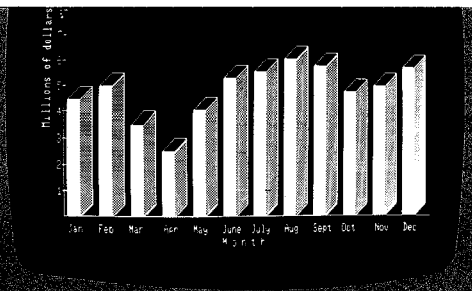
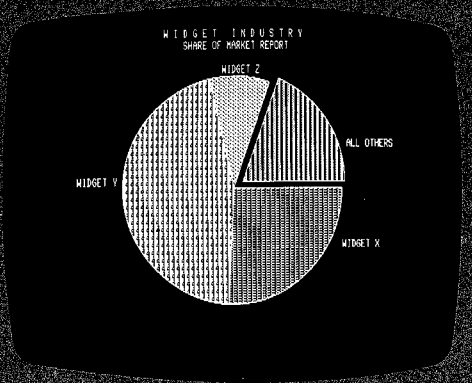
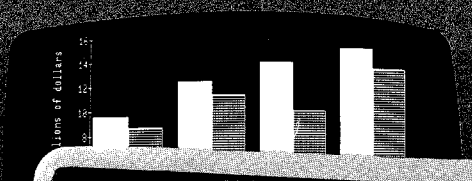
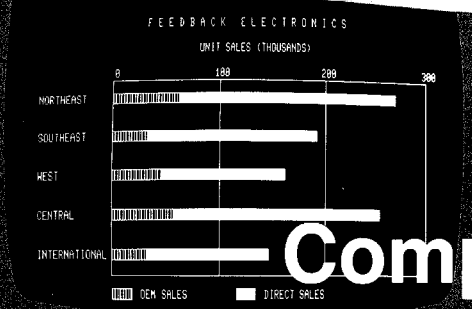




Radio Shack®

"The biggest name in little computers"





TRS-80®
Computer Graphics

CUSTOM MANUFACTURED IN THE USA BY RADIO SHACK, A DIVISION OF TANDY CORPORATION

A Special Note on Model III Computer Graphics...

Be sure to use the GCLS command ("clear the Graphics Screen") at TRSDOS READY when you first turn on your computer. Otherwise, random graphics may appear on the Screen.

Thank You

Radio Shack®
A DIVISION OF TANDY CORPORATION
FORT WORTH, TEXAS 76102

8759223

TERMS AND CONDITIONS OF SALE AND LICENSE OF RADIO SHACK COMPUTER EQUIPMENT AND SOFTWARE PURCHASED FROM A RADIO SHACK COMPANY-OWNED COMPUTER CENTER, RETAIL STORE OR FROM A RADIO SHACK FRANCHISEE OR DEALER AT ITS AUTHORIZED LOCATION

LIMITED WARRANTY

I. CUSTOMER OBLIGATIONS

A. CUSTOMER assumes full responsibility that this Radio Shack computer hardware purchased (the "Equipment"), and any copies of Radio Shack software included with the Equipment or licensed separately (the "Software") meets the specifications, capacity, capabilities, versatility, and other requirements of CUSTOMER.

B. CUSTOMER assumes full responsibility for the condition and effectiveness of the operating environment in which the Equipment and Software are to function, and for its installation.

II. RADIO SHACK LIMITED WARRANTIES AND CONDITIONS OF SALE

A. For a period of ninety (90) calendar days from the date of the Radio Shack sales document received upon purchase of the Equipment, RADIO SHACK warrants to the original CUSTOMER that the Equipment and the medium upon which the Software is stored is free from manufacturing defects. THIS WARRANTY IS ONLY APPLICABLE TO PURCHASES OF RADIO SHACK EQUIPMENT BY THE ORIGINAL CUSTOMER FROM RADIO SHACK COMPANY-OWNED COMPUTER CENTERS, RETAIL STORES AND FROM RADIO SHACK FRANCHISEES AND DEALERS AT ITS AUTHORIZED LOCATION. The warranty is void if the Equipment's case or cabinet has been opened, or if the Equipment or Software has been subjected to improper or abnormal use. If a manufacturing defect is discovered during the stated warranty period, the defective Equipment must be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer for repair, along with a copy of the sales document or lease agreement. The original CUSTOMER'S sole and exclusive remedy in the event of a defect is limited to the correction of the defect by repair, replacement, or refund of the purchase price, at RADIO SHACK'S election and sole expense. RADIO SHACK has no obligation to replace or repair expendable items.

B. RADIO SHACK makes no warranty as to the design, capability, capacity, or suitability for use of the Software, except as provided in this paragraph. Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon license of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer along with the sales document.

C. Except as provided herein no employee, agent, franchisee, dealer or other person is authorized to give any warranties of any nature on behalf of RADIO SHACK.

D. Except as provided herein, **RADIO SHACK MAKES NO WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.**

E. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation(s) may not apply to CUSTOMER.

III. LIMITATION OF LIABILITY

A. EXCEPT AS PROVIDED HEREIN, RADIO SHACK SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO CUSTOMER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY "EQUIPMENT" OR "SOFTWARE" SOLD, LEASED, LICENSED OR FURNISHED BY RADIO SHACK, INCLUDING, BUT NOT LIMITED TO, ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THE "EQUIPMENT" OR "SOFTWARE". IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS, OR ANY INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR IN ANY MANNER ARISING OUT OF OR CONNECTED WITH THE SALE, LEASE, LICENSE, USE OR ANTICIPATED USE OF THE "EQUIPMENT" OR "SOFTWARE".

NOTWITHSTANDING THE ABOVE LIMITATIONS AND WARRANTIES, RADIO SHACK'S LIABILITY HEREUNDER FOR DAMAGES INCURRED BY CUSTOMER OR OTHERS SHALL NOT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE PARTICULAR "EQUIPMENT" OR "SOFTWARE" INVOLVED.

B. RADIO SHACK shall not be liable for any damages caused by delay in delivering or furnishing Equipment and/or Software.

C. No action arising out of any claimed breach of this Warranty or transactions under this Warranty may be brought more than two (2) years after the cause of action has accrued or more than four (4) years after the date of the Radio Shack sales document for the Equipment or Software, whichever first occurs.

D. Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitation(s) or exclusion(s) may not apply to CUSTOMER.

IV. RADIO SHACK SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the RADIO SHACK Software on **one** computer, subject to the following provisions:

A. Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software.

B. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.

C. CUSTOMER may use Software on one host computer and access that Software through one or more terminals if the Software permits this function.

D. CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on **one** computer and as is specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software.

E. CUSTOMER is permitted to make additional copies of the Software **only** for backup or archival purposes or if additional copies are required in the operation of **one** computer with the Software, but only to the extent the Software allows a backup copy to be made. However, for TRSDOS Software, CUSTOMER is permitted to make a limited number of additional copies for CUSTOMER'S own use.

F. CUSTOMER may resell or distribute unmodified copies of the Software provided CUSTOMER has purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from CUSTOMER.

G. All copyright notices shall be retained on all copies of the Software.

V. APPLICABILITY OF WARRANTY

A. The terms and conditions of this Warranty are applicable as between RADIO SHACK and CUSTOMER to either a sale of the Equipment and/or Software License to CUSTOMER or to a transaction whereby RADIO SHACK sells or conveys such Equipment to a third party for lease to CUSTOMER.

B The limitations of liability and Warranty provisions herein shall inure to the benefit of RADIO SHACK, the author, owner and/or licensor of the Software and any manufacturer of the Equipment sold by RADIO SHACK.

VI. STATE LAW RIGHTS

The warranties granted herein give the **original** CUSTOMER specific legal rights, and the **original** CUSTOMER may have other rights which vary from state to state.

Contents

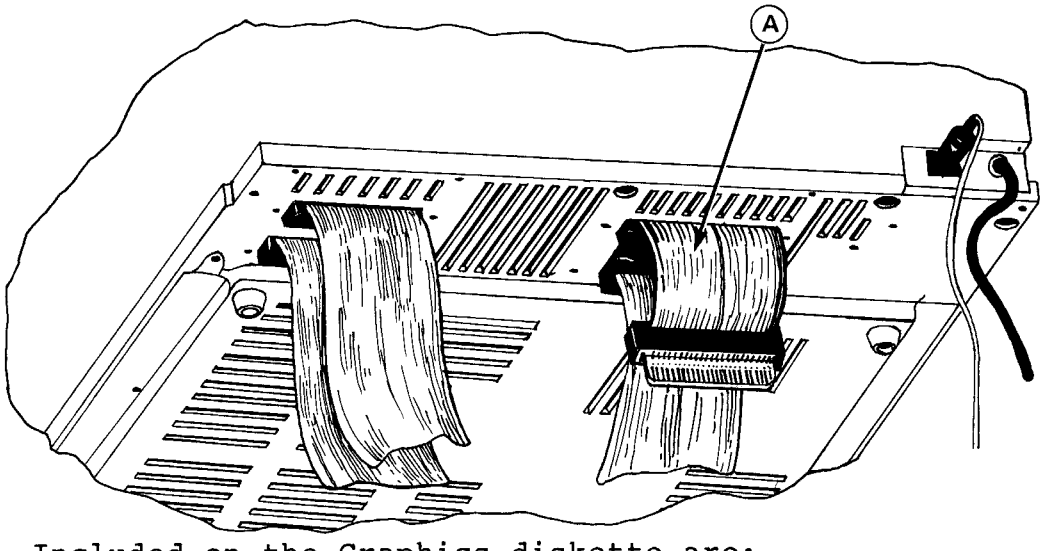
To Our Customers . . .

The TRS-8Ø® Computer Graphics package revolutionizes your Model III by letting you draw intricate displays from simple program instructions. With the highly-defined Graphics Screen, the list of practical applications is nearly endless!

The TRS-8Ø Computer Graphics package includes a:

- Computer Graphics Diskette
- Computer Graphics Operation Manual

However, before you can use this package, your Model III must have 48K of RAM (Random Access Memory) and one disk drive. Your computer must also be modified by a qualified Radio Shack service technician. The only difference you'll notice is a cable which protrudes from the bottom of the Model III case. Do not attempt to disconnect this cable! This cable is provided to allow you to attach peripheral devices (such as a hard disk) to the I/O Bus Jack of the Model III. The cable connector which is attached directly to the I/O Bus Jack (see Point A in the figure below) must be firmly attached for the Computer Graphics package to work.



Included on the Graphics diskette are:

- TRSDOS 1.3
- Disk BASIC
- Graphics BASIC (BASICG)
- Graphics Subroutine Library (GRPLIB)

- Graphics Utilities
- Sample Programs in BASICG and FORTRAN.

To print graphic displays, you can use any Radio Shack printer that has graphic capabilities such as Line Printer VII (26-1167), Line Printer VIII (26-1168), DMP-1ØØ (26-1253), DMP-2ØØ (26-1254), DMP-4ØØ (26-1251), or DMP-5ØØ (26-1252).

You can also utilize the Graphics Subroutine Library with several languages, including, but not limited to FORTRAN (26-22ØØ).

About This Manual . . .

For your convenience, we've divided this manual into five sections plus appendixes:

- Computer Graphics Overview
- Graphics BASIC (BASICG) Language Description
- Graphics Utilities
- FORTRAN Description
- Programming the Graphics Board
- Appendixes

This package contains two separate (but similar) methods for Graphics programming:

- Graphics BASIC (BASICG)
- Graphics Subroutine Library

If you're familiar with Model III TRSDOS™ and BASIC, you should have little trouble in adapting to Graphics BASIC. If you want to review BASIC statements and syntax, see your **Model III Operation and BASIC Language Reference Manual** and **Model III Disk System Owner's Manual.** Then read Chapters 1, 2 and 3, along with Appendixes A, B, E, and F of this manual.

If it's Graphics applications in FORTRAN you're after, refer to the TRS-8Ø FORTRAN manual. Then read Chapters 1, 2, 3, and 4 as well as Appendixes C, D, E, and F of this manual.

Note: This manual is written as a reference manual for the TRS-8Ø Computer Graphics package. It is not intended as a teaching guide for graphics programming.

Notational Conventions

The following conventions are used to show syntax in this manual:

CAPITALS	Any words or characters which are uppercase must be typed in exactly as they appear.
lowercase underline	Fields shown in lowercase underline are variable information that you must substitute a value for.
<ENTER>	Any word or character contained within brackets represents a keyboard key to be pressed.
...	Ellipses indicate that a field entry may be repeated.
filespec	A field shown as filespec indicates a standard TRSDOS file specification of the form: filename/ext.password:d Note that with TRSDOS 1.3, d (Drive) can be any number from Ø-3.
punctuation	Punctuation other than ellipses must be entered as shown.
delimiters	Commands must be separated from their operands by one or more blank spaces. Multiple operands, where allowed, may be separated from each other by a comma, a comma followed by one or more blanks, or by one or more blanks. Blanks and commas may not appear within an operand.

