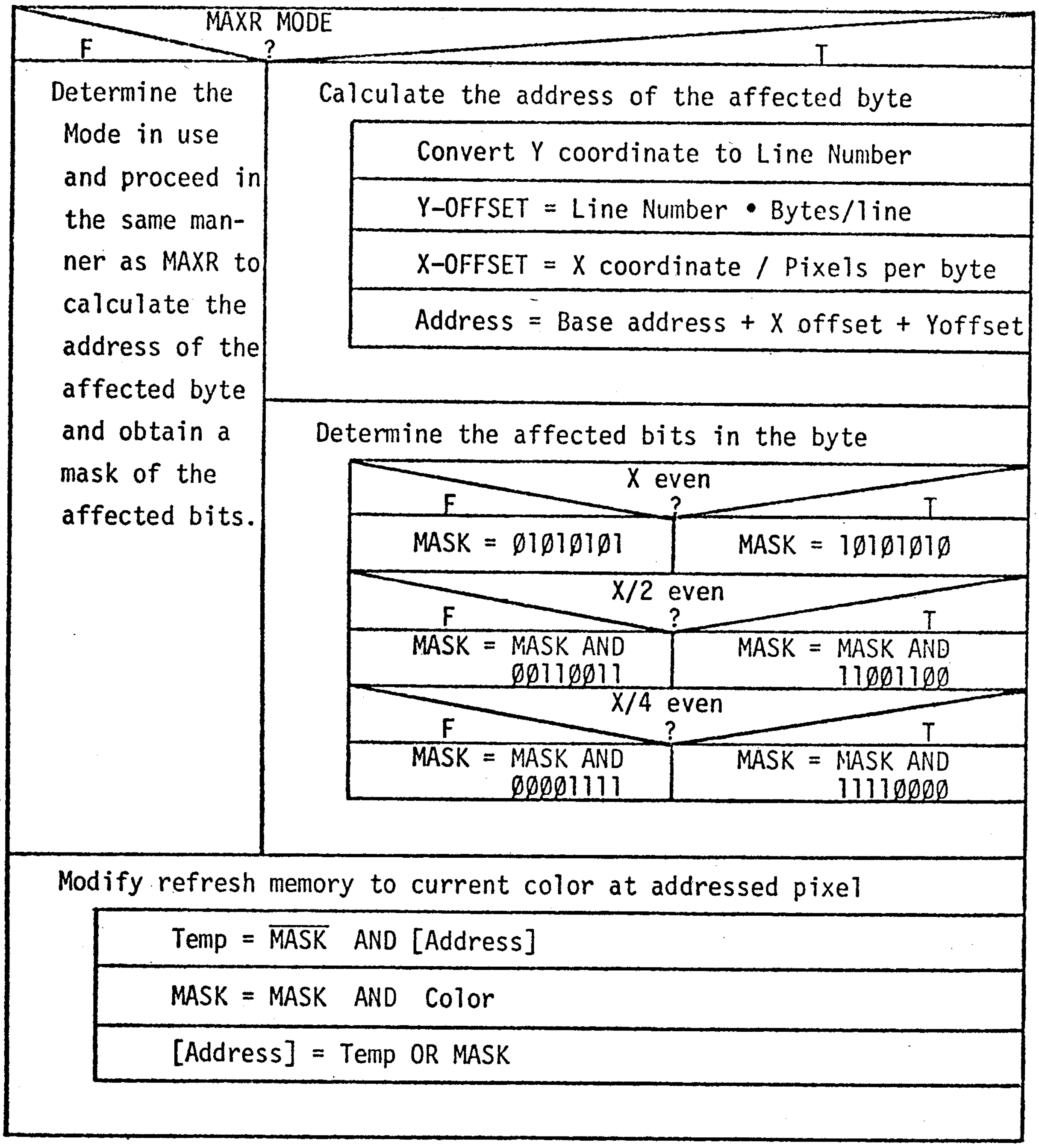
coordinates to be increasing to the top. Since LINE must also scale its arguments, CURSOR and LINE can usually share the same internal scaling routine for efficiency. This routine is also outlined in Figure 4-4.

DOT Logic

DOT is the only routine (other than PAGE) which actually modifies the refresh memory. LINE and CHAR both use it to modify the desired pixels in the display (this is what makes them hardware-independent). This routine is extremely hardware-dependent. Indeed, one of the primary reasons for defining this protocol was for protection from differing display idiosyncracies. The DOT routine must translate the coordinates in the software cursor to the actual bits in memory that correspond. Keep in mind 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--top line first. Note that this algorithm is adjusted slightly for the DAZZLER since it divides the display into four quadrants, each in its own block of memory with each byte describing points on more than one line. These modifications to the algorithm are explained in the sample implementation.

The first step is to determine the address of the byte containing 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 normally 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

DOT



```
MAXR MODE ?
F:
    Determine the Mode in use and proceed in the same manner as MAXR to
    calculate the address of the affected byte and obtain a mask of the
    affected bits.
T:
    Calculate the address of the affected byte
        Convert Y coordinate to Line Number
        Y-OFFSET = Line Number • Bytes/line
        X-OFFSET = X coordinate / Pixels per byte
        Address = Base address + X offset + Yoffset
    Determine the affected bits in the byte
        X even ?
            F: MASK = Ø1Ø1Ø1Ø1
            T: MASK = 1Ø1Ø1Ø1Ø
        X/2 even ?
            F: MASK = MASK AND ØØ11ØØ11
            T: MASK = MASK AND 11ØØ11ØØ
        X/4 even ?
            F: MASK = MASK AND ØØØØ1111
            T: MASK = MASK AND 1111ØØØØ
Modify refresh memory to current color at addressed pixel
    Temp = $\overline{\text{MASK}}$ AND [Address]
    MASK = MASK AND Color
    [Address] = Temp OR MASK
```

Figure 4-5. Nassi-Shneiderman Chart for DOT Routine.

adding the offsets to the desired line in the buffer and the desired point on the line yields the address of the byte requiring modification.

The second step is to determine which bits in the byte correspond to the desired pixel. The hypothetical display the Nassi-Shneiderman chart depicts has eight pixels in each byte. The selected bits are then changed to match the current color and the refresh memory updated to reflect the revised point. An effective procedure is to generate a mask containing 1's at bit positions corresponding to the addressed point and 0's 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 ORed into the cleaned-up byte of refresh memory.

LINE Logic

Probably the most crucial facet of any graphics system is its line generator. 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 4-6.) 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 the point (0,0) to a point (X,Y) where X is greater than or equal to Y and both are greater than or equal to zero (Figure 4-7). (This situation is still 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 $(0 \leq x \leq X)$ and for