1/ Computer Graphics Overview

Graphics is the presentation of dimensional artwork. With TRS-8Ø Computer Graphics, the artwork is displayed on a two-dimensional plane -- your computer screen. Like an artist's easel or a teacher's blackboard, the screen is a "drawing board" for your displays.

TRS-8Ø Computer Graphics has two colors:

- Black (OFF)
- White (ON)

Graphics programming is different from other types of programming because your ultimate result is a pictorial display (bar graph, pie chart, etc.) rather than textual display (sum, equation, etc.). This is an important distinction. After working with graphics for a while, you'll find yourself thinking "visually" as you write programs.

In computer-generated graphics, displays can include tables, charts, graphs, illustrations and other types of artwork. Once they're created, you can "paint" displays with a variety of styles and shapes, or even simulate animation.

The Computer Graphics program uses a "high-resolution" screen. The more addressable points or dots (called "pixels") on a computer's screen, the higher the resolution. A lower resolution screen has fewer addressable pixels.

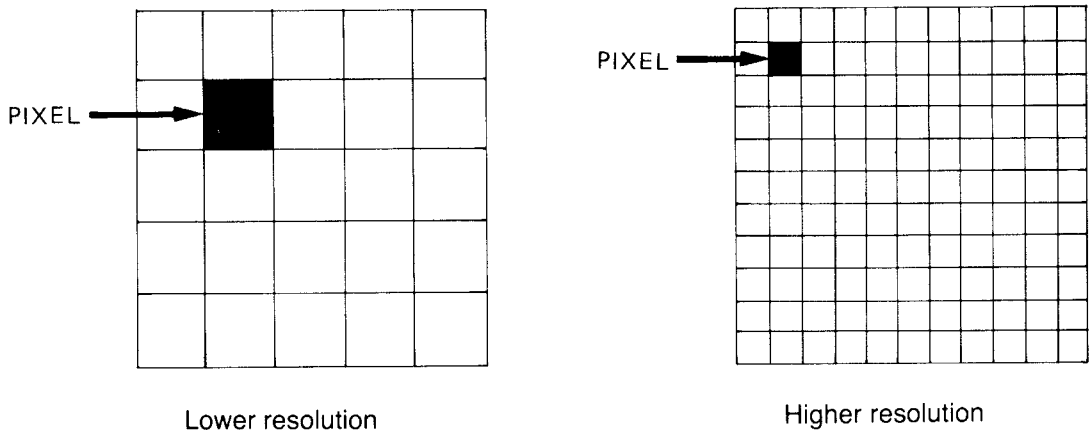


Figure 1. Resolution

Since the TRS-8Ø has high-resolution -- 64Ø pixels on the X-axis (Ø to 639) and 24Ø pixels on the Y-axis (Ø to 239) -- you can draw displays that have excellent clarity and detail.

How TRS-8Ø Computer Graphics Works

The concept of graphics is fairly simple. Each point on the screen can be turned ON (white) or OFF (black).

When you clear the Graphics Screen, all graphic points are turned OFF.

Therefore, by setting various combinations of the pixels (usually with a single command) either ON or OFF, you can generate lines, circles, geometric figures, pictures, etc.

The Graphics Subroutine Library, which is part of the Computer Graphics package, contains subroutines which provide the same capabilities, as well as similar names and parameters, as the commands and functions in Graphics BASIC. The main difference between the Subroutine Library and BASICG is the manner in which coordinates are specified (e.g., BASICG coordinates are specified as arguments for each command while the Subroutine Library specifies coordinates with a separate subroutine call). Another difference concerns the names of a few routines (e.g., LINE vs. LINEB vs. LINEBF, etc.). All of these differences will

be described in detail in the appropriate sections of this manual.

The Graphics Screen

TRS-8Ø Computer Graphics has two "screens" -- Text and Graphics. (We'll call them screens, although they are really modes.) Both screens can act independently of each other and make use of the computer's entire display area.

The Text Screen, also referred to as the "Video Display," is the "normal" screen where you type in your programs. The Graphics Screen is where graphic results are displayed. Both screens can be cleared independently. Note: The Graphics Screen will not automatically be cleared when you return to TRSDOS. It will be cleared when you re-enter BASICG.

The Graphics Screen cannot be displayed at the same time as the Text Screen.

While working with Computer Graphics, it might be helpful to imagine the screen as a large Cartesian coordinate plane (with a horizontal X- and a vertical Y-axis). However, unlike some coordinate systems, TRS-8Ø Computer Graphics' coordinate numbering starts in the upper-left corner -- (Ø,Ø) -- and increases toward the lower-right corner -- (639,239). The lower-left corner is (Ø,239) and the upper-right corner is (639,Ø).

Since the screen is divided into X-Y coordinates (like the Cartesian system), each pixel is defined as a unique position. In TRS-8Ø Computer Graphics, you can directly reference these coordinates as you draw.

About Ranges...

Some TRS-8Ø Computer Graphics commands accept values within the Model III integer range (-32768 to 32767), instead of just Ø to 639 for X and Ø to 239 for Y. Since most of the points in the integer range are off the screen, these points are part of what is called Graphics "imaginary" Cartesian system.

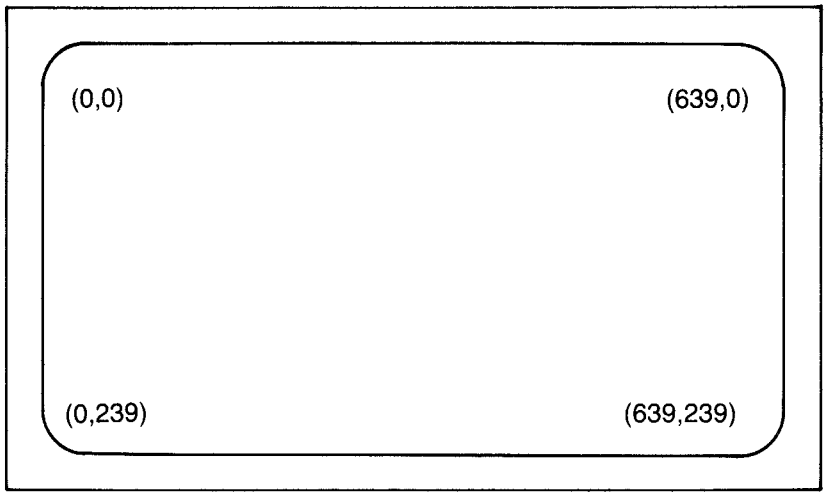


Figure 2. Graphics Visible Screen

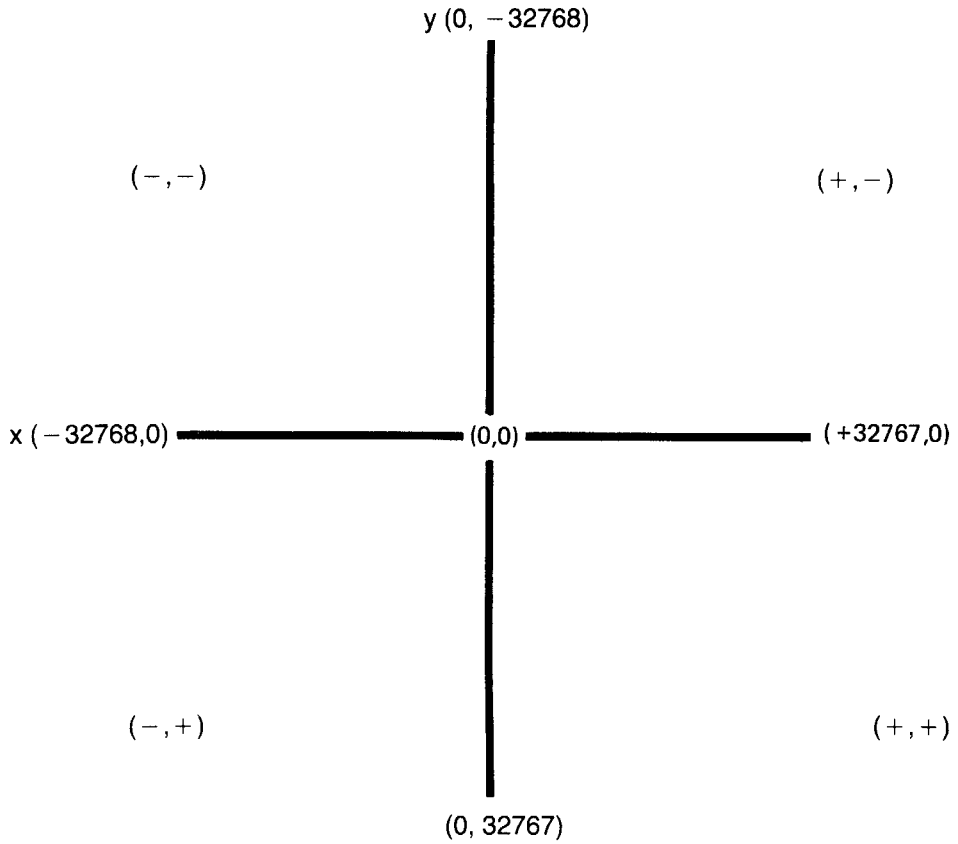


Figure 3. Graphics "Imaginary" Cartesian System

2/ Graphics BASIC

Graphics BASIC (BASICG) vs. BASIC

The Graphics BASIC file on the supplied diskette is named BASICG.

Program files created under BASICG are not directly loadable with BASIC files (and vice versa). If you attempt to load a BASIC file in compressed format from BASICG (and vice versa), an NB error may occur. See Appendix B for a list of BASICG error messages.

If you want to load a file from one BASIC to the other to the other, we recommend that you first save the file in ASCII format (**SAVE"filename/ext",A**).

You can then load and run a BASIC file from either BASICG or BASIC. You cannot run programs that contain BASICG statements while in BASIC.

Important Note: Because of memory limitations, some programs (i.e., some application programs) will not run in BASICG. BASICG uses approximately 6.5K more memory than BASIC. When you enter BASIC with Ø files, there are 39,282 bytes free. When you enter BASICG with Ø files, there are 32,675 bytes free. Some Graphics Commands use Free Memory. This means that the larger your BASIC programs are, the more limitations on your Graphics capabilities.

Each Graphics program statement has a specific syntax and incorporates a Graphics BASIC command or function.

Table 1 gives a brief description of the BASICG commands; Table 2 lists the BASICG functions. This section of the manual will describe each statement and function in detail.

BASICG Commands

Command	Description
CIRCLE	Draws a circle, arc, semicircle, etc.
CLR	Clears the Graphics Screen.
GLOCATE	Sets the Graphics Cursor and the direction for putting characters on the Graphics Screen.
GET	Reads the contents of a rectangle on the Graphics Screen into an array for future use by PUT.
LINE	Draws a line from the startpoint to the endpoint in the specified line style and color. Also creates a box.
PAINT	Paints an area, starting from a specified point. Also paints a specified style.
PRESET	Sets an individual dot (pixel) OFF (or ON).
PRINT #-3	Writes characters to the Graphics Screen.
PSET	Sets an individual dot (pixel) ON (or OFF).
PUT	Stores graphics from an array onto the Graphics Screen.
SCREEN	Selects the Graphics or Text Screen.
VIEW	Creates a viewport which becomes the current Graphics Screen.

Table 1

BASICG Functions

Function	Description
&POINT	Returns the OFF/ON color value of a pixel.
&VIEW	Returns the current viewport coordinates.

Table 2

Starting-Up

Before using the diskette included with this package, be sure to make a "safe copy" of it. See your **Model III Disk System Owner's Manual** for information on BACKUP.

To load BASICG:

1. Power up your System according to the start-up procedure in your **Model III Disk System Owner's Manual.**
2. Insert the backup diskette into Drive Ø.
3. Initialize the System as described in the "Operation" section of the **Model III Disk System Owner's Manual.**
4. When TRSDOS Ready appears, type:

   BASICG <ENTER>

The Graphics BASIC start-up prompts, followed by the READY prompt (>), appear and you are in Graphics BASIC. You can now begin BASICG programming.

Remember that Model III numeric values are as follows:

Model III Numeric Values

Numeric Type	Range	Storage Requirement	Example
Integer	-32768, 32767	2 bytes	24Ø, 639, -1Ø
Single-Precision	$-1*1Ø^{38}$,$-1*1Ø^{-38}$ $+1*1Ø^{38}$,$+1*1Ø^{-38}$ Up to 7 significant digits (Prints six)	4 bytes	22.5Ø,3.14259 -1ØØ.ØØ1
Double-Precision	$-1*1Ø^{38}$,$-1*1Ø^{-38}$ $+1*1Ø^{38}$,$+1*1Ø^{-38}$ Up to 17 significant digits (Prints 16)	8 bytes	123ØØØØ.ØØ 3.1415926535897932

Table 3

With each BASICG command or function, there are various options which you may or may not include in a program statement (depending on your needs). Each option is separated from the previous option by a delimiter, usually a comma. When you do not specify an available option (e.g., you use the default value) and you specify subsequent options, you must still enter the delimiter or a Syntax Error will result. (See your **Model III Operation and BASIC Language Reference Manual** for more information).