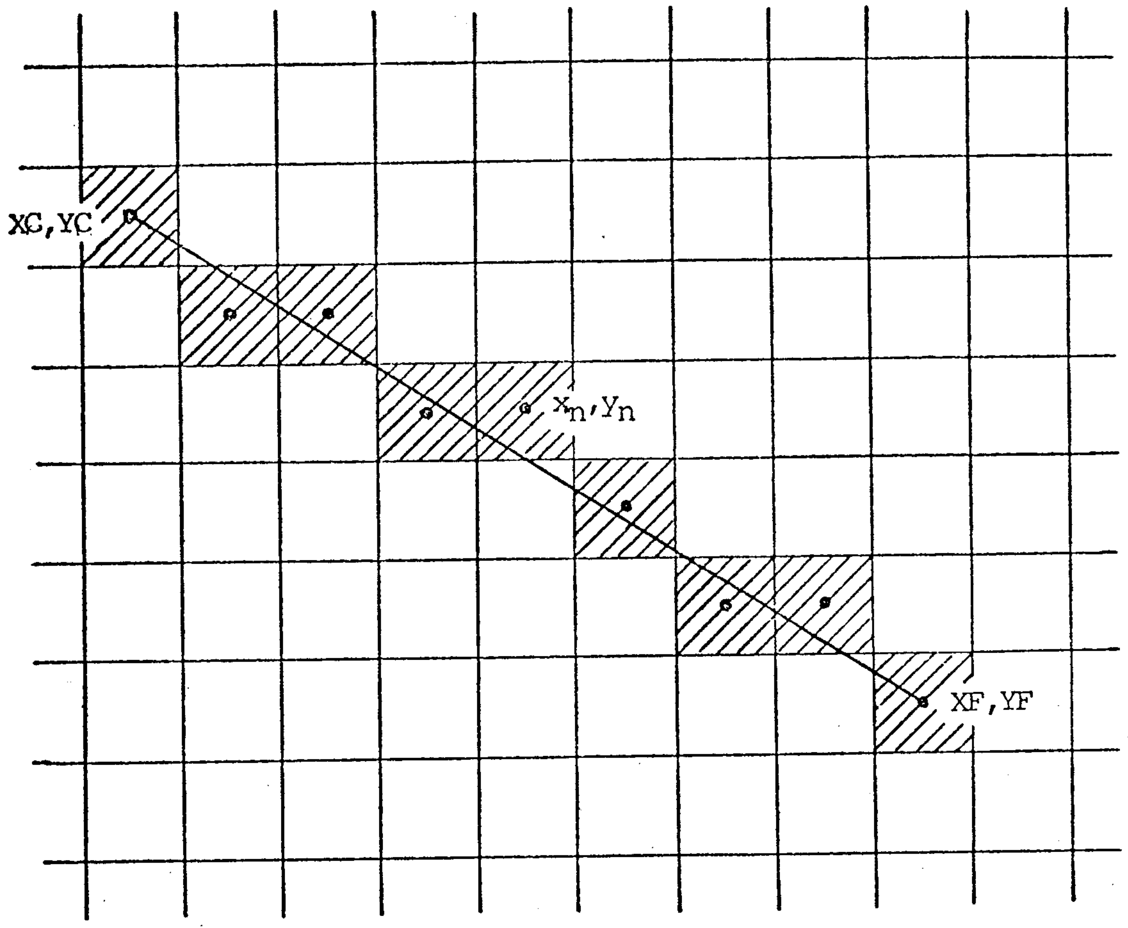


Figure 4-6. Generating an Arbitrary Line.

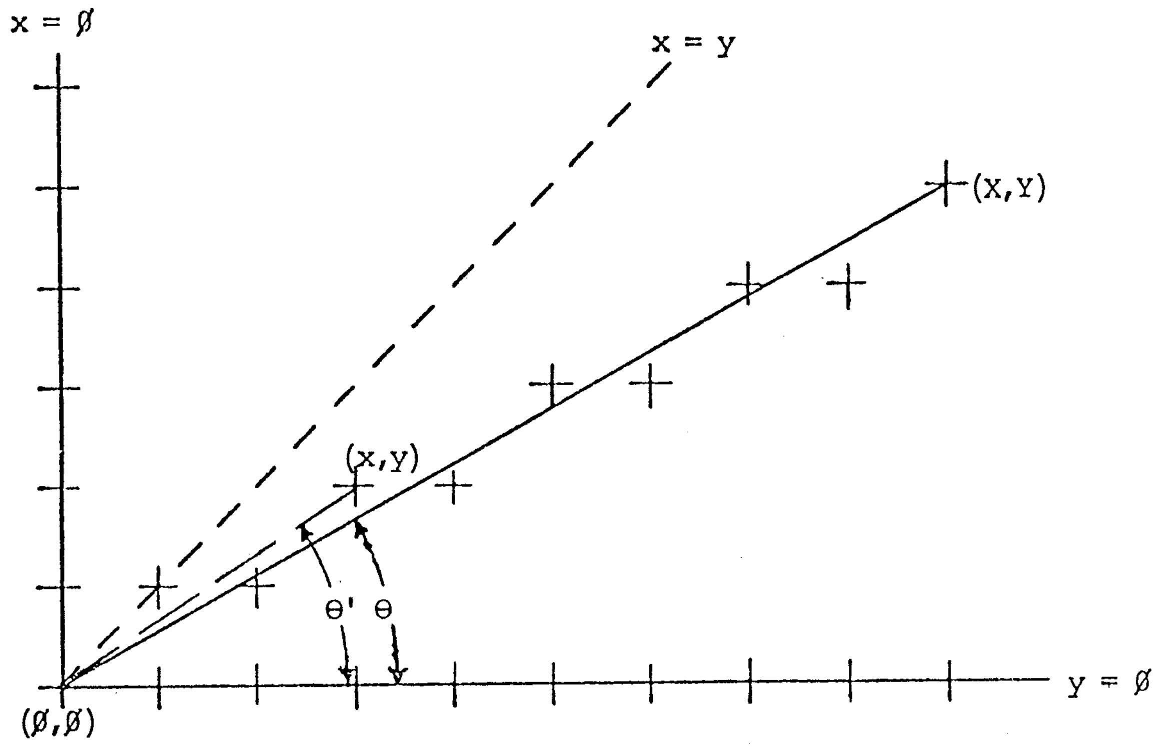


Figure 4-7. Simplified Line Generation.