Because you are dealing with two distinct screens, the Graphics Screen and the Text Screen, we strongly urge you to read the description of the **SCREEN** command before continuing.

CIRCLE
Draws Circle, Semicircle, Ellipse, Arc, Point

CIRCLE (x,y),r,c,start,end,ar

(x,y) specifies the centerpoint of the figure. x and y are integer expressions.
r specifies the radius of the figure in pixels and is an integer expression.
c specifies the OFF/ON color of the figure and is a integer expression of either Ø (OFF/black) or 1 (ON/white). c is optional; if omitted, 1 is used.
start specifies the startpoint of the figure and is a numeric expression from Ø to 6.283185. start is optional; if omitted, Ø is used.
end specifies the endpoint of the figure and is a numeric expression from Ø to 6.283185. end is optional; if omitted, 6.283185 is used.
ar specifies the aspect ratio of the circle, is a single-precision floating-point number > Ø.Ø (to $1*1Ø^{38}$) and determines the major axis of the figure. ar is optional; if omitted, .5 is used and a circle is drawn.

The CIRCLE command lets you draw five types of figures:

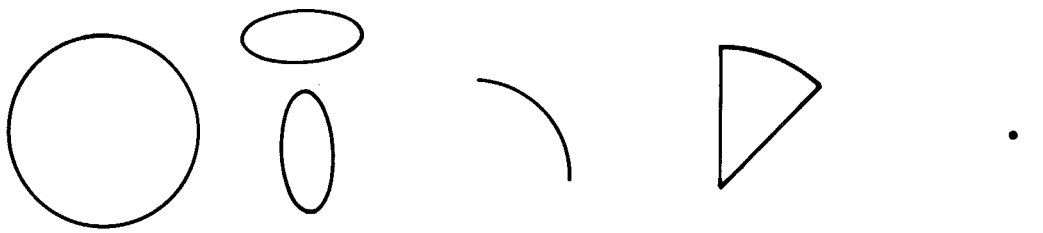


Figure 4. Types of Displays with CIRCLE

With CIRCLE, you can enter values for PI (and 2 x PI) up to 37 significant digits without getting an overflow error.

3.14159265358979323846264338327950288 41
6.28318530717958647692528676655900057682

However, you'll probably only be able to visually detect a change in the circle's start and end when PI is accurate to a few significant digits (e.g., 3.1, 6.28, etc.). The start and end values can't be more than 2 x PI (e.g., 6.2832 will not work) or an Illegal Function Call error will occur.

(x,y)
Centerpoint

The (x,y) coordinates in the CIRCLE statement specify the centerpoint of the figure. x and y are numeric expressions in the integer number range.

Example

```
CIRCLE (x,y),r

CIRCLE (32Ø,12Ø),r
```

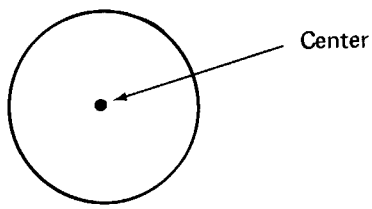


Figure 5. Center of Circle

r
Radius

The radius of a circle is measured in pixels and is a numeric expression in the integer range. Radius is the distance from the centerpoint to the edge of the figure. Although a negative value will be accepted by BASICG, the results of using a negative value are unpredictable.

The radius is either on the X-axis or Y-axis, depending on the aspect ratio (see ar). If the aspect ratio is greater than 1, the radius is measured on the Y-axis. If the aspect ratio is less than or equal to 1, the radius is measured on the X-axis.

Example

```
10 CIRCLE(320,120),100
```

This example draws a circle. The radius is 100 and the centerpoint is (320,120).

c
Color

You can set the ON/OFF (white/black) color of a figure's border and radius lines (see start/end) by specifying a numeric value of 1 or 0.

If you omit color, BASICG uses 1 (ON/white).

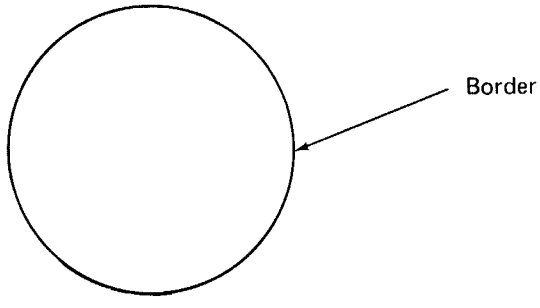


Figure 6. Border of Circle

start/end
Startpoint/Endpoint of Circle

The range for start and end is 0 to 6.283185 (2 x PI).

If you do not enter start and end, the default values of 0 and 6.28 respectively, are used.

A negative start or end value will cause the respective radius to be drawn in addition to the arc (i.e., it will draw a "piece of the pie"). The actual start and endpoints are determined by taking the absolute value of the specified start and endpoints. These values are measured in radians.

Note: Radius will not be drawn if start or end is -0. To draw a radius with start or end as 0, you must use -0.000...01.

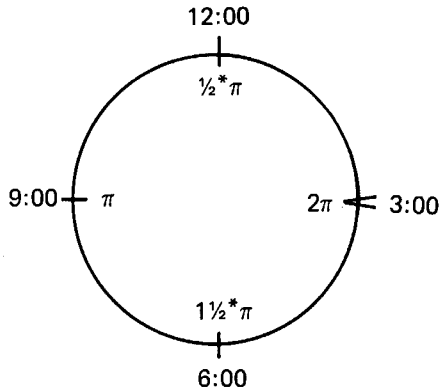


Figure 7. Clock/Radian Equivalents

Degrees	Radians	Clock Equivalent
0	0	3:00
90	1.57	12:00
180	3.14	9.00
270	4.71	6:00
360	6.28	3:00

Table 4. Degree/Radians/Clock Equivalents

You can draw semicircles and arcs by varying start and end. If start and end are the same, a point (one pixel) will be displayed instead of a circle.

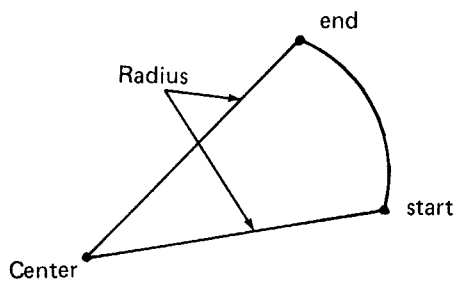


Figure 8. CIRCLE's (-) start, (-) end

You can have a positive start and a negative end (or vice versa) as well as negative starts and ends. In these cases, only one radius line is drawn.

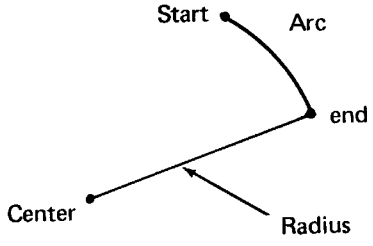


Figure 9. CIRCLE's (+) start, (-) end

Hints and Tips about start and end:

- When using the default values for start and end, you must use commas as delimiters if you wish to add more parameters.
- If you use PI, it is not a reserved word in BASICG and must be defined in your program.

ar
Aspect Ratio

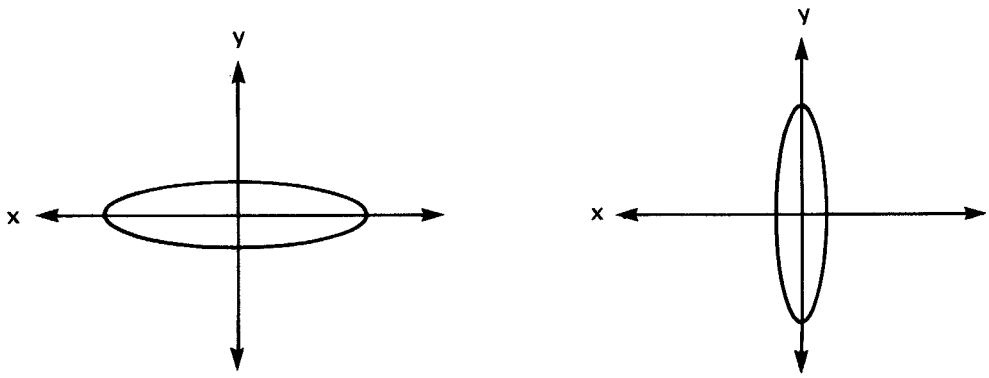
You can draw ellipses by varying the aspect ratio from the default value (.5) for a circle (and semicircle).

Every ellipse has a "major axis" which is the ellipse's longer, predominant axis. With an ellipse (as with a circle), the two axes are at right angles to each other.

The mathematical equation for determining the aspect ratio is:

ar = length of Y-axis/length of X-axis

- If the aspect ratio is .5, a circle is drawn.
- If the ratio is less than .5, an ellipse with a major axis on the X-axis is drawn.
- If the ratio is greater than .5, an ellipse with a major axis on the Y-axis is drawn.



X-Axis Ellipse (*ar* < .5) Y-Axis Ellipse (*ar* > .5)

Figure 1Ø. CIRCLE's Ellipse

The range for aspect ratio is a single-precision floating-point number greater than Ø.Ø (to $1*1Ø^{38}$). Although a negative value will be accepted by BASICG, the results of using a negative value are unpredictable.

Hints and Tips about aspect ratio:

- Entering .5 as the ratio produces a circle.
- Numbers between Ø and .5 produce an ellipse with a major axis on X.
- Numbers over .5 generate an ellipse with a major axis on Y.
- Even though you can enter large aspect ratios, large numbers may produce straight lines.

Examples

```
CIRCLE (32Ø,12Ø),9Ø,1
```

This example draws a white-bordered circle with the centerpoint of (32Ø,12Ø) and radius of 9Ø.

```
CIRCLE (32Ø,12Ø),9Ø,1,,,.7
```

This statement draws a white-bordered ellipse with an origin of (32Ø,12Ø) and radius of 9Ø. The major axis is the Y-axis.

```
CIRCLE (32Ø,12Ø),9Ø,1,-6.2,-5
```

This statement draws an arc with a vertex ("origin") of (32Ø,12Ø) and radius of 9Ø. start is 6.2 and end is 5. Radius lines are drawn for start and end.

```
CIRCLE (32Ø,12Ø),9Ø,1,,-4
```

This example draws an arc with a vertex of (32Ø,12Ø) and radius of 9Ø. start is Ø and end is 4. A radius line is drawn for end.

```
1Ø PI=3.1415926
2Ø CIRCLE (32Ø,12Ø),1ØØ,1,PI,2*PI,.5
```

A semicircle is drawn.

```
1Ø CIRCLE (15Ø,1ØØ),1ØØ,1,-5,-1
2Ø CIRCLE (22Ø,1ØØ),1ØØ,1,5,1
```

Two arcs are drawn with the same start and end point. The arc with the negative start and end has two radius lines drawn to the vertex. The arc with a positive start and end has no radius lines.

```
CIRCLE (32Ø,12Ø),14Ø,,-4,6.1
```

This statement draws an arc with a vertex at (32Ø,12Ø) and a radius of 14Ø. Start is 4 and end is 6.1. A radius line is drawn for start.

```
CIRCLE (32Ø,12Ø),14Ø,1,Ø,1,.5
```

This example draws an arc with a vertex of (32Ø,12Ø) and radius of 14Ø.

Sample Program

```
4 SCREEN Ø
5 CLR
1Ø FOR X=1Ø TO 2ØØ STEP 1Ø
2Ø CIRCLE (3ØØ,1ØØ),X,1,,,.9
3Ø NEXT X
4Ø FOR Y=1Ø TO 2ØØ STEP 1Ø
5Ø CIRCLE (3ØØ,1ØØ),Y,1,,,.1
6Ø NEXT Y
7Ø FOR Z=1Ø TO 2ØØ STEP 1Ø
8Ø CIRCLE (3ØØ,1ØØ),Z,1,,,.5
9Ø NEXT Z
1ØØ GOTO 5
```

A set of 2Ø concentric ellipses is drawn with a major axis on Y, a set of 2Ø concentric ellipses is drawn with a major axis on X, and a set of 2Ø concentric circles is drawn. The ellipses and circles in each of the three groups are concentric and the radius varies from 1Ø to 2ØØ.

CLR
Clears the Graphics Screen

CLR

CLR clears the Graphics Screen.

Example

```
1Ø SCREEN Ø
2Ø CIRCLE(32Ø,12Ø),1ØØ,1
```

This program line will draw a circle. Now type:

```
CLR <ENTER>
```

and the Graphics Screen will be cleared but the Text Screen will remain unchanged. This can be seen by typing:

```
SCREEN 1
```

GET
Reads the Contents of Rectangular Pixel Area into Array

GET(x1,y1)-(x2,y2),array name

(x1,y1) are coordinates of one of the opposing corners of a rectangular pixel area. x1 is an integer expression from Ø to 639. y1 is an integer expression from Ø to 239.
(x2,y2) are coordinates of the other corner of a rectangular pixel area. x2 is an integer expression from Ø to 639. y2 is an integer expression from Ø to 239.
array name is the name you assign to the array that will store the rectangular area's contents. array name must be specified.

Important Note: BASICG recognizes two syntaxes of the command GET -- the syntax described in this manual and the syntax described in the **Model III Operation and BASIC Language Reference Manual.** BASIC recognizes only the GET syntax described in the **Model III Operation and BASIC Language Reference Manual.**

GET reads the graphic contents of a rectangular pixel area into a storage array for future use by PUT (see PUT).

A rectangular pixel area is a group of pixels which are defined by the diagonal line coordinates in the GET statement.

The first two bytes of array name are set to the horizontal (X-axis) number of pixels in the pixel area; the second two bytes are set to the vertical (Y-axis) number of pixels in the pixel area. The remainder of array name represents the status of each pixel, either ON or OFF, in the pixel area. The data is stored in a row-by-row format. The data is stored 8 pixels per byte and each row starts on a byte boundary.

Array Limits

When the array is created, BASICG reserves space in memory for each element of the array. The size of the array is limited by the amount of memory available for use by your

program -- each real number in your storage array uses four memory locations (bytes).

The array must be large enough to hold your graphic display and the rectangular area must include all the points you want to store.

Your GET rectangular pixel area can include the entire screen (i.e., GET(Ø,Ø)-(639,239),array name), if the array is dimensioned large enough.

To determine the minimum array size:

1. Divide the number of X-axis pixels by 8 and round up to the next higher integer.

2. Multiply the result by the number of Y-axis pixels. When counting the X-Y axis pixels, be sure to include the first and last pixel.

3. Add four to the total.

4. Divide by four (for real numbers) or two (for integers) rounding up to the next higher integer.

The size of the rectangular pixel area is determined by the (x,y) coordinates used in GET:

```
Position:          upper-left corner = startpoint = (x1,y1)
                   lower-left corner = endpoint   = (x2,y2)

Size (in pixels):  width  = x2-x1+1
                   length = y2-y1+1
```

Example

```
GET(1Ø,1Ø)-(8Ø,5Ø),V
```

This block is 71 pixels wide on the X-axis (1Ø through 8Ø) and 41 long on the Y-axis (1Ø through 5Ø).

- For real: 71/8 = 9 * 41 = 369 + 4 = 373/4 = 94
- For integer: 71/8 = 9 * 41 = 369 + 4 = 373/2 = 187

Depending on the type of array you use, you could set up your minimum-size dimension statement this way:

- Real DIM V(93)

or

. Integer DIM V%(186)

Examples

```
1Ø DIM V(249)
2Ø CIRCLE (65,45),2Ø,1
3Ø GET (1Ø,1Ø)-(12Ø,8Ø),V
```

An array is created, a circle is drawn and stored in the array via the GET statement's rectangular pixel area's parameters (i.e., (1Ø,1Ø)-(12Ø,8Ø)).

Calculate the dimensions of the array this way:

Rectangular pixel area is 111 x 71. That equals:

```
111/8= 14 * 71 =994 + 4 = 998/4 = 25Ø
```

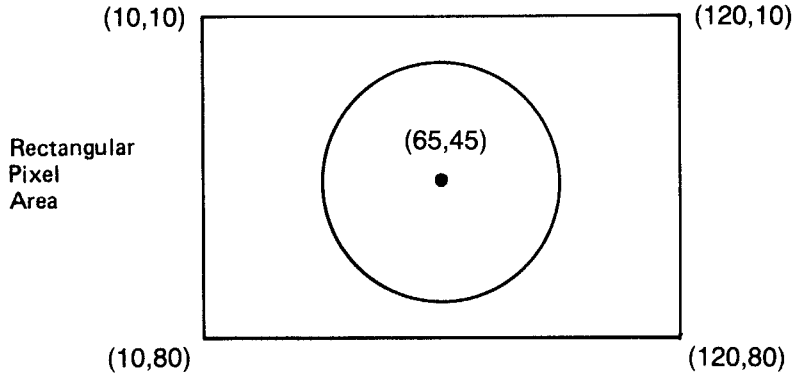


Figure 11

```
1Ø DIM V(3Ø,3Ø)
2Ø CIRCLE (5Ø,5Ø),1Ø
3Ø GET (1Ø,1Ø)-(8Ø,8Ø),V
```

A two-dimensional array is created, a circle is drawn and stored in the array via the GET statement's rectangular pixel area's parameters (i.e., (1Ø,1Ø)-(8Ø,8Ø)).

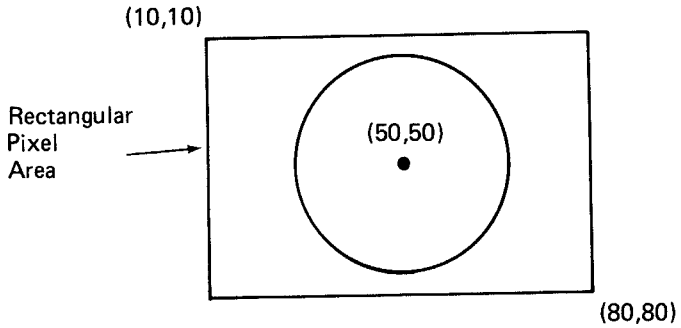


Figure 12

```
1Ø DIM V%(564)
2Ø CIRCLE (65,45),5Ø,1,1,3
3Ø GET(1Ø,1Ø)-(12Ø,8Ø),V%
```

A one-dimensional integer array is created, an arc is drawn and stored in the array via the GET statement's rectangular area's parameters.

GLOCATE
Sets the Graphics Cursor

GLOCATE (x,y), direction

(x,y) specifies the location of the Graphics Cursor and is a pair of integer expressions. direction specifies the direction that the characters will be written to the Graphics Screen and has an integer value of Ø, 1, 2, or 3. direction is optional; if omitted, Ø is used.

Since the Text Screen and the Graphics Screen cannot be displayed at the same time, you need an easy way to display textual data on the Graphics Screen. GLOCATE provides part of this function by allowing you to specify where on the Graphics Screen to start displaying the data, (x,y), and which direction to display it -- direction.

The allowable values for direction are:

Ø - zero degree angle
1 - 9Ø degree angle
2 - 18Ø degree angle
3 - 27Ø degree angle

Examples

```
1Ø GLOCATE (32Ø,12Ø),Ø
```

This program line will cause characters to be displayed starting in the center of the screen in normal left-to-right orientation.

```
1ØØ GLOCATE (32Ø,1Ø),1
```

This program line will cause characters to be displayed starting in the center of the top portion of the screen in a vertical orientation, going from the top of the screen to the bottom of the screen.

```
2ØØ GLOCATE (63Ø,12Ø),2
```

This program line will cause characters to be displayed upside down starting at the right of the screen and going towards the left.

```
3ØØ GLOCATE (32Ø,23Ø),3
```

This program line will cause the characters to be displayed vertically, starting at the center of the lower portion of the screen going towards the top of the screen.

LINE
Draws a Line or Box

LINE (x1,y1)-(x2,y2), c, B or BF, style

(x1,y1) specifies the starting coordinates of a line and is a pair of integer expressions. (x1,y1) is optional; if omitted, the last ending coordinates of any previous command are used as the startpoint. If a command has not been previously specified, (Ø,Ø) is used.
(x2,y2) specifies the ending coordinates of a line. (x2,y2) is a pair of integer expressions.
c specifies the color and is a numeric expression of either Ø or 1. c is optional; if omitted, 1 is used.
B or BF specifies drawing and/or shading (solid white or solid black) a box. B draws a box and BF fills a box with shading. B/BF is optional; if omitted, only a line is drawn.
style is the setting for the pattern of a line and is a numeric value in the integer range. style is optional; if omitted, -1 (solid line) is used. style must be omitted if BF is used.

LINE draws a line from the starting point (x1,y1) to the ending point (x2,y2).

If the starting point is omitted, either (Ø,Ø) is used if a previous end coordinate has not been specified or the last ending point of the previous command is used. If one or both parameters are off the screen, only the part of the line which is visible is displayed.

With over 65,5ØØ line styles possible, each style is slightly different. You'll find it's almost impossible to detect some of the differences since they are so minute.

LINE with Box Option

The start and end coordinates are the diagonal coordinates of the box (either a square or rectangle). When you don't specify the B or BF options, the "diagonal" line is drawn. When you specify the B option, the perimeter is drawn but not the diagonal line. When you specify the BF option, the perimeter is drawn, and the

area bounded by the perimeter is shaded in the specified color (c).

```
LINE(14Ø,8Ø)-(5ØØ,2ØØ),1,B
```

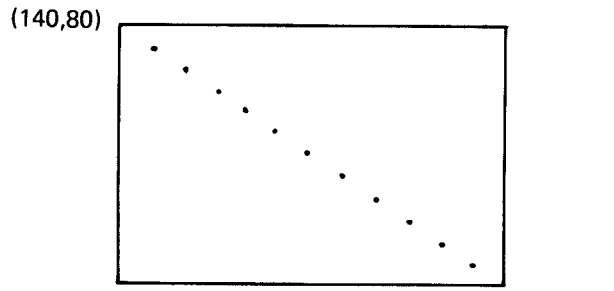


Figure 13

style

style sets the pixel arrangement in 16-bit groups.

For example, ØØØØ 1111 ØØØØ 1111 (binary), ØFØF (hex), or 3855 (decimal).

style can be any number in the integer range (negative or positive). Using hexadecimal numbers, you can figure the exact line style you want. There will always be four numbers in the hexadecimal constant.

To use hexadecimal numbers for style:

1. Decide what pixels you want OFF (bit=Ø) and ON (bit=1).
2. Choose the respective hexadecimal numbers (from the **Base Conversion Chart**, Appendix E).

Example

```
ØØØØ 1111 ØØØØ 1111 = &HØFØF
```

Creates a dashed line.

Type	Binary Numbers	Hex Numbers
Long dash	ØØØØ ØØØØ 1111 1111	&HØØFF
Short dash	ØØØØ 1111 ØØØØ 1111	&HØFØF
"Short-short" dash	11ØØ 11ØØ 11ØØ 11ØØ	&HCCCC
Solid line	1111 1111 1111 1111	&HFFFF
OFF/ON	Ø1Ø1 Ø1Ø1 Ø1Ø1 Ø1Ø1	&H5555
"Wide" dots	ØØØØ 1ØØØ ØØØØ 1ØØØ	&HØ8Ø8
Medium dots	1ØØØ 1ØØØ 1ØØØ 1ØØØ	&H8888
Dot-dash	1ØØØ 1111 1111 1ØØØ	&H8FF8

Table 5. Sample Line Styles

Examples

```
LINE -(1ØØ,4Ø)
```

This example draws a line in white (ON) starting at the last endpoint used and ending at (1ØØ,4Ø).

```
LINE (Ø,Ø)-(319,199)
```

This statement draws a white line starting at (Ø,Ø) and ending at (319,199).

```
LINE(1ØØ,1ØØ)-(2ØØ,2ØØ),1,,45
```

This example draws a line from (1ØØ,1ØØ) to (2ØØ,2ØØ) using line style 45 (&HØØ2D).

```
LINE (1ØØ,1ØØ)-(3ØØ,2ØØ),1,,&HØØFF
```

This LINE statement draws a line with "long dashes." Each dash is eight pixels long and there are eight blank pixels between each dash.

```
LINE (1ØØ,1ØØ)-(3ØØ,2ØØ),1,,-1ØØØ
```

This statement draws a line from (1ØØ,1ØØ) to (3ØØ,2ØØ) using line style -1ØØØ.

```
LINE (2ØØ,2ØØ)-(-1ØØ,1ØØ)
```

A line is drawn from the startpoint of (2ØØ,2ØØ) to (-1ØØ,1ØØ).

```
1Ø LINE (3Ø,3Ø)-(18Ø,12Ø)
2Ø LINE -(12Ø,18Ø)
3Ø LINE -(3Ø,3Ø)
```

This program draws a triangle.

```
1Ø LINE -(5Ø,5Ø)
2Ø LINE -(12Ø,8Ø)
3Ø LINE -(-1ØØ,-1ØØ)
4Ø LINE -(3ØØØ,1ØØØ)
```

This program draws four line segments using each endpoint as the startpoint for the next segment.