each 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 be increased by one. 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 above the line, y remains unchanged. If 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 we have some method to determine the position on 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 $X \geq Y \geq 0$ using the following algorithm:

```
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 back to Figure 4-7, 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 be comparing the quotients $\frac{Y}{X}$ and $\frac{y}{x}$.

Unfortunately, division is a time consuming process on microcomputers. Using the properties of inequalities to eliminate the divisions we can build the decision table in Table 4-1, which requires only multiplication. Returning

	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 $x \cdot X$	$xY < Xy$	$xY = Xy$	$xY > Xy$
Result of $xY - Xy$	Neg	Zero	Pos

Table 4-1. Point Position Relative to a Line.

to our original algorithm, we set dy to 1 if

$$(x + 1) \cdot Y > X \cdot (y + \tfrac{1}{2})$$

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 that on the right either remains the same or increases by X. By remembering the products from the previous iteration and whether or not y is incremented, the multiplication can be reduced to additions. For maximum efficiency the right-hand product can be maintained negated so the comparison can be made with a single addition.

The restriction that the line be from (0,0) to a point (X,Y) with $X \geq Y \geq 0$ requires the use of coordinate translations, rotations, and reflections. The first step is to translate the line so it starts at (0,0). Since the line originates at the cursor, traditionally we would subtract the cursor from the other endpoint to obtain its relative position. Because a 256 by 256 display does not give us room for a sign bit in an eight 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 $\geq$ XC, then we are in the first or fourth quadrant (see Figure 4-8); otherwise, we are in the second or third. Similarly, if YF $\geq$ YC we are in the first or second quadrant; otherwise, the third or fourth. 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 in Table 4-2. Finally XM and YM are compared to determine the exact sector.

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

Table 4-2. Component Magnitudes in the Four Quadrants.

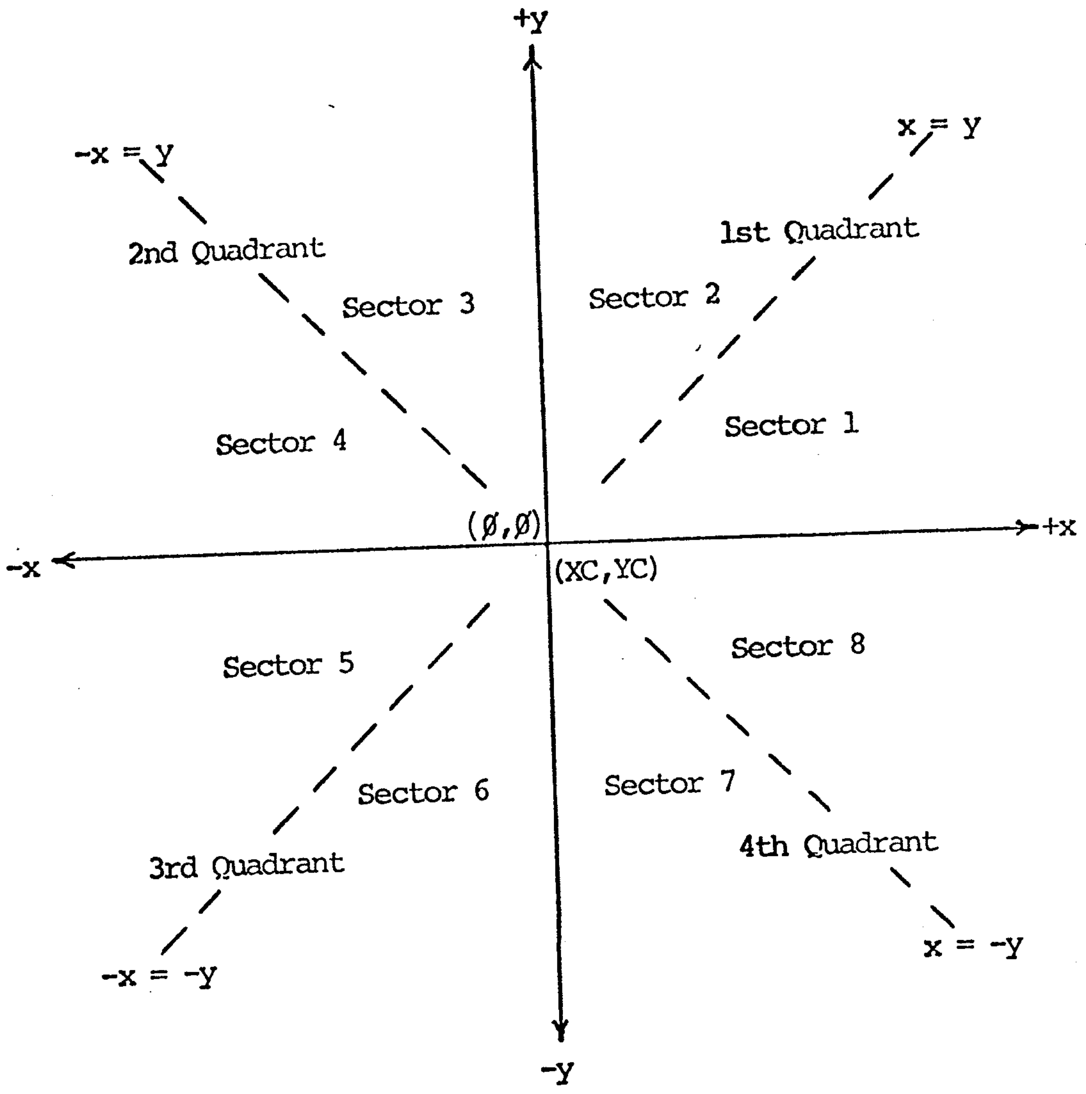


Figure 4-8. Quadrant and Sector Definition.

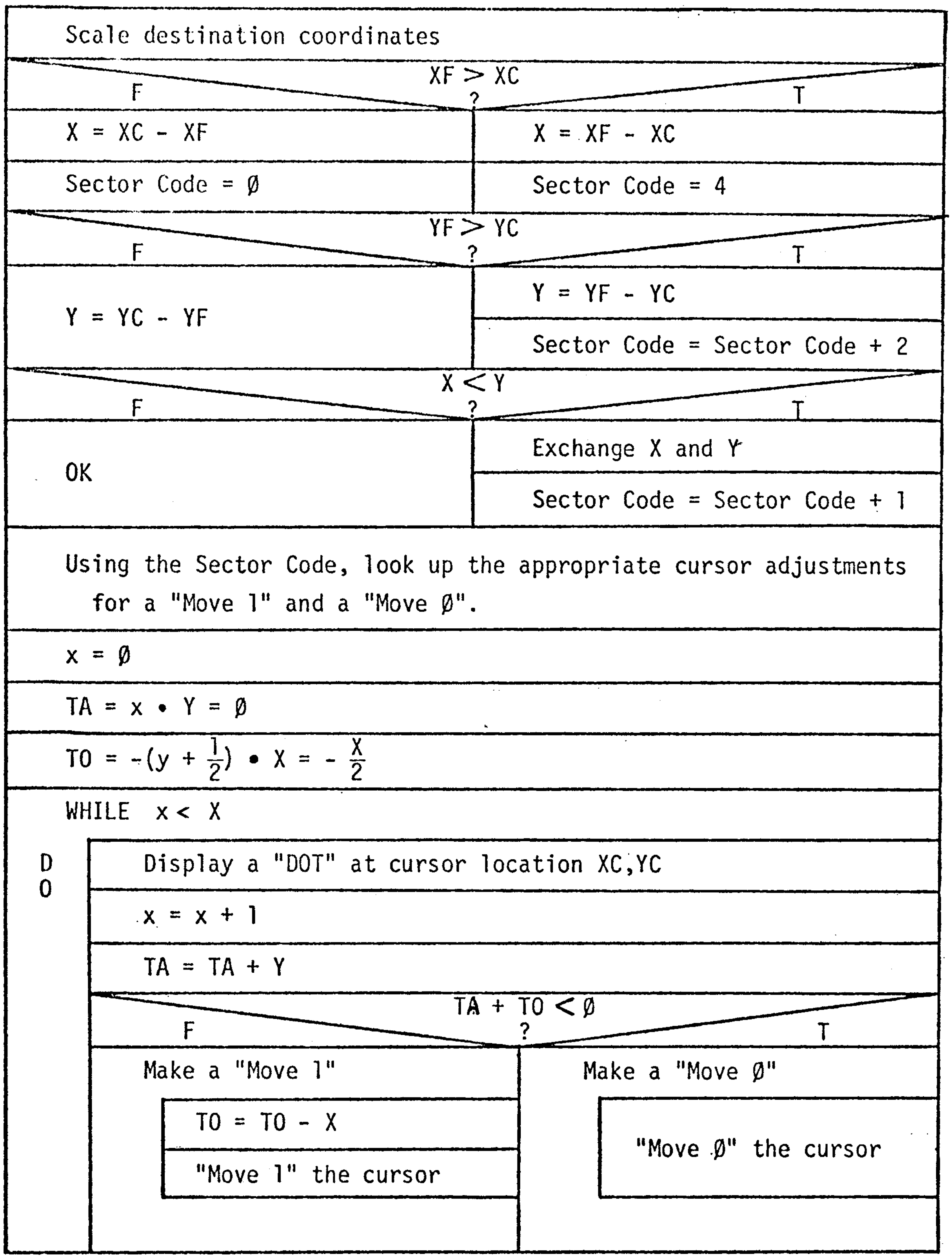
The easiest technique for remembering this multiple logical decision is to weight the results of each decision and check the sum. Each sector is then assigned an equivalent weight and the sector parameter table reordered accordingly. Column 2 of Table 4-3 applies a weight of 4 to (XF > XC), 2 to (YF > YC) and 1 to (YP > XP).

	SECTOR			MOVE 0		MOVE 1	
SECTOR	WEIGHT	X	Y	x incr	y incr	x incr	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 4-3. Coordinate Equivalents for each Sector.

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" and setting dy to 1 "move 1" and generate the equivalence chart in Table 4-3. As the algorithm steps along in transformed coordinates, it modifies the cursor position using the "move 0" and "move 1" X and Y increments appropriate for the sector the line actually lies in.

The LINE routine algorithm is summarized in the Nassi-Shneiderman chart of Figure 4-9.



```
Scale destination coordinates
XF > XC ?
  F:
    X = XC - XF
    Sector Code = Ø
  T:
    X = XF - XC
    Sector Code = 4
YF > YC ?
  F:
    Y = YC - YF
  T:
    Y = YF - YC
    Sector Code = Sector Code + 2
X < Y ?
  F:
    OK
  T:
    Exchange X and Y
    Sector Code = Sector Code + 1
Using the Sector Code, look up the appropriate cursor adjustments
  for a "Move 1" and a "Move Ø".
x = Ø
TA = x • Y = Ø
TO = -(y + 1/2) • X = - X/2
WHILE  x < X
DO
    Display a "DOT" at cursor location XC,YC
    x = x + 1
    TA = TA + Y
    TA + TO < Ø ?
      F:
        Make a "Move 1"
            TO = TO - X
            "Move 1" the cursor
      T:
        Make a "Move Ø"
            "Move Ø" the cursor
```

Figure 4-9. Nassi-Shneiderman Chart for LINE Routine.

CHAR Logic

Almost everyone is familiar with generating characters with a 5 by 7 matrix of points (see Figure 4-10). However, not too many people realize why 5 by 7 is the smallest common size. The limiting width, of course, is the minimum number of points capable of displaying the three separate parallel lines required for the letters "M" and "W". This sets the minimum width possible to five, but why must seven be the minimum height? The answer is it does not! 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 lower case characters, even larger matrices are required. This results from the greater complexity and finer detail of the lower case 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 using an extra 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 this graphics standard is designed for. Even if just one row of blank points is left between adjacent characters, 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

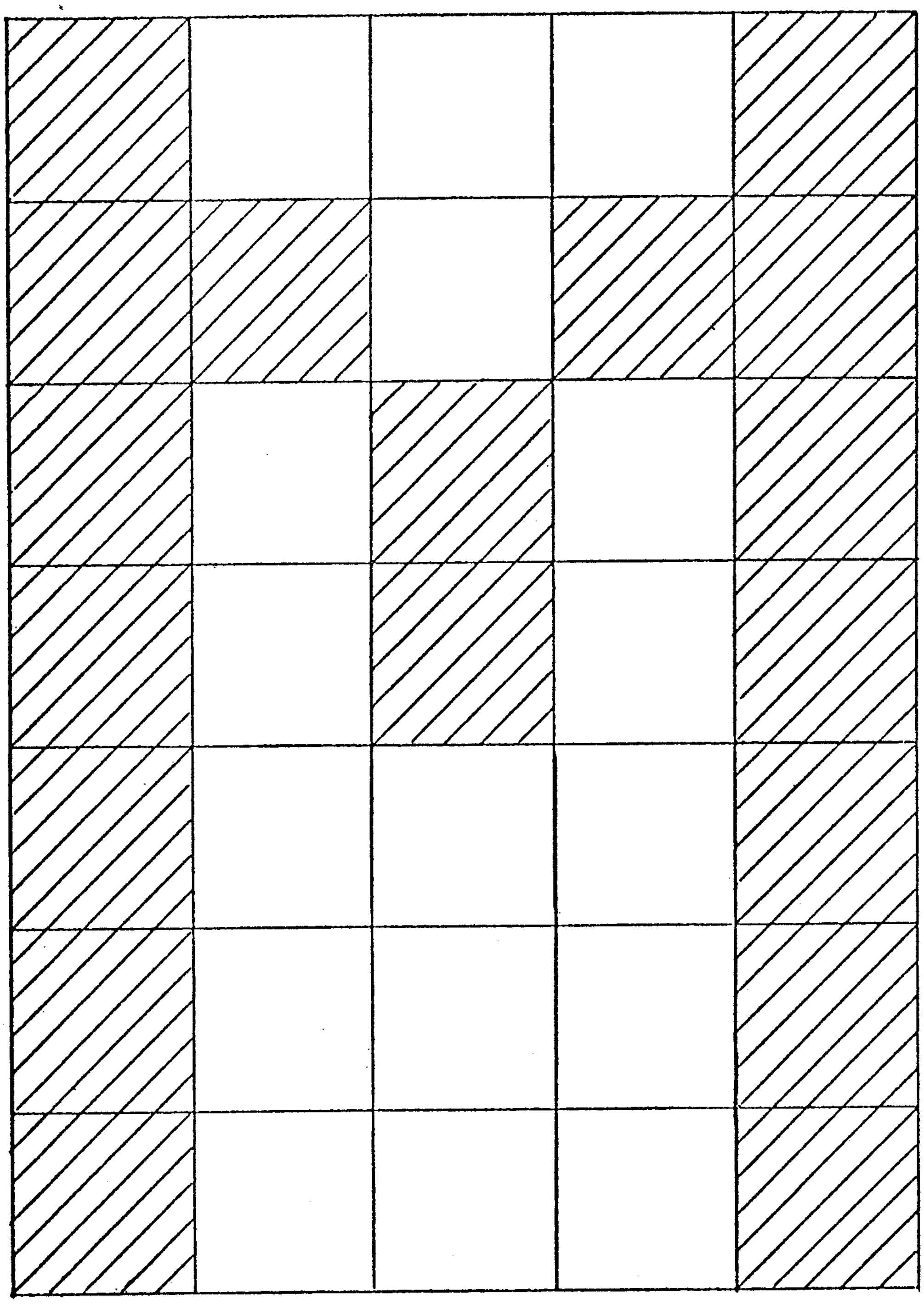


Figure 4-10. Typical Character Generation using 5X7 Matrix.

proportional to the area of the matrix (see Table 4-4).

CHAR SIZE	LOWER CASE	CHAR/LINE (256 by 256)	LINES/PAGE (256 by 256)	MEMORY USED FOR TABLES (bytes)
9 x 11	Yes	25	18	1200
7 x 9	Yes	32	21	864
5 x 7	No	42	32	320
4 x 5*	No	64	32	192

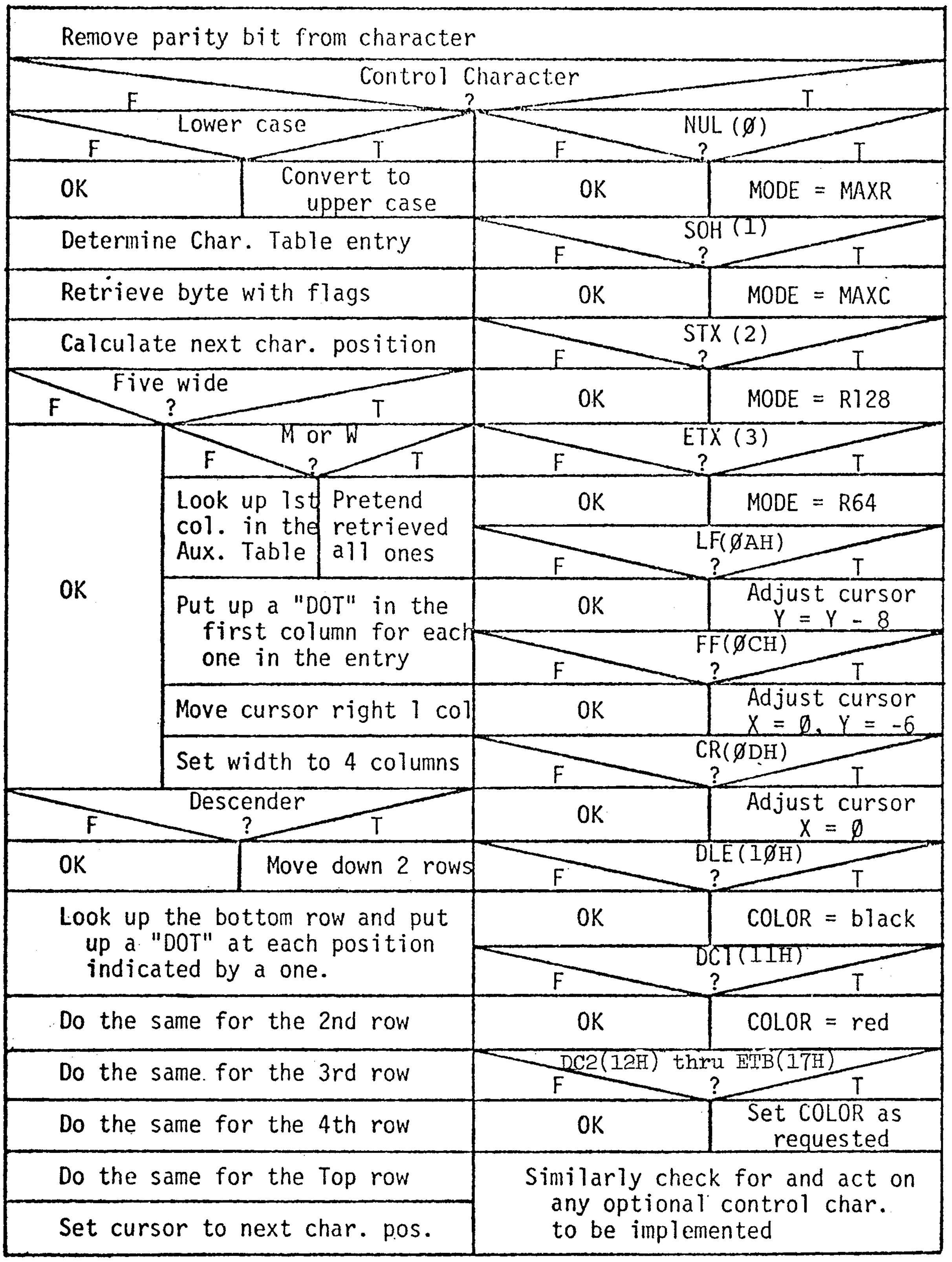
*See text

Table 4-4. Effects of Different-sized Character Matrices.

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

To simplify table lookups and the special handling of five wide characters, three bytes (24 bits) 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 five 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

CHAR