PAINT
Paints Screen

PAINT (x,y), tiling, border, background

(x,y) specifies the X-Y coordinates where painting is to begin. x is a numeric expression from Ø to 639 and y is a numeric expression from Ø to 239.
tiling specifies the paint style and can be a string or a numeric expression. tiling is optional; if omitted, 1 is used. tiling cannot be a null string ("") and no more than 64 bytes may be contained in the tiling string.
border specifies the OFF/ON color of the border where painting is to stop and is a numeric expression of either Ø (OFF) or 1 (ON). border is optional; if omitted, 1 is used.
background specifies the color of the background that is being painted and is a 1-byte string of either Ø (CHR$(&HØØ)) or 1 (CHR$(&HFF)). background is optional; if omitted, CHR$(&HØØ) is used.

PAINT shades the Graphics Screen with tiling starting at the specified X-Y coordinates, proceeding upward and downward.

x,y
Paint Startpoint

x,y is the coordinate where painting is to begin and must:

- Be inside the area to be painted.
- Be on the working area of the screen.

For example:

```
1Ø CIRCLE(32Ø,12Ø),8Ø
2Ø PAINT(32Ø,12Ø),1,1
```

A circle with a centerpoint of (32Ø,12Ø) is drawn and painted in white.

tiling
Paint Style

tiling is the pattern in a graphics display. By specifying each pixel, you can produce a multitude of tiling styles thereby simulating different shades of paint on the screen.

tiling is convenient to use in bar graphs, pie charts, etc., or whenever you want to shade with a defined pattern.

There are two types of tiling:

- Numeric expressions
- Strings

Numeric Expressions. There are only two numeric expressions that can be used for the paint style -- Ø and 1. 1 paints all pixels ON (solid white) and Ø paints all pixels OFF (solid black).

To use numeric expressions, enter either a Ø or 1. For example:

```
PAINT (32Ø,12Ø),1,1
```

Strings (Point-by-Point Painting). You can paint precise patterns using strings by defining a multi-pixel grid, pixel-by-pixel, on your screen as one contiguous pattern.

String painting is called "pixel" painting because you are literally painting the screen "pixel-by-pixel" in a predetermined order.

You can define the tile length as being one to 64 vertical tiles, depending on how long you want your pattern. Tile width, however, is always eight horizontal pixels (8 pixels representing one 8-bit byte). The dimensions of a tile pattern are length by width. Tile patterns are repeated as necessary to paint to the specified borders. Because of its symmetry, you'll probably find equilateral pixel grids most convenient.

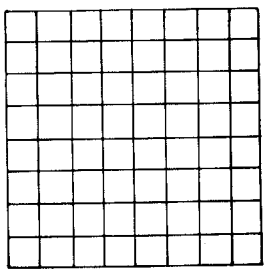


Figure 14. Example of an 8-by-8 Pixel Grid

Strings allow numerous graphic variations because of the many pixel combinations you can define.

Important Note: You cannot use more than two consecutive rows of tiles which match the background or an Illegal Function Call error will occur. For example:

```
PAINT (1,1),CHR$(&HFF)+CHR$(&HFF)+CHR$(&HØØ)+CHR$(&HØØ)
+CHR$(&HØØ)+CHR$(&HØØ),1,CHR$(&HØØ)
```

returns an Illegal Function Call error.

Using Tiling

You may want to use a sheet of graph paper to draw a style pattern. This way, you'll be able to visualize the pattern and calculate the binary and hexadecimal numbers needed.

Note: Tiling should only be done on either a totally black or white background; otherwise, results are unpredictable.

To draw an example of a tile on paper:

1. Take a sheet of paper and draw a grid according to the size you want (8 x 8, 24 x 8, etc.). Each boxed area on this grid, hypothetically, represents one pixel on your screen.

2. Decide what type of pattern you want (zigzag, diagonal lines, perpendicular lines, etc.).

3. Fill in each grid in each 8-pixel-wide row of the tile if you want that pixel to be ON, according to your pattern. If you want the pixel to be OFF, leave the

grid representing the pixel blank.

4. On your paper grid, count each ON pixel as 1 and each OFF pixel as Ø. List the binary numbers for each row to the side of the grid. For example, you might have ØØØ1 1ØØØ on the first row, Ø111 ØØ11 on the second row, etc.

5. Using a hexadecimal conversion chart, convert the binary numbers to hexadecimal numbers. (Each row equates to a two-digit hexadecimal number.)

6. Insert the hexadecimal numbers in a tile string and enter the string in your program.

Note: For a listing of commonly used tiling styles, see Appendix F.

Example

For example, if you're working on an 8 x 8 grid and want to draw a plus ("+") sign:

8 x 8 grid								Binary	Hex
Ø	Ø	Ø	1	1	Ø	Ø	Ø	ØØØ1 1ØØØ	18
Ø	Ø	Ø	1	1	Ø	Ø	Ø	ØØØ1 1ØØØ	18
Ø	Ø	Ø	1	1	Ø	Ø	Ø	ØØØ1 1ØØØ	18
1	1	1	1	1	1	1	1	1111 1111	FF
1	1	1	1	1	1	1	1	1111 1111	FF
Ø	Ø	Ø	1	1	Ø	Ø	Ø	ØØØ1 1ØØØ	18
Ø	Ø	Ø	1	1	Ø	Ø	Ø	ØØØ1 1ØØØ	18
Ø	Ø	Ø	1	1	Ø	Ø	Ø	ØØØ1 1ØØØ	18

Figure 15

Tile string:
```
A$=CHR$(&H18)+CHR$(&H18)+CHR$(&H18)+CHR$(&HFF)+CHR$(&HFF)
   +CHR$(&H18)+CHR$(&H18)+CHR$(&H18)
```

b
Border

Border is the OFF/ON color of the border of a graphics design where painting is to stop and is a numeric expression of either Ø or 1. If omitted, 1 (ON) is used and all the pixels on the border are set (solid white).

background
Background Area

Background is a 1-byte character which describes the background of the area you are painting. CHR$(&HØØ) specifies a black background and CHR$(&HFF) is a totally white background. If background is not specified, BASICG uses CHR$(&HØØ).

Painting continues until a border is reached or until PAINT does not alter the state of any pixels in a row. However, if

pixels in a given row are not altered and the tile that was to be painted in that row matches the background tile, painting will continue on to the next row.

Note: BASICG uses Free Memory for tiling.

Examples

```
10 CIRCLE (300,100),100
20 PAINT (300,100),1,1
```

Paints the circle in solid white.

```
10 CIRCLE (100,100),300
20 PAINT (100,100),1,1
```

Paints the circle. Only the visible portion of the circle is painted on the screen.

```
5 A=1
6 SCREEN 0
10 CIRCLE (320,120),100
20 CIRCLE (100,100),50
30 CIRCLE (400,200),60
40 CIRCLE (500,70),50
50 PAINT (320,120),A,1
60 PAINT (100,100),A,1
70 PAINT (400,200),A,1
80 PAINT (500,70),A,1
```

The tiling style is assigned the value 1 in line 5 (A=1) for all PAINT statements. Four circles are drawn and painted in solid white.

```
10 LINE (140,80)-(500,200),1,B
20 PAINT (260,120),CHR$(&HEE)+CHR$(&H77)+CHR$(00),1
```

Paints box in specified tiling style using strings.

```
10 CIRCLE (300,100),100
20 PAINT (300,100),"D",1
```

This example uses a character constant to paint the circle in vertical black and white stripes. The character "D" (0100

Ø1ØØ) sets this vertical pattern: one vertical row of pixels ON, three rows OFF.

```
1Ø CIRCLE (32Ø,12Ø),2ØØ
2Ø PAINT (32Ø,12Ø),"332211",1
3Ø PAINT (1ØØ,7Ø),"EFEF",1
```

This example draws and paints a circle, then paints the area surrounding the circle with a different paint style (line 3Ø). This PAINT statement's (line 3Ø) startpoint must be outside the border of the circle.

```
1Ø PAINT (32Ø,12Ø),CHR$(&HFF),1
2Ø CIRCLE (32Ø,12Ø),1ØØ,Ø
3Ø PAINT (32Ø,12Ø),CHR$(Ø)+CHR$(&HFF),Ø,CHR$(&HFF)
```

Paints the screen white, draws a circle and paints the circle with a pattern.

```
1Ø PAINT (32Ø,12Ø),CHR$(&HFF),1
2Ø CIRCLE (32Ø,12Ø),1ØØ,Ø
3Ø PAINT (32Ø,12Ø),CHR$(Ø)+CHR$(&HAA),Ø,CHR$(&HFF)
```

Paints the screen white, draws a circle and paints the circle with a pattern.

```
1Ø CIRCLE(3ØØ,1ØØ),1ØØ
2Ø A$=CHR$(&HØØ)+CHR$(&H7E)+CHR$(&H18)+CHR$(&H18)+CHR$(&H18)
   +CHR$(&H18)+CHR$(&H18)+CHR$(&HØØ)
3Ø PAINT(3ØØ,1ØØ),A$,1
```

This draws the circle and paints with the letter T within the parameters of the circle.

```
1Ø A$=CHR$(&H41)+CHR$(&H22)+CHR$(&H14)+CHR$(&HØ8)+CHR$(&H14)
  +CHR$(&H22)+CHR$(&H41)+CHR$(&HØØ)
2Ø PAINT (3ØØ,1ØØ),A$, 1
```

This paints Xs over the entire screen.

```
1 CLEAR 1ØØ
5 SCREEN Ø
1Ø TILE$(Ø)=CHR$(&H22)+CHR$(&HØØ)
2Ø TILE$(1)=CHR$(&HFF)+CHR$(&HØØ)
3Ø TILE$(2)=CHR$(&H99)+CHR$(&H66)
4Ø TILE$(3)=CHR$(&H99)
5Ø TILE$(4)=CHR$(&HFF)
6Ø TILE$(5)=CHR$(&HFØ)+CHR$(&HFØ)+CHR$(&HØF)+CHR$(&HØF)
7Ø TILE$(6)=CHR$(&H3C)+CHR$(&H3C)+CHR$(&HFF)
8Ø TILE$(7)=CHR$(&HØ3)+CHR$(&HØC)+CHR$(&H3Ø)+CHR$(&HCØ)
9Ø A$=TILE$(Ø)+TILE$(1)+TILE$(2)+TILE$(3)+TILE$(4)
   +TILE$(5)+TILE$(6)+TILE$(7)
1ØØ PAINT(3ØØ,1ØØ),A$,1
```

This example paints the screen with a tiling pattern made up of eight individually defined tile strings (Ø-7).

&POINT (function)
Returns Pixel Value

```
&POINT(x,y)

  x specifies an X-coordinate and is an integer
    expression.
  y specifies an Y-coordinate and is an integer
    expression.
  values returned by &POINT are:
    Ø (pixel OFF)
    1 (pixel ON)
    -1 (pixel is off the screen)
```

The &POINT command lets you read the OFF/ON value of a pixel from the screen.

Values for &POINT that are off the screen (i.e., PRINT &POINT (8ØØ,5ØØ)) return a -1, signifying the pixel is off the screen.

Example

```
1Ø PSET(3ØØ,1ØØ),1
2Ø PRINT &POINT(3ØØ,1ØØ)
```

Reads and prints the value of the pixel at the point's coordinates (3ØØ,1ØØ) and displays its value: 1.

```
PRINT &POINT(3ØØØ,1ØØØ)
```

Since the pixel is off the screen, a -1 is returned.

```
PRINT &POINT(-3ØØØ,-1ØØØ)
```

Since the pixel is off the screen, a -1 is returned.

```
PSET(2ØØ,1ØØ),Ø
PRINT &POINT(2ØØ,1ØØ)
```

Reads and prints the value of the pixel at the point's coordinates (2ØØ,1ØØ) and displays its value: Ø.

```
1Ø PSET(3ØØ,1ØØ),1
2Ø IF &POINT(3ØØ,1ØØ)=1 THEN PRINT "GRAPHICS BASIC!"
```

Sets the point ON. Since the point's value is 1, line 2Ø is executed and Graphics BASIC is displayed:

```
GRAPHICS BASIC!
```

```
5 SCREEN Ø
1Ø PSET(RND(64Ø),RND(24Ø)),1
2Ø IF &POINT(32Ø,12Ø)=1 THEN STOP
3Ø GOTO 1Ø
```

Sets points randomly until (32Ø,12Ø) is set.

```
5 CLR
1Ø LINE(5Ø,8Ø)-(12Ø,1ØØ),1,BF
2Ø PRINT &POINT(1ØØ,8Ø)
3Ø PRINT &POINT(11Ø,8Ø)
4Ø PRINT &POINT(115,9Ø)
5Ø PRINT &POINT(5Ø,4Ø)
6Ø PRINT &POINT(13Ø,12Ø)
```

The first three pixels are in the filled box, so the value 1 (one) is displayed for each of the statements in lines 2Ø, 3Ø, and 4Ø. The pixels specified in lines 5Ø and 6Ø are not in the shaded box and Øs are returned.