```
Remove parity bit from character
Control Character ?
  F:
    Lower case ?
      F: OK
      T: Convert to upper case
    Determine Char. Table entry
    Retrieve byte with flags
    Calculate next char. position
    Five wide ?
      F: OK
      T:
        M or W ?
          F: Look up 1st col. in the Aux. Table
          T: Pretend retrieved all ones
        Put up a "DOT" in the first column for each one in the entry
        Move cursor right 1 col
        Set width to 4 columns
    Descender ?
      F: OK
      T: Move down 2 rows
    Look up the bottom row and put up a "DOT" at each position indicated by a one.
    Do the same for the 2nd row
    Do the same for the 3rd row
    Do the same for the 4th row
    Do the same for the Top row
    Set cursor to next char. pos.
  T:
    NUL (Ø) ?
      F: OK
      T: MODE = MAXR
    SOH (1) ?
      F: OK
      T: MODE = MAXC
    STX (2) ?
      F: OK
      T: MODE = R128
    ETX (3) ?
      F: OK
      T: MODE = R64
    LF(ØAH) ?
      F: OK
      T: Adjust cursor Y = Y - 8
    FF(ØCH) ?
      F: OK
      T: Adjust cursor X = Ø, Y = -6
    CR(ØDH) ?
      F: OK
      T: Adjust cursor X = Ø
    DLE(1ØH) ?
      F: OK
      T: COLOR = black
    DC1(11H) ?
      F: OK
      T: COLOR = red
    DC2(12H) thru ETB(17H) ?
      F: OK
      T: Set COLOR as requested
    Similarly check for and act on any optional control char. to be implemented
```

Figure 4-11. Nassi-Shneiderman Chart for CHAR Routine.

are displayed two positions lower than their matrices indicate. Each character therefore is displayed in an n by 7 display area where n ranges from 2 to 5.

The basic character generation algorithm is applicable to any sized 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 lower case, it is necessary to shift lower case characters to their upper case 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, the table's physical contents 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 added to the address of the first character in the table 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. Since only the points actually making up the character are affected, background data is not erased and an overprint results.

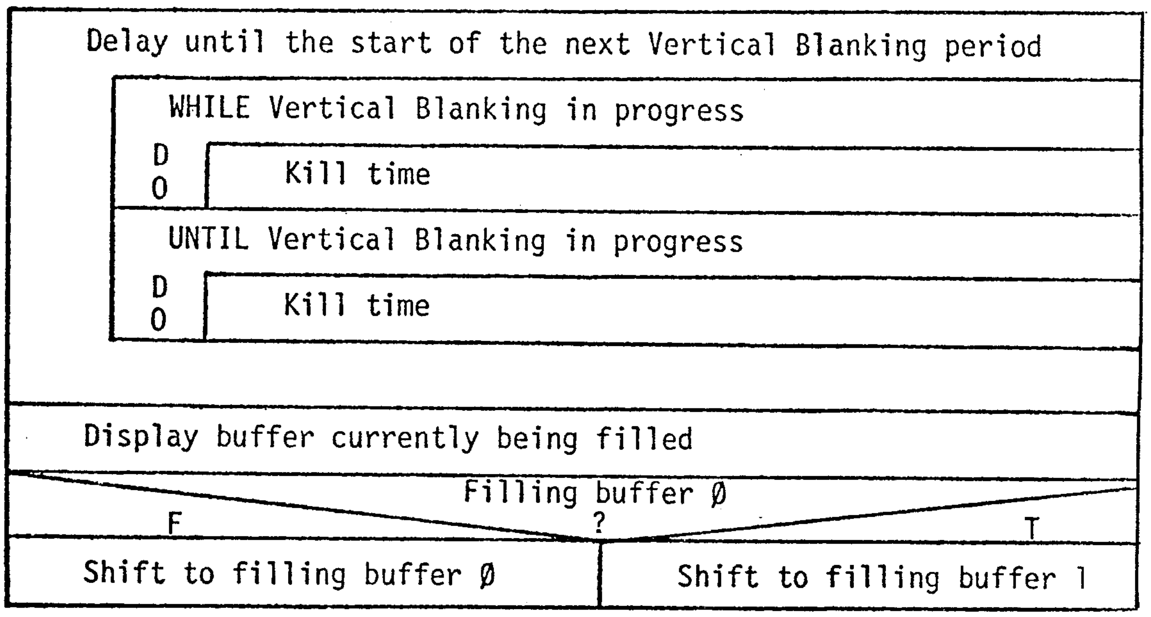
Control characters are handled separately. Mode and color changes will depend on a special portion of the CHAR routine. Since these will be overly hardware dependent, their implementation is left as an exercise to the reader. Carriage control characters modify the cursor position without otherwise affecting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

ANIMAT Logic

The first requirement of the ANIMAT logic is to wait for vertical blanking of the TV screen to start. Most displays provide an input port with a status bit indicating when vertical blanking is in progress. By delaying first until the status bit indicates normal scan, and 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 whose refresh buffer locations are changeable 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 being displayed (first call to ANIMAT after an INITG) or one buffer is being filled while another is being displayed (all subsequent calls to ANIMAT).

ANIMAT



```
Delay until the start of the next Vertical Blanking period
    WHILE Vertical Blanking in progress
    DO
        Kill time
    UNTIL Vertical Blanking in progress
    DO
        Kill time
Display buffer currently being filled
Filling buffer Ø ?
    F: Shift to filling buffer Ø
    T: Shift to filling buffer 1
```

Figure 4-12. Nassi-Shneiderman Chart for ANIMAT Routine.

Sample Implementation of the Algorithms

An implementation of the Z80 Assembly Language Protocol for use with the CROMEMCO DAZZLER illustrates how the algorithms and standards presented in the first half of this chapter translate into working software. Except for a few instances where the architecture of the Z80 or DAZZLER allowed substantial simplification, the program code corresponds exactly to the Nassi-Shneiderman charts. 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. Also, throughout the following, the DAZZLER is assumed to be jumpered to use ports 0EH and 0FH.

Z80/DAZZLER INITG ROUTINE

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 200H (512D). The algorithm used depends on the GRAPH routine being called. For example, GRAPHZ80.REL and GRAPHFOR.REL will place the first picture buffer at the first even 200H boundary following the end of the program, regardless of where the relocatable module is loaded. The second picture buffer immediately follows the first. The refresh buffer address calculated is then stored for use by the other graphics subroutines. Placing all the variables in one 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.

Finally, the DAZZLER is turned on without clearing the picture buffer. This allows the user to exit from a graphics program and then be able to re-enter it without fear of the picture buffer automatically being cleared. The high byte of the refresh buffer address is retrieved from memory and rotated into the bit position expected by the DAZZLER. An OUT instruction with the high bit set actually starts the display. The last step before restoring register values is to load the appropriate parameter description into HL. The value 8AFCH indicates double buffered animation is available, MAXC mode has 15 colors and 64 by 64 resolution, the display is in color, and MAXR mode has 1 color and 128 by 128 resolution.

Z80/DAZZLER PAGE ROUTINE

The PAGE routine takes advantage of the hardware requirement that refresh buffers exist only on 512 boundaries and are 2K bytes long. The low byte of the address is used for a free zero, while the whole HL register is incremented until H corresponds to the high byte of the first address beyond the buffer.

Z80/DAZZLER CURSOR ROUTINE

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 SCALE with the coordinates presented and save the scaled result as the new software cursor position(XPOS,YPOS).

Some cleverness was used in formatting the MODE byte to efficiently indicate the desired mode. The numeric value associated with the mode is rotated right one bit position. The resultant value can be incremented up to 254 times and still remain negative if in MAXC or R64 mode, 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 divides X and Y by two, checks to see if R128 or MAXR is selected, and divides again if they are not.

Z80/DAZZLER DOT ROUTINE

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 two, 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. (Refer to the DAZZLER manual.)

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. Once a point has been uniquely determined, a pixel is plotted and the routine returns, restoring all registers. DOT's register restore sequence is also used by LINE and CHAR. All plotting on the screen is done by the DOT routine. Both LINE and CHAR perform plotting by multiply-calling the DOT subroutine.

Z80/DAZZLER LINE ROUTINE

The LINE routine is almost a block for block encoding of the LINE algorithm outlined in the first part of this chapter. The variable name correspondence table (Table 4-5) 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 since they represent the product of two eight-bit numbers. 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.

Z80 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"

Table 4-5. Variable Name Definitions for LINE.

The cursor adjustments required for a "move 0" and a "move 1" are found through a look-up table. 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 four (two shift lefts) 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.

Z80/DAZZLER CHAR ROUTINE

The CHAR routine, with the exception of control character processing, also follows its Nassi-Shneiderman chart (Figure 4-11) rigorously. The major change has been to convert the string of IFs used for control character processing to a SELECT construct. 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-Shneiderman 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 register-restoring routine onto the top of the stack, the RET instruction will jump to the restoring sequence from DOT, restore all registers, and then return to the calling program, regardless of where the RET is encountered.

The character matrix table is indexed by ASCII value minus 32, i.e. the first entry is a blank. Since each entry is three bytes long, the index must be multiplied by three to get the offset into the table. The first four bits of the first byte of each entry contain all the flags describing the character. The width bits are masked off and the cursor value for the next character position is calculated. If the width is six (includes a blank pixel between characters), the special subroutine to generate the first column of a five column wide character is executed. The descender indicator flag is then checked and the cursor 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 two through five instead of the normal one through four.

Control character handling proceeds in three phases. Phase one 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 two is entered if the control character is not a mode control. This is an individual check for each of the carriage control characters and the specially-defined characters (see Chapter 3 of this part). Note that to get to the top line, form feed must determine what resolution is in use. Phase three, 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 zeroes. If any other color, a check is made to determine if the DAZZLER is in a color-supporting mode (MAXC or R64). If not, the COLOR byte 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 low three bits of the color-select character.

Z80/DAZZLER ANIMAT ROUTINE

The ANIMAT routine's implementation is not complicated and has already been described well. The flag byte ANIM indicates whether the first 2K buffer or the second 2K 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 (undesirable) result, of course, would be that the one buffer was never displayed.

Chapter 5: DEMONSTRATION PROGRAMS AND CONCLUSION

Several demonstration programs have been provided on the DGR diskettes in source-file form to illustrate the correct methods of interface to the various graphics programs. These are briefly discussed in the following sections.

Sample Z80 Assembly Language Program

The program GDEMO.Z80 is provided to illustrate the use of the various original subroutines of the graphics standard. The file GDEMO.REL is its assembled relocatable object code module. This program is run by using LINK.COM to link and load it into memory along with GRAPHZ80.REL. The command line to be used is:

```
LINK GDEMO,GRAPHZ80/G
```

Once the program has been loaded, execution will begin. Execution may be terminated at any time (and control returned to CDOS) by typing CTRL-C on the console. Execution may be stopped at any time to allow viewing of the display by typing CTRL-S. Typing CTRL-S a second time will restart the sequence. The sequence runs continuously until terminated. More information on creating, assembling, and linking user-Z80 files may be found in the Macro Assembler Instruction Manual. The GDEMO program contains four independent demonstrations. EXTernals are used to allow mnemonic references to the standard protocol's ENTRY addresses.

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, the variable delay utility subroutine PAUSE is used to give time to observe the display.

The second demonstration tests the generation of all 64 of the upper case ASCII characters. Again, advantage is taken of the lowest resolution mode to display larger characters. The 64 characters are drawn eight times, once in each color, to show the ability to dynamically vary the display. On the last iteration, they 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 while executing line generation. To avoid claiming "Full Color Control" on a monochrome display (since this is a "universal" graphics routine), the color bit in MAXCD is tested. MCCOLS is then checked to see how many colors or gray 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 shift to mode RXXX after mode R64 is used is to select the DAZZLER's sixteen-level gray scale mode.

The final demonstration is a short animation sequence. The header is inserted in both buffers after they have been cleared using PAGE. The algorithm used to animate the figure will work with either double buffered displays or selectively erasable displays. For the DAZZLER the figure is backed up one step and drawn in black to erase it from the nondisplaying buffer (PAGE would require too much time and would also 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 repeated until the screen has been traversed.

Sample FORTRAN Graphics Program

The sample FORTRAN program BARPLOT.FOR is provided as a simple example of interfacing FORTRAN to the Graphics routines. The file BARPLOT.REL is its associated relocatable object file (already compiled). Since a compiled FORTRAN program is very similar in structure and operation to an assembled Z80 program, the procedure for loading and running BARPLOT is exactly similar to that described in the previous section. Type:

```
LINK BARPLOT,GRAPHFOR/G
```

to run the program. This program will demonstrate how to use the Dazzler Graphics routines to draw simple bar graphs using FORTRAN.

Sample BASIC Graphics Programs

Because of the greater complexity (and hence, versatility) of the BASIC-Graphics interface, two demonstration programs are provided on the DGR disks, one simple and one more complex. The simple example is BOUNCE.LIS. To run this program, type "GRAPHX" to load and execute both GRAPHX.COM and BASIC as described in Chapter 3. (Be sure when doing this that either the current drive or drive-A contains the program BASIC.COM.) Now, from BASIC type:

```
ENTER BOUNCE.LIS
```

Next LIST the program. By reading the statements along with the REMarks in the program, you can gain some knowledge as to the correct and easiest ways to design BASIC-Graphics programs. Now type RUN to begin execution. This program will first draw a blue box and then "bounce" a red ball around its interior. Note the subroutine to produce a "plop" sound from the CROMEMCO Joysticks if they are connected.

When you are satisfied that you understand the operation of BOUNCE.LIS, press the ESCape key to stop execution, type CLOSE or CLOSE\1\ to close the DG driver, type SCR to scratch the present program, and finally type:

```
ENTER ROTATE.LIS
```

This is the more complex example mentioned earlier. It is designed to read a data file from the disk, create a picture of a geometrical figure, and then rotate this figure in 3-D. Upon typing RUN, the program will prompt you. It will first ask for the name of a DATA file. Two are supplied on this disk; therefore, type either "CUBE" or "TETRAHED". Type ESC at any time to stop the rotation; then type "E" to return to BASIC enter-mode. You may create your own DATA files for use with ROTATE. Use EDIT, the CDOS Text Editor, to enter the coordinates of the points to be connected, remembering to include all lines of the figure. Also, remember to name your data files with the .DAT extension. The orientation of the axes of the 3-D figure is as follows:

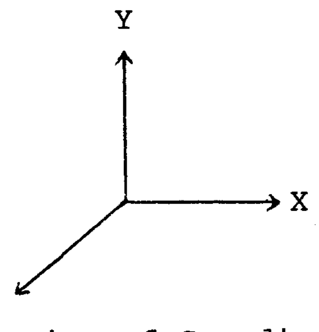


Figure 5-1. Orientation of Coordinate Axes for ROTATE.LIS.

Conclusion

The availablility 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, as we have seen, 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. It is hoped that the programs provided in this package will not only stimulate an interest in graphics as a method of displaying computer output but will also encourage the design and exchange of graphics programs among users.

PART II - INTERACTIVE DAZZLER PLOTTING

Chapter 6: LOADING AND EXECUTING DAZZPLOT

Dazzleplot is a program which provides one with fast and simple control over the Dazzler hardware for many kinds of plotting and drawing. This software does all of the translation between simple up/down, left/right instructions and the complex memory storage required by the Dazzler. The program also performs a number of special purpose functions such as clearing the screen, drawing coordinate axes with tic marks, changing among several different picture buffers, selecting high/low resolution and intensity, providing complete cursor control, and a number of others. Dazzleplot is stored on the disk under the name DAZZPLOT.COM. Since it is a .COM file, its name may be typed directly from CDOS and it will load from the disk and start executing at 100H. Input to DAZZPLOT may be provided from two sources: input from the CROMEMCO Joysticks and input from the console keyboard. These are described in detail in Chapter 7. (The first of these requires a pair of CROMEMCO JS-1 Joysticks interfaced through a D+7A analog-to-digital board.)

The hardware requirements to run Dazzleplot are not as extensive as for the Dazzler Graphics package. 16K of memory and the Dazzler itself are sufficient to operate DAZZPLOT. However, to take advantage of all 16 of the available picture buffers, 48K of memory is required. Dazzleplot is loaded and begins execution at 100H as mentioned; however, it then immediately moves itself in memory to just below the operating system (CDOS) and executes from this location. It then clears picture buffers 0 (location 200H) and 1 (location A00H), as this is where DAZZPLOT itself was originally loaded. The reason for this move to high memory is the following: From locations 100H to 200H in memory, DAZZPLOT stores a short program which will turn on picture buffer 0. Thus, at any point during execution of Dazzleplot, you may save what you are doing on disk by performing the following steps:

1) Use the CTRL-E or exchange command to move the picture buffer which you are currently using to the buffer 0 position.
2) Press the ESCape key to exit from Dazzleplot and return to CDOS.
3) Type the command:
   ```
   SAVE filename.COM 9
   ```
   where filename is replaced by a name of your choice and where 9 is the number of 256-byte blocks required to save picture buffer 0 on the disk.

To reload this buffer and display it at any time in the future, simply type the filename (since it is a .COM file). The picture will be loaded into memory and the Dazzler will be turned on at the correct location by a short program at 100H. To return to DAZZPLOT simply type "D"; however, <u>be</u> <u>sure</u> DAZZPLOT actually resides in memory before doing this.

If Dazzleplot is executed when there is insufficient memory in the user area for all 16 picture buffers (for example, with a 32K CDOS), it will automatically print an error message if one of the illegal buffers is selected by use of the CTRL-W command. Each Dazzler picture buffer requires 2K of RAM. The starting addresses of these are listed in Table 7-5.

Chapter 7: DAZZLEPLOT CONTROL COMMANDS

There are quite a number of input commands recognized by Dazzleplot and for ease of explanation they can be broken into three groups. Each of these groups of commands is described separately in this chapter. The first group of numeric character inputs control the movement of the cursor or actual plotting or drawing process. Each time one of the numeric characters is entered the current cursor location is moved accordingly and the current color or black and white intensity is deposited at this new location. The color and/or intensity is specified by the second group of input alphabetic commands. The color can be specified or changed at any time and remains the same until a new color is input. Thus, after specifying a color or intensity several movements can be input and this same color will be used to draw the line specified by these movement commands. The third group of input commands are control characters. These are used to alter the basic operating characteristics of Dazzleplot such as high/low resolution, color/black and white mode, cursor on/off/blink, reverse background on/off, and several others.

Inputs from any of these three groups may be freely mixed at any time with predictable, although sometimes unusual, results. For example, if you have drawn some figure in high resolution mode and then enter CTRL-L (change to low resolution), the resulting drawing may appear rather strange. While in the high resolution mode all points are the same color (128 by 128 grid). When you change to low resolution, each point can be any of 16 colors or intensities (64 by 64 grid).

One of the CROMEMCO JS-1 Joysticks may also be used as input to Dazzleplot. This actually provides for the most interactive method of control of cursor movement. When using Joystick as an input to this program, the Joystick movement replaces (actually duplicates) the numeric input commands. When you move the Joystick up the cursor moves up, and when you move it to the right the cursor moves right, and so on for all directions of movement. As you move the cursor, the specified color or intensity will be deposited in the picture storage area. Also, it is important to note that the speed of cursor movement is dependent upon how far the Joystick is displaced from its zero or rest position. (This zero position may have to be adjusted initially by use of the vertical and horizontal trimmer pots which are mounted on the face of the Joystick. If the cursor is moving across the screen on its own when you first execute DAZZPLOT, simply adjust these for zero movement.)

The four buttons on the JS-1 console have been assigned four often used control functions. These predefined functions are as follows:

Button 1 - CTRL-C	Center cursor without screen clear.
Button 2 - CTRL-O	Move cursor to lower left corner (X=0,Y=0).
Button 3 - CTRL-A	Draw coordinate axes.
Button 4 - CTRL-P	Print out X and Y coordinates of cursor.

These functions are provided as a convenience to the user doing interactive plotting with the Joystick. Note that their functions may be duplicated at the keyboard.

Cursor Movement Controls

The movement of the cursor and thus all drawing or plotting is controlled with eight numeric input commands (ASCII '1'-'4' and '6'-'9'). Entry of any one of these numeric commands moves the cursor by exactly one position in the appropriate direction. Coincident with such a move is the depositing of a color or intensity in this new location. The choice of which color or intensity is to be deposited is determined by the last alphabetic color command entered prior to the numeric move command. The allowed numeric input commands and their meanings are defined as follows:

ASCII	MOVEMENT
8	Up
2	Down
4	Left
6	Right
9	Up and Right
3	Down and Right
1	Down and Left
7	Up and Left
5	No Effect

Table 7-1. Console Cursor Controls.

Note that these numeric command assignments correspond in direction to imaginary pointers which are part of the adding machine type of number keypad found on many terminals such as the CROMEMCO 3100-series CRT's (see Figure 7-1). For example, the command 9 means move towards the upper-right, and the number 9 key is in the upper-right corner of the keypad.

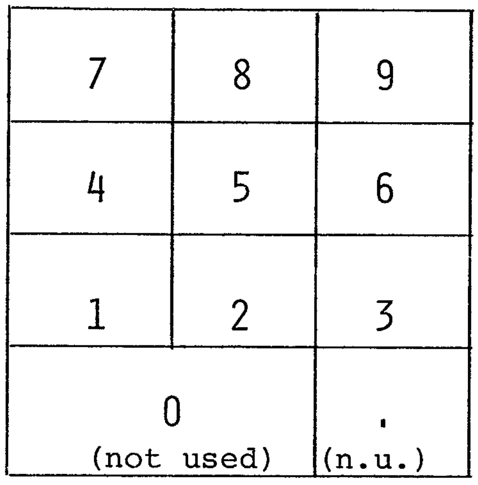


Figure 7-1. Standard Adding Machine Keypad.

Color and Intensity Controls

The color and/or intensity of points drawn with Dazzleplot are controlled through the use of this set of alphabetic color commands. Executing one of these commands changes a color/ intensity flag which will be used to load a picture storage location upon entry of the next move command (whether numeric from keyboard or a displacement from Joystick). When Dazzleplot is in the high resolution mode (128 by 128 grid), this color flag is used to specify the color of the entire plot because in this mode all points can have only one of two values, on or off. However, in the low resolution mode (64 by 64 grid) this color flag is used to load the appropriate 4-bit nybble into the picture storage area. In this mode each point can have one of 16 intensities or colors. Thus, in low resolution you can draw multicolored plots, diagrams, or any other kind of picture.

The exact 4-bit combination which is loaded into the color flag is specified by a set of eight color commands which are used together with two intensity commands. Since the two intensity commands are independent of the eight color commands, a total of 16 possible combinations are allowed. It is important to remember that the two intensity commands (H and L) can be changed without altering the color command and vice versa. What this means is that you must set both a color and an intensity before you can know what will be deposited at the next move location. Points will continue to be plotted in this color or intensity until a new one is selected. The exact commands and their effect are given in the following table.

ASCII	COLOR	GRAY SHADE	
'N'	No Change or Black	Black	Gray 8
'R'	Red	Gray 1	Gray 9
'G'	Green	Gray 2	Gray 10
'Y'	Yellow	Gray 3	Gray 11
'B'	Blue	Gray 4	Gray 12
'M'	Magenta	Gray 5	Gray 13
'C'	Cyan	Gray 6	Gray 14
'W'	White	Gray 7	White
'H'	High Intensity		X
'L'	Low Intensity	X	

Table 7-2. Color and Intensity Controls.

Program Control Commands

Dazzleplot recognizes a number of different control commands which can alter most of the operating characteristics of the program itself as well as the operation of the Dazzler hardware. These commands can be divided into five major areas of operation and are so-outlined in Table 7-3. Their detailed descriptions are given in alphabetic order in the following text.

Control-A (ASCII 'SOH' = 01H)

This command draws a pair of X and Y axes to be used for subsequent plots. When in the normal background mode, these axes are white on black and when in the reverse mode, they are black on white. The two axes intersect at the current cursor location as it was defined immediately prior to entering this command. The axes have "tic" marks at specified intervals to indicate X and Y values quantitatively. As initialized there is a tic mark every tenth point from the origin along each axis. This setting may be changed by the CTRL-I command.

Control-B (ASCII 'STX' = 02H)

This command sets the cursor in a blinking mode so that it is always visible as a blinking white cross. Otherwise, operation in this mode is identical to operation in the cursor-on mode (CTRL-N).

Control-C (ASCII 'ETX' = 03H)

This command centers the cursor on the screen (X=127,Y=127) without clearing or altering the display.

Control-E (ASCII 'ENG' = 05H)

This command is used to exchange the currently selected picture buffer with the "base" or zeroth buffer. This is useful

1. Memory Control

CTRL-E	Exchange one of the alternate picture buffers with the base picture buffer at location 200H.
CTRL-S	Select alternate picture buffer (up to 16 possible).

2. Picture Characteristics Control

CTRL-H	Set to High resolution mode (128x128).
CTRL-L	Set to Low resolution mode (64x64).
CTRL-X	Set Dazzler to Black/White mode.
CTRL-Z	Set Dazzler to Color mode.
CTRL-Q	Select normal picture background.
CTRL-R	Select complementary (Reverse) picture background.

3. Cursor Display and Location Control

CTRL-B	Select cursor-blinking.
CTRL-N	Select cursor-always-on.
CTRL-F	Select cursor-always-off.
CTRL-C	Center cursor on screen with no screen-clear.
CTRL-K	Center cursor and clear (remember Klear) picture buffer (cleared to white if complementary background has been selected).
CTRL-O	Move cursor to Origin or lower-left corner of screen; (X,Y)=(0,0).
CTRL-P	Print the current cursor Position (X,Y) on the console. These values are in the range 0-255 as they were for the Dazzler Graphics routines.

4. X and Y Axes Control

CTRL-A	Draw a pair of X-Y Axes centered at the current cursor location and having tic-marks at the current spacing.
CTRL-I	Change tic-mark Interval spacing (0 through 15 possible).

5. Program Entry and Exit Control

CTRL-J	Re-initialize DAZZPLOT flags and parameters.
ESC	Turn off Dazzler, exit Dazzleplot, and return to CDOS.

Table 7-3. Dazzleplot Control Characters and Their Functions.

if it is desired to save a particular picture on the disk--the buffer can first be exchanged into the base buffer, then saved by a command of the format:

```
SAVE [D:]filename.[ext] 9
```

If the optional 3-letter extension .COM is used, this picture may be viewed at any later time simply by typing the filename.

Because DAZZPLOT clears the base picture buffer (as well as the next following buffer--see Chapter 6), the exchange command may also be used to transfer either of these two buffers elsewhere in memory so that DAZZPLOT may at a later time be re-loaded from the disk. Note that following execution of CTRL-E the current buffer remains selected, although its contents will have changed.

Control-F (ASCII 'ACK' = 06H)

This command turns off the cursor and makes it invisible on the display. Even though invisible, a current cursor position is always maintained by Dazzleplot and operation is the same as if a cursor were visible. In all cases except the No Change condition (combination 'L' and 'N' commands) a visible line can be seen to appear when move instructions are entered. In these cases the next point to be drawn will always be adjacent to the last one and in the direction specified by the next move instruction.

Control-H (ASCII 'BS' = 08H)

This command sets the Dazzler in high resolution mode. In this mode the display is made of 128 by 128 points, each of which can have only one of two possible values, on or off. The color currently selected will determine the color of the entire picture when in this mode.

Control-I (ASCII 'HT' = 09H)

This command is always immediately followed by a single key input from the keyboard, whose possible values are summarized in Table 7-4. The command together with the single input immediately following it specifies the interval between tic marks on the (X,Y) axes drawn by CTRL-A. The interval between tic marks can be set to any integer value between 0 and 15 by following Control-I with an ASCII '0' through 'F', corresponding to the hex numbers 0-F. For example, the default interval of 10 can be restored after some change by typing Control-I immediately followed by the letter 'A'.

Hex Number	Tic Mark Interval
0	0 (no tic marks)
1	1 (solid tic marks)
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10 (default value)
B	11
C	12
D	13
E	14
F	15

Table 7-4. Tic Mark Intervals for CTRL-I Command.

Control-J (ASCII 'LF' = 0AH)

This command initializes the critical flags for Dazzleplot and is always called upon program entry. However, if you wish to reinitialize Dazzleplot after some time in use you need only enter Control-J. Reinitializing performs the following functions: the display is returned to high intensity white, high resolution color mode with a blinking cursor; the tic mark spacing is internally reset to the default value of 10, although this will not be visible until the next time CTRL-A is used; the reverse background flag is set to normal so that subsequent plotting will be done with the normal colors (not their complements); and picture buffer 0 is again selected as the current buffer.

Control-K (ASCII 'VT' = 0BH)

This command actually performs two functions: first, it clears the display memory of the currently selected buffer, and then it defines the current cursor position to be in the center of the display. When in the normal background mode, the display is cleared to black (memory = 0H). When in the reverse background mode, this command clears the display to white (memory = FFH).

Control-L (ASCII 'FF' = 0CH)

This command sets the Dazzler in low resolution mode. In this mode the display is made of 64 by 64 points, each of which can have either any one of 16 possible colors or shades of gray. The specific colors which are allowed are listed together in table 7-2.

Control-N (ASCII 'SO' = 0EH)

This command turns on the cursor and makes it visible on the display at all times. When in this mode, the cursor is represented by a white cross on the display which moves so that the point in the center of the cross is always the point which has most recently been loaded or drawn. Thus, the next point to be drawn will be adjacent to this and in the direction specified by the next move instruction. When in the reverse background mode, the cursor is a black cross.

Control-O (ASCII 'SI' = 0FH)

This command moves the current cursor location to the lower left hand corner of the display. This location is defined as X=0 and Y=0 and is commonly used as the origin for plots (especially plots occurring entirely in quadrant I of the Cartesian plane). For reference the maximum values of X and Y are (X=255,Y=255). Points plotted are automatically scaled to this even if the resolution of the hardware device (in this case the Dazzler) is less.

Control-P (ASCII 'DLE' = 10H)

This command calculates and prints out on the console the X and Y values of the current cursor location. These values are always in the range 0-255 regardless of the resolution currently selected.

Control-Q (ASCII 'DC1' = 11H)

This command sets a normal background for drawing. This means that for high resolution you will draw white or color lines on a black background. For low resolution the actual colors given in table 7-2 are loaded into memory when drawing.

Control-R (ASCII 'DC2' = 12H)

This command produces a reverse background condition with respect to the condition described for Control-Q. When in high resolution mode, this command produces a black drawing on a white or color background. In low resolution mode the complement of the selected color in table 7-2 is loaded into memory when drawing. Note that the effect of this command is not apparent until plotting is actually done. It has <u>no</u> effect on alreay-plotted points in the picture buffer.

Control-S (ASCII 'DC3' = 13H)

This command is always immediately followed by a single key input from the keyboard, whose possible values are summarized in Table 7-5. The command is used to change the memory location of the picture storage area to any one of 16 possible addresses; i.e., it is used to select one of 16 possible picture buffers. The allowed addresses are also given in Table 7-5. DAZZPLOT always executes from the highest possible portion of the CDOS user area, and the picture buffers are located below that. Therefore, if a smaller CDOS size is used (e.g., 32K), there will not be sufficient memory for all 16 picture buffers. The CTRL-S command automatically checks for this and will print a warning to the console if an out of range buffer is selected.

Hex Number	Picture Buffer	Memory Start Addresses
0	First	200H
1	Second	A00H
2	Third	1200H
3	Fourth	1A00H
4	Fifth	2200H
5	Sixth	2A00H
6	Seventh	3200H
7	Eighth	3A00H
8	Ninth	4200H
9	Tenth	4A00H
A	Eleventh	5200H
B	Twelfth	5A00H
C	Thirteenth	6200H
D	Fourteenth	6A00H
E	Fifteenth	7200H
F	Sixteenth	7A00H

Table 7-5. Numbers and Addresses of the 16 Possible Picture Buffers.

Control-X (ASCII 'CAN' = 18H)

This command puts the Dazzler in black and white mode.

When in high resolution mode, this command will produce binary black and white drawings or plots. When in low resolution, this command will produce drawings with up to 16 different shades of gray. This mode can be used to display or reproduce photographic-like images or actual digitized photographs. In this mode the gray level intensity is determined as described in table 7-2 for levels of 0 to 15.

Control-Z (ASCII 'SUB' = 1AH)

This command puts the Dazzler in color mode. In high resolution mode this results in a single-color plot. In low resolution mode this allows you to draw or plot multicolor pictures with up to 16 different colors (8 each for both high and low intensities--see Table 7-2).

ESCape Command (ASCII 'ESC' = 1BH)

This command is used to return to CDOS. It will first turn off the Dazzler and then execute a jump to location 0, the warm start for CDOS. Note that you may return to DAZZPLOT (if ESC was pressed accidentally) by using RDOS to "G100" and then typing "D" for DAZZPLOT in response to the console message.

Chapter 8: CONCLUSION AND EXAMPLE OF USE OF DAZZPLOT

The example following is designed to acquaint the beginning user of Dazzler plotting routines with some of the color and motion changes possible. Simply perform the numbered steps below:

1) Insert your DGR diskette in drive-A of the computer and boot-up CDOS.
2) Now type the word "DAZZPLOT" to load and execute the program.
3) Press CTRL-K to clear the picture buffer (after the sign-on message has been printed).
4) Press CTRL-A to plot X-Y axes centered at X=127, Y=127.
5) Now press CTRL-W immediately followed by the character "1" to select picture buffer 1 (press CTRL-W,0 at any time to return to the first buffer).
6) Press CTRL-L to select low resolution (64x64) followed by CTRL-K to clear the picture buffer.
7) Press CTRL-O to move the cursor to the origin.
8) Press CTRL-A to plot the axes centered at the origin (X=0,Y=0).
9) Now press CTRL-C to again center the cursor. Press "G" to select the color green.
10) Press the "9" key 6 times to move the cursor in an upper-right-hand direction.
11) Press "M" for magenta and then the "2" key 25 times to move the cursor downward.
12) Press "C" for cyan and then the "4" key 7 times followed by the "6" key 14 times.
13) Now press CTRL-F to turn off the blinking cursor and

you should have a large number "1" plotted in several colors on your TV screen.

14) This concludes the example. Press "ESC" (escape key) to abort the program and return to CDOS.

This chapter concludes both Part II, the section on the program Dazzleplot, and also the Dazzler Graphics manual. It is hoped that the programs provided in this package and described here will stimulate an interest in graphics software with an emphasis on its serious application. There are many practical uses for graphics--from visual display of real-time control devices to 3-D plotting as an educational and visual aid. The availability of graphics software will become increasingly important as the resolution and color distinction of graphics output devices gets higher. Persons who make interesting or novel application of the programs contained herein are encouraged to share their ideas and input with CROMEMCO.