PRESET
Sets Pixel OFF (or ON)

PRESET(x,y),switch

x specifies an X-coordinate and is an integer expression.
y specifies an Y-coordinate and is an integer expression.
switch specifies a pixel's OFF/ON code and is an integer of either Ø (OFF) or 1 (ON).
switch is optional; if omitted, Ø (OFF) is used.

PRESET sets a pixel either OFF (Ø) or ON (1), depending on switch. If switch is not specified, Ø (OFF) is used.

Values for (x,y) that are larger than the parameters of the screen (i.e., greater than 639 for x and 239 for y) are accepted, but these points are off the screen and therefore are not PRESET.

Note: The only choice for switch is Ø or 1. If you enter any other number, an Illegal Function Call error will result.

Examples

```
1Ø PRESET (5Ø,5Ø),1
2Ø PRESET (5Ø,5Ø),Ø
```

Turns ON the pixel located at the specified coordinates (in line 1Ø) and turns the pixel OFF (in line 2Ø).

```
1Ø PRESET (32Ø,12Ø),1
2Ø PRESET (3ØØ,1ØØ),1
3Ø PRESET (34Ø,14Ø),1
4Ø FOR I=1 TO 1ØØØ: NEXT I
5Ø PRESET (32Ø,12Ø)
6Ø PRESET (3ØØ,1ØØ)
7Ø PRESET (34Ø,14Ø)
8Ø FOR I=1 TO 1ØØØ: NEXT I
```

Sets the three specified pixels ON (through the three PRESET statements), pauses, and then turns the three pixels OFF.

```
PRESET(3ØØØ,1ØØØ),1
```

The values for (x,y) are accepted, but since the coordinates are beyond the parameters of the screen, the point is not PRESET.

PRINT #-3,
Write Text Characters to the Graphics Screen

PRINT #-3, item list

item list may be either string constants (messages enclosed in quotes), string variables, numeric constants (numbers), variables, or expressions involving all of the preceding items. The items to be printed may be separated by commas or semicolons. If commas are used, the Cursor automatically advances to the next print zone before printing the next item. If semicolons are used, a space is not inserted between the items printed on the screen. In cases where no ambiguity would result, all punctuation can be omitted.

PRINT #-3, is used to write text characters to the Graphics Screen. This is the easiest way to display textual data on the Graphics Screen. Characters are displayed starting at the current Graphics Cursor and going in the direction specified by the most recently executed GLOCATE command. If a GLOCATE command was not executed prior to the PRINT #-3, command, a direction of Ø is assumed.

PRINT #-3, will only print text characters (see Appendix C of the **Model III Operation and BASIC Language Reference Manual**). Each character displayed in the Ø or 2 direction uses an 8 X 8 pixel grid; each character displayed in the 1 or 3 direction uses a 16 X 8 grid. Executing this command will position the Graphics Cursor to the end of the last character that was displayed.

Displaying text in direction Ø engages a wraparound feature. If the end of a line is reached, BASICG will continue the

display on the next line. If the end of the screen is reached, BASICG will continue the display at the beginning of the screen without scrolling. If there is not enough room to display at least one character at the current Graphics Cursor, an Illegal Function Call error will result. When displaying text in other directions, an attempt to display text outside of the currently defined screen will cause an Illegal Function Call error to be given.

PSET
Sets Pixel ON (or OFF)

PSET(x,y),switch

x specifies an X-coordinate and is an integer expression.
y specifies an Y-coordinate and is an integer expression.
switch specifies a pixel's OFF/ON color code and is a numeric expression of Ø (OFF) or 1 (ON). switch is optional; if omitted, 1 (ON) is used.

PSET sets a pixel either OFF (Ø) or ON (1), depending on switch. If switch is not specified, 1 (ON) is used.

The only choice for switch with PSET is Ø and 1. If you enter any other number, an Illegal Function Call will occur.

Values for (x,y) that are larger than the parameters of the screen (i.e., greater than 639 for x and 239 for y) are accepted, but these points are off the screen and therefore are not PSET.

Note: The only distinction between PRESET and PSET in BASICG is the default value for switch. The default value for PRESET is Ø, while the value for PSET is 1.

Examples

```
1Ø A=1
2Ø PSET (5Ø,5Ø),A
```

Turns the pixel located at the specified coordinates ON.

```
10 PSET (RND(640),RND(240)),1
20 GOTO 10
```

Pixels are randomly set to 1 (ON) over the defined area (the entire screen).

```
PSET (-300,-200),1
```

The values for (x,y) are accepted, but since it is beyond the parameters of the screen, the pixel is not set.

```
10 PSET (320,120),1
20 A$=INKEY$: IF A$= "" THEN 20
30 PSET(320,120),0
```

Line 10 sets ("turns ON") a pixel; line 30 resets ("turns OFF") the same dot.

PUT
Puts Rectangular Pixel Area from Array onto Screen

PUT(xl,yl),array name,action

(xl,yl) are coordinates of the upper-left corner of the rectangular pixel area which is to contain a graphic display. xl is a numeric expression from Ø to 639 and yl is a numeric expression from Ø to 239.

array name is the name of an array (previously specified by GET) that contains the data to be written into the rectangular pixel area.

action determines how the data is written into the rectangular pixel area and is one of the following:

PSET	Sets or resets each point in the specified pixel area to the value in the specified array.
PRESET	Sets or resets each point in the specified pixel area to the inverse of the value in the specified array.
XOR	Performs a logical exclusive-OR between the bits in the specified array and the pixels in the destination area and displays the result.
OR	Performs a logical OR between the bits in the specified array and the pixels in the destination area and displays the result.
AND	Performs a logical AND between the bits in the specified array and the pixels in the destination area and displays the result.

action is optional; if omitted, XOR is used.

Important Note: BASICG recognizes two syntaxes of the command PUT -- the syntax described in this manual and the syntax described in the **Model III Operation and BASIC Language Reference Manual.** BASIC recognizes only the PUT syntax described in the **Model III Operation and BASIC Language Reference Manual.**

The PUT function puts a rectangular pixel area stored in an array, and defined by GET, onto the screen. GET and PUT work

jointly. Together, they allow you to "get" a rectangular pixel area which contains a graphic display, store it in an array, then "put" the array back on the screen later.

Remember that before you GET or PUT, you have to create an array to store the bit contents of the display rectangular pixel area. The size of the array must match that of the display rectangular pixel area.

PUT moves your GET rectangular pixel area to the startpoint in your PUT statement and the startpoint is the new upper-left corner of the rectangular pixel area.

To illustrate:

```
5 DIM V(3)
1Ø GET (2,3)-(7,7),V
1ØØ PUT (5Ø,5Ø),V,PSET
```

After GETting, PUT this rectangular pixel area to (5Ø,5Ø). The new coordinates are:

```
(5Ø,5Ø) (51,5Ø) (52,5Ø) (53,5Ø) (54,5Ø) (55,5Ø)
(5Ø,51) (51,51) (52,51) (53,51) (54,51) (55,51)
(5Ø,52) (51,52) (52,52) (53,52) (54,52) (55,52)
(5Ø,53) (51,53) (52,53) (53,53) (54,53) (55,53)
(5Ø,54) (51,54) (52,54) (53,54) (54,54) (55,54)
```

The rectangular pixel area ((5Ø,5Ø)-(55,54)) is exactly the same pixel size as (2,3)-(7,7); only the location is different.

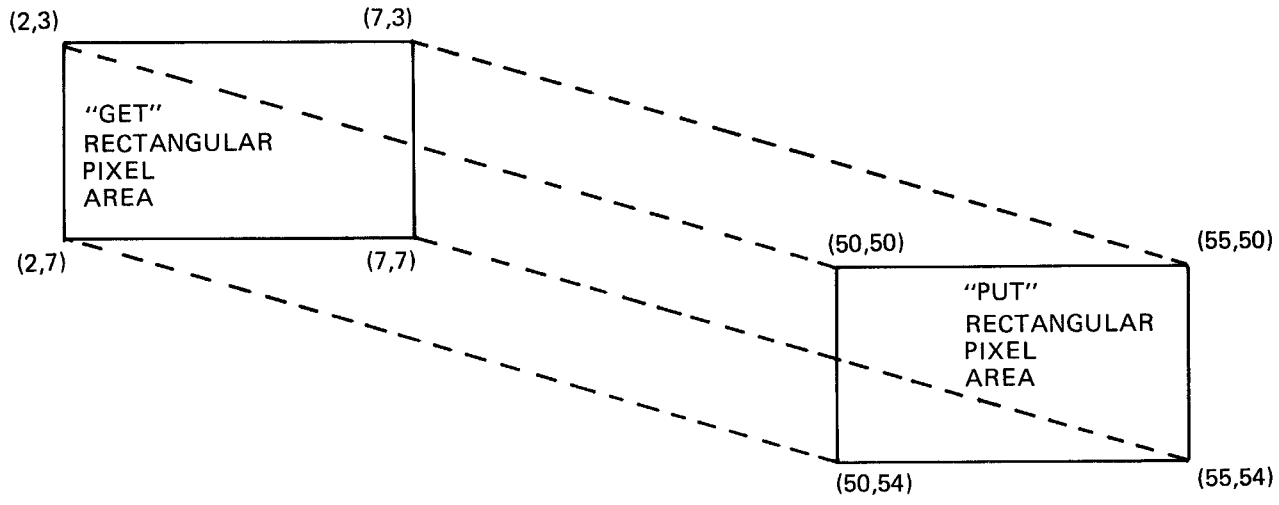


Figure 16

With PUT, action can be PSET, PRESET, OR, AND, or XOR.

These operators are used in BASICG to test the OFF/ON (or Ø/1) conditions of a pixel in the original pixel area and the destination pixel area.

For example (using PSET), the pixel is set ON only if the bit in the PUT array is set ON. If the bit is OFF, the pixel is turned OFF (reset).

With PRESET, the pixel is set ON only if the bit in the PUT array is set OFF. If the bit is ON, the pixel is turned OFF (reset).

Using OR, the pixel is set ON if the bit in the PUT array is ON or the corresponding pixel in the destination area is ON. In all other cases, the pixel is turned OFF (reset). In other words:

OR	OFF	ON
OFF	OFF	ON
ON	ON	ON

With AND, the pixel is set ON if both the bit in the PUT array and the corresponding pixel in the destination area are ON. In all other cases, the pixel is turned OFF (reset). In other words:

AND	OFF	ON
OFF	OFF	OFF
ON	OFF	ON

Using XOR, the pixel is set ON if either the bit in the PUT array or the corresponding pixel in the destination area (but not both) is ON. In all other cases, the pixel is turned OFF (reset). In other words:

XOR	OFF	ON
OFF	OFF	ON
ON	ON	OFF

The following BASICG program will graphically illustrate the differences between the various action options. Since the program will give you a "hard-copy" printout of the action options, you'll need to connect your TRS-8Ø to a graphic printer. See "Graphics Utilities" later in this manual for more details on using the Computer Graphics package with a printer.

```
1Ø DATA "OR", "AND", "PRESET", "PSET", "XOR"
2Ø CLR : SCREEN Ø
3Ø FOR Y= 1Ø TO 21Ø STEP  5Ø
4Ø FOR X=  Ø TO 4ØØ STEP 2ØØ
5Ø LINE (X+4Ø,Y-5)-(X+1ØØ,Y+25),1,B
6Ø NEXT X
7Ø LINE (5Ø,Y)-(9Ø,Y+1Ø),1,BF
8Ø FOR X= 2ØØ TO 4ØØ STEP 2ØØ
9Ø LINE (X+5Ø,Y)-(X+7Ø,Y+2Ø),1,BF
1ØØ NEXT X
11Ø NEXT Y
12Ø DIM V(1ØØ)
13Ø GET (5Ø,1Ø)-(9Ø,3Ø),V
14Ø FOR N= 1 TO 5
15Ø R= (N-1)*5+1
16Ø READ A$
165 GLOCATE (136,R*1Ø),Ø
17Ø PRINT #-3, A$;
175 GLOCATE (36Ø,R*1Ø),Ø
18Ø PRINT #-3, "= ";
19Ø ON N GOTO 2ØØ, 21Ø, 22Ø, 23Ø, 24Ø
2ØØ PUT (45Ø,1Ø),  V,OR:     GOTO 25Ø
21Ø PUT (45Ø,6Ø),  V,AND:    GOTO 25Ø
22Ø PUT (45Ø,11Ø), V,PRESET: GOTO 25Ø
23Ø PUT (45Ø,16Ø), V,PSET:   GOTO 25Ø
24Ø PUT (45Ø,21Ø), V,XOR
25Ø NEXT N
26Ø CMD "I", "GPRINT"
27Ø SCREEN1
```

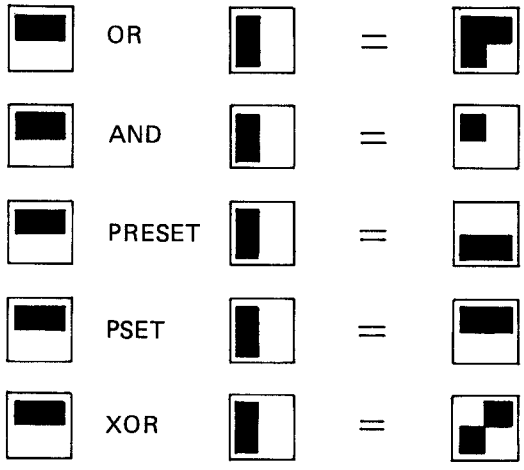


Figure 17

Hints and Tips about PUT:

- An Illegal Function Call error will result if you attempt to PUT a rectangular pixel area to a section of the screen which is totally or partially beyond the parameters of the screen. For example:

```
GET(5Ø,5Ø)-(15Ø,15Ø),V
PUT(2ØØ,2ØØ),V,PSET
```

  returns an error because the rectangular pixel area cannot be physically moved to the specified rectangular pixel area (i.e., (2ØØ,2ØØ)-(3ØØ,3ØØ)).

- If you use PUT with a viewport (see VIEW), all coordinates must be within the parameters of the viewport or you'll get an Illegal Function Call error.

Examples

PUT with PSET

```
1Ø DIM V%(63)
15 SCREEN Ø
2Ø CIRCLE (3Ø,3Ø),1Ø
3Ø GET (1Ø,1Ø)-(4Ø,4Ø),V%
4Ø FOR I=1 TO 5ØØ: NEXT I
5Ø CLR
6Ø PUT (11Ø,11Ø),V%,PSET
7Ø FOR I=1 TO 5ØØ: NEXT I
```

In this example, the circle is drawn, stored, moved and re-created. First the white-bordered circle appears in the upper left corner of the screen (position (3Ø,3Ø) -- program line 2Ø). After a couple of seconds (because of the delay loop), it disappears and then reappears on the screen -- (11Ø,11Ø) -- program line 6Ø.

What specifically happened is:

1. An array was created (line 1Ø).

2. A circle was drawn (line 2Ø).

3. GET -- The circle which was within the source rectangular pixel area, as specified in the GET

statement's parameters is stored in the array (line 3Ø).

4. The screen is cleared (line 5Ø).

5. PUT -- The circle from the array was PUT into the destination rectangular pixel area as specified in the PUT statement (line 6Ø) with the PSET option.

```
1Ø DIM V%(7ØØ)
2Ø LINE (2Ø,2Ø)-(2Ø,8Ø)
3Ø LINE (8Ø,Ø)-(8Ø,8Ø)
4Ø LINE (3Ø,3Ø)-(3Ø,8Ø)
5Ø LINE (1Ø,5)-(1Ø,8Ø)
6Ø GET (Ø,Ø)-(1ØØ,1ØØ),V%
7Ø FOR I=1 TO 1ØØØ: NEXT I
8Ø PUT (18Ø,12Ø),V%,PSET
9Ø FOR I=1 TO 1ØØØ: NEXT I
```

Draws four lines. GET stores the lines in the rectangular pixel area. PUT moves the lines to another rectangular pixel area.

SCREEN
Selects Screen

SCREEN type

type specifies which "Screen" to use and is a numeric expression of either Ø or 1.
Ø = Graphics Screen
1 = Text Screen

SCREEN lets you set the proper screen. SCREEN Ø selects the Graphics Screen; SCREEN 1 selects the Text Screen. Any value other than Ø or 1 with SCREEN gives an error.

SCREEN is convenient to use when you want to display either a Graphics Screen or a Text Screen. For example, you may have run a program and then added to it. With SCREEN, you can remove the graphics display, add to the program, and then return to the Graphics Screen.

Whenever BASICG tries to display a character on the Text Screen (like in an INPUT or PRINT statement), the screen is

automatically set to the Text Screen. If the program is still running after executing the statement, BASICG will revert to the screen that was in effect prior to executing the statement.

Examples

```
1Ø SCREEN 1
2Ø LINE (15Ø,15Ø)-(2ØØ,2ØØ)
```

The computer executes the short program but the Graphics Screen cannot display the graphics because of the SCREEN 1 command. To display the line, type: SCREEN Ø <ENTER>.

```
1Ø CLR
2Ø SCREEN 1
3Ø LINE(1Ø,1Ø)-(255,191)
4Ø LINE(Ø,191)-(255,Ø)
5Ø A$=INKEY$: IF A$=""THEN 5Ø
6Ø SCREEN Ø
7Ø A$=INKEY$: IF A$=""THEN 7Ø
8Ø GOTO 1Ø
```

The computer executes the program (draws two intersecting lines) but the screen cannot display the graphics because of SCREEN 1. By pressing any key, the graphics are displayed because of SCREEN Ø.

```
1Ø CIRCLE (2ØØ,1ØØ),1ØØ
2Ø PAINT (2ØØ,1ØØ),"44",1
```

Now run the program and type:

```
SCREEN Ø <ENTER>
```

This command turns the Graphics Screen ON.
By entering the SCREEN 1 and SCREEN Ø commands, you can alternately turn the Graphics Screen OFF and ON without losing the executed program display.

VIEW (command)
Redefines the Screen (Creates a Viewport)

VIEW (x1,y1)-(x2,y2), c, b

(x1,y1) are coordinates of the upper-left corner of a rectangular viewport area. x1 is an integer expression between Ø and 639. y1 is an integer expression between Ø and 239.
(x2,y2) are coordinates of the lower-right corner of a rectangular viewport area. x2 is an integer expression >= to x1 and <= 639. y2 is an integer expression >= y1 and <=239.
c specifies the color of the interior of the viewport and is an integer expression of either Ø or 1. c is optional; if omitted, the viewport is not shaded.
b specifies the border color of the viewport and is an numeric expression of either Ø or 1. b is optional; if omitted, a border is not drawn.

VIEW creates a "viewport" which redefines the screen parameters (Ø-639 for X and Ø-239 for Y). This defined area then becomes the only place you can draw graphics displays.

If you enter more than one viewport, you can only draw displays in the last defined viewport.

Since VIEW redefines the SCREEN:

- CLR clears the interior of the viewport only.
- If you PSET or PRESET points, draw circles, etc., beyond the parameters of the currently defined viewport, only the portions that are in the viewport will be displayed.
- If you try to read a point beyond the viewport (with POINT), it will return a -1.
- You can only GET and PUT arrays within the viewport.
- You can't PAINT outside the viewport.

The upper-left corner of the viewport is read as (Ø,Ø) (the "relative origin") when creating items inside the viewport. All the other coordinates are read relative to this origin. However, the "absolute coordinates" of the viewport, as they are actually defined on the Graphics Cartesian system, are retained in memory and can be read using VIEW as a function.

Every viewport has absolute and relative coordinates and graphic displays are drawn inside using the relative coordinates. For example:

```
1Ø VIEW (1ØØ,1ØØ)-(2ØØ,2ØØ),Ø,1
2Ø LINE (3Ø,15)-(8Ø,6Ø),1
```

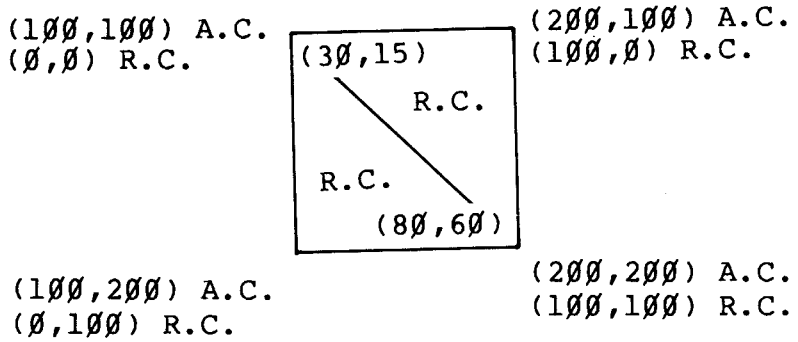


Figure 18

Note: After each of the following examples, you'll have to redefine the entire screen to VIEW(Ø,Ø)-(639,239) before performing any other Graphics functions.

Examples

```
VIEW (1ØØ,1ØØ)-(2ØØ,2ØØ),Ø,1
```

Draws a black viewport (pixels OFF) that is outlined in white (border pixels ON).

```
VIEW (1ØØ,1ØØ)-(2ØØ,2ØØ),1,1
```

Draws a white viewport (pixels ON) that is outlined in white (border pixels ON).

```
VIEW (5Ø,5Ø)-(1ØØ,1ØØ),1,Ø
```

Draws a white viewport (pixels ON) that is outlined in black (border pixels OFF).

```
10 VIEW (10,10)-(600,200),0,1
20 VIEW (50,50)-(100,100),0,1
30 LINE(RND(500),RND(190))-(RND(500),RND(190))
40 GOTO 30
```

First you defined a large viewport that almost covered the entire screen. Next you defined a smaller viewport. The Random command draws lines within the specified parameters but only the segments of the lines that are within the parameters of the smaller viewport are visible since it was specified last.

```
10 VIEW(80,80)-(400,200),0,1
20 VIEW(100,90)-(300,170),0,1
30 VIEW(120,100)-(200,200),0,1
40 VIEW(50,50)-(100,100),0,1
```

Draws four viewports. All further drawing takes place in the last viewport specified.

```
10 VIEW(210,80)-(420,160),0,1
20 CIRCLE(300,120),180,1
30 LINE(15,15)-(60,60),1
40 CIRCLE(90,40),50,1
50 LINE(40,30)-(500,30),1
```

Draws a viewport. Draws a circle but only a portion is within the parameters of the viewport. This circle's centerpoint is relative to the upper left corner of the viewport and not to the absolute coordinates of the graphics Cartesian system. A line is drawn which is totally within the parameters of the viewport. Another circle is drawn which is totally within the parameters of the viewport. Another line is drawn which is only partially within the parameters of the viewport.

```
10 VIEW (190,70)-(440,180),0,1
20 CIRCLE (300,140),170,1
30 CIRCLE (100,230),400,1
40 LINE (10,10)-(500,230),1
```

Draws a viewport. A circle is drawn but only a portion is within the parameters of the viewport. Another circle is drawn and a larger portion is within the parameters of the

viewport. A line is drawn but only a segment is within the parameters of the viewport.

&VIEW (function)
Returns Viewport Coordinates

&VIEW(p)

(p) specifies a coordinate on the X- or Y-axis and is a integer expression between Ø-3. Ø returns the left X-coordinate of your viewport. 1 returns the upper Y-coordinate. 2 returns the right X-coordinate. 3 returns the lower Y-coordinate.

&VIEW returns a corner coordinate of a viewport. It is important to note the parentheses are not optional. If you enter the &VIEW function without the parentheses, a Syntax Error will result.

To display one of the four viewport coordinates, you must enter one of the following values for p:

- Ø returns the upper left X-coordinate
- 1 returns the upper left Y-coordinate
- 2 returns the lower right X-coordinate
- 3 returns the lower right Y-coordinate

Important Note: When you have defined several viewports, &VIEW only returns the coordinates of the last-defined viewport.

Examples

Set up the following viewport:

```
VIEW(1ØØ,8Ø)-(22Ø,15Ø),Ø,1
```

Now type: PRINT &VIEW(Ø) <ENTER>

Displays: 1ØØ

Type: PRINT &VIEW(1) <ENTER>

Displays: 8Ø

Enter: PRINT &VIEW(2) <ENTER>

Displays: 22Ø

Type: PRINT &VIEW(3) <ENTER>

Displays: 15Ø

Set up the following viewports:

```
VIEW(1ØØ,8Ø)-(22Ø,15Ø),Ø,1 <ENTER>
VIEW(25Ø,17Ø)-(35Ø,22Ø),Ø,1 <ENTER>
```

Now enter: PRINT &VIEW(Ø) <ENTER>

Displays: 25Ø

Type: PRINT &VIEW(1) <ENTER>

Displays: 17Ø

Now type: PRINT &VIEW(2) <ENTER>

Displays: 35Ø

Type: PRINT &VIEW(3) <ENTER>

Displays: 22Ø

3/ Graphics Utilities

There are six utilities included with the TRS-8Ø Computer Graphics package which are intended to be used as stand-alone programs. However, if you are an experienced programmer, you can use these with BASICG and FORTRAN. The source-code for each utility, that illustrate Graphics programming techniques, is listed later in this section.

The Graphics Utilities let you:

- Save graphic displays to diskette.
- Load graphic displays from diskette.
- Print graphic displays on a graphics printer.
- Turn graphics display OFF or ON.
- Clear graphics memory.

To use these utilities from BASICG, use the CMD"I" command followed by a comma and the name of the utility in quotation marks (e.g., CMD"I","GCLS" <ENTER>) and control returns to TRSDOS Ready. From TRSDOS, enter the utility directly, without quotation marks (e.g., GCLS <ENTER>).

To call these routines from FORTRAN, see the Subprogram Linkage section of your **TRS-8Ø Model III FORTRAN Manual** (26-22ØØ).

Note: These utilities load into high memory starting at FØØØ (hex); therefore, they cannot be used with DEBUG, DO, or any communication drivers that use high memory.

Utilities	
Command	**Action**
GCLS	Clears graphics screen.
GLOAD	Loads graphics memory from diskette.
GPRINT	Lists graphics on the printer.
GPRT2	Prints graphic display on the printer without 9Ø degree rotation.
GPRT3	Prints graphic display on the printer without 9Ø degree rotation.
GROFF	Turns Graphic Screen OFF.
GRON	Turns Graphic Screen ON.
GSAVE	Saves graphics memory to diskette.

Table 6

GCLS
Clears Graphics Screen

GCLS

GCLS clears the Graphics Screen by erasing the contents of graphics memory corresponding to the visible Graphics Screen. GCLS erases graphics memory by writing zeroes (OFF) to every bit in memory. GCLS does not clear the Text Screen (video memory).

Examples

When TRSDOS Ready is displayed, type:

```
GCLS <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
CMD"I","GCLS" <ENTER>
```

or

```
1ØØ CMD"I","GCLS"
```

GLOAD
Loads Graphics Memory from Diskette

GLOAD filename /ext .password :d

filename consists of a name of up to eight characters; the first character must be a letter.
/ext is an optional name-extension; ext is a sequence of up to three numbers or letters.
.password is an optional password; password is a name of up to eight characters; the first character must be a letter.
:d is an optional drive specification; d is one of the digits Ø through 3.

Note: There cannot be spaces within a file specification. TRSDOS terminates the file specification at the first space.

With GLOAD, you can load TRSDOS files that have graphic contents into graphics memory. These files must have been previously saved to diskette using GSAVE.

Examples

When TRSDOS Ready is displayed, type:

```
GLOAD PROGRAM/DAT.PASSWORD:Ø <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
CMD"I","GLOAD PROGRAM" <ENTER>
```

or

```
1ØØ CMD"I", "GLOAD PROGRAM"
```

GPRINT
Lists Graphic Display to Printer

GPRINT

GPRINT lets you print graphics memory on a graphics (dot-addressable) printer, such as Radio Shack's DMP-1ØØ (26-1253) or DMP-2ØØ (26-1254). Both of these printers have a 9 1/2" carriage. However, distortion will occur when Graphic routines are printed with GPRINT. This is because GPRINT is not a true pixel-by-pixel "Screen Dump" since the pixel size and spacing on the screen is different from the pixel size and spacing on the Printer. GPRINT is a point of departure for you to obtain hard-copy representations of graphics.

To print graphic displays, GPRINT turns the contents of the Graphic Screen clockwise 9Ø degrees and then prints.

However, FORMS must be used to set printing parameters.

Most uses will require that you set FORMS when TRSDOS Ready is displayed:

```
FORMS (LINES=6Ø,WIDTH=Ø) <ENTER>
```

See your **Model III Operation and BASIC Language Reference** and printer owner's manual for more details on setting printing parameters.

Important Note! Do not press <BREAK> while GPRINT is executing.

Examples

When TRSDOS Ready is displayed, type:

```
GPRINT <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
CMD"I","GPRINT" <ENTER>
```

or

```
1ØØ CMD"I","GPRINT"
```

For a complete GPRINT sample session, see Appendix D.

GPRT2
Print Graphics

GPRT2

GPRT2 is similar to GPRINT but is designed for use with wide-carriage (15") printers such as the DMP-4ØØ and DMP-5ØØ.

GPRT2 is different from GPRINT in that the image is not rotated 9Ø degrees and a different aspect ratio is used.

If GPRT2 does not produce the quality of print out you desire, try GPRT3 or GPRINT.

Important Note! Do not press <BREAK> while GPRT2 is executing.

Examples

When TRSDOS Ready is displayed, type:

```
GPRT2 <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
CMD"I","GPRT2" <ENTER>
```

or

```
1ØØ CMD"I","GPRT2"
```

GPRT3
Print Graphics (Double on the Y-Axis)

GPRT3

GPRT3 is similar to GPRINT but is designed for use with wide-carriage (15") printers such as the DMP-4ØØ and DMP-5ØØ.

GPRT3 is different from GPRINT in that the image is not rotated 9Ø degrees and a different aspect ratio is used.

If GPRT3 does not produce the quality of print-out you desire, try GPRT2 or GPRINT.

Important Note! Do not press <BREAK> while GPRT3 is executing.

Examples

When TRSDOS Ready is displayed, type:

```
GPRT3 <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
CMD"I","GPRT3" <ENTER>
```

or

```
1ØØ CMD"I","GPRT3"
```

GROFF
Turns Graphics Display OFF

GROFF

GROFF turns the Graphics Screen OFF. GROFF is different from GCLS since GROFF simply removes the Graphics display without erasing the contents of graphic memory. GCLS completely clears graphics memory by writing zeroes (OFF) to every bit in memory.

Examples

When TRSDOS Ready is displayed, type:

```
GROFF <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
CMD"I","GROFF" <ENTER>
```

or

```
1ØØ CMD"I","GROFF"
```

GRON
Turns Graphics Display ON

```
GRON
```

GRON turns the Graphics Screen ON.

Examples

When TRSDOS Ready is displayed, type:

```
GRON <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
CMD"I","GRON" <ENTER>
```

or

```
1ØØ CMD"I","GRON"
```

GSAVE
Saves Graphics Memory to Diskette

GSAVE filename /ext .password :d

filename consists of a name of up to eight characters; the first character must be a letter.
/ext is an optional name-extension; ext is a sequence of up to three numbers or letters.
.password is an optional password; password is a name of up to eight characters; the first character must be a letter.
:d is an optional drive specification; d is one of the digits Ø through 3.

Note: There cannot be spaces within a file specification. TRSDOS terminates the file specification at the first space.

With GSAVE, the contents in graphics memory is saved under a specified filename which follows the standard TRSDOS format. To load the file back into memory, use GLOAD.

Examples

When TRSDOS Ready is displayed, type:

```
GSAVE PROGRAM/DAT.PASSWORD:Ø <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
CMD"I","GSAVE PROGRAM" <ENTER>
```

or

```
1ØØ CMD"I","GSAVE PROGRAM"
```

4/ Graphics Subroutine Library (FORTRAN)

The Graphics Subroutine Library included on the Computer Graphics diskette lets you use the functions of TRS-8Ø Computer Graphics while programming in Model III FORTRAN (26-22ØØ). This library (GRPLIB/REL) must be linked to any FORTRAN program that accesses the Graphics Subroutines.

BASICG vs. the Graphics Subroutine Library

The Graphics Subroutine Library contains subroutines which provide the same capabilities as the Graphics commands and functions in BASICG. The Graphics subroutines have basically the same names and parameters as the BASICG commands. The major differences between the Library subroutines and the BASICG commands are:

- The BASICG command LINE has three corresponding library subroutines: LINE, LINEB, and LINEBF. LINEB and LINEBF provide the functions of the BASICG command LINE with the parameters B and BF respectively.
- The BASICG command PAINT has two corresponding library subroutines: PAINT and PAINTT. PAINT is for painting solid black or white, and PAINTT is for painting with tiling.
- The Library subroutines that correspond to BASICG commands that use (x,y) coordinates (except for VIEW) use (x,y) coordinates that have been previously set. The subroutines used to set the coordinates are SETXY and SETXYR.

Setting Points Using SETXY and SETXYR

The coordinates specified by SETXY or SETXYR will be called the "current" and "previous" coordinates. Subroutines that use one (x,y) coordinate pair use the "current" coordinates and subroutines that use two (x,y) pairs use both the "current" and the "previous" coordinates. Each call to SETXY or SETXYR sets the coordinates as follows:

1. Assign the values of the "current" (x,y) coordinates to the "previous" (x,y) coordinates, (discarding the old "previous" coordinates).

2. Assign new values for the "current" (x,y) coordinates as specified by the arguments supplied. SETXY simply sets the "current" coordinates to the values of its arguments. SETXYR adds the values of its arguments to the "current" coordinates to obtain the new coordinates.

Initialization

Before any calls are made to Graphics, the Graphics library and board must be initialized. A special initialization routine (GRPINI) is included in the library. A call to GRPINI must be made as the first access to the Graphics library.

Example

```
ØØ1ØØ  C     SAMPLE INITIALIZATION
ØØ15Ø        DIMENSION V(3Ø,3Ø)
ØØ2ØØ        CALL GRPINI(Ø)
```

Linking

The Library (GRPLIB/REL) must be linked to any programs that access the Graphics Subroutines. You must use the linker (L8Ø) to generate the load module.

Example

```
L8Ø <ENTER>
*SAMPLE:1-N
*GRPHSAM,GRPLIB-S,FORLIB-S,-U
*-E
```

This example links both the Graphics Library and the FORTRAN Subroutine Library to the relocatable file GRPHSAM/REL. In this example, SAMPLE:1-N is the file name, drive specification, and switch, respectively; GRPHSAM, GRPLIB-S, FORLIB-S, and -U are the names of the relocatable modules to be linked and their respective switches. -E ends the routine and creates the executable program SAMPLE. The *'s in the example are prompts for the user -- not data to be entered.

Note: If there are unresolved external references, the FORTRAN Library may need to be scanned a second time.

Errors

If you enter incorrect parameters for any of the Graphics Subroutines, your screen will display:

GRAPHICS ERROR

and return program control to TRSDOS Ready. This is the only error message you'll get when executing the Subroutines.

Important Note: Free memory is utilized by the Graphic Routine for temporary storage. Extreme care should be exercised if your program accesses this memory.

Routines/Functions

Most of the FORTRAN Subroutines and functions described in this section have a corresponding command in the Graphics BASIC Language Reference section of this manual.

FORTRAN Routines

Routine	Action
CIRCLE	Draws a circle, arc, semicircle, or ellipse.
CLS	Clears the Graphics Screen.
GET	Reads the contents of a rectangular pixel area into an array.
GPRINT	Displays textual data on the Graphics Screen.
GRPINI	Graphics initialization routine.
LINE	Draws a line.
LINEB	Draws a box.
LINEBF	Draws a filled box.
LOCATE	Sets the direction for displaying textual data on the Graphics Screen.
PAINT	Paints the screen in specified OFF/ON color.
PAINTT	Paints the screen in a specified pattern.
PRESET	Sets pixel OFF/ON.
PSET	Sets pixel OFF/ON.
PUT	Puts the stored array on the screen.
SCREEN	Selects the screen.
SETXY	Sets (x,y) coordinates (absolute).
SETXYR	Sets (x,y) coordinates (relative).
VIEW	Sets up a viewport where graphics is displayed.

Table 7

FORTRAN Functions

Function	Action
POINT	Reads a pixel's value at a specified coordinate.
FVIEW	Reads a viewport's parameters.

Table 8

CIRCLE
Draws a Circle, Arc, Semicircle, Point or Ellipse

CIRCLE (radius,color,start,end,ar)

radius is of INTEGER type and specifies the radius of the circle.
color is of LOGICAL type, specifies the OFF/ON color of the border of the circle and is a integer expression of either Ø or 1.
start is of REAL type and specifies the startpoint of the circle.
end is of REAL type and specifies the endpoint of the circle.
ar is the aspect ratio, is of REAL type and determines the major axis of the circle. If ar is Ø, Ø.5 is used.

CIRCLE draws a circle. By varying start, end, and aspect ratio, you can draw arcs, semicircles, or ellipses using current X- and Y-coordinates as the centerpoint (set by SETXY or SETXYR).

If start and end are Ø.Ø, a circle is drawn starting from the center right side of the circle. Note: In the CIRCLE statement, end is read as 2 x PI even though you have entered Ø.Ø. If you enter Ø.Ø for aspect ratio, a symmetric circle is drawn.

Example

```
CALL CIRCLE(1ØØ,1,Ø.Ø,Ø.Ø,Ø.Ø)
```

Sample Program

This example draws and paints a circle.

```
ØØØ1Ø  C    SAMPLE PROGRAM FOR CIRCLE
ØØØ2Ø       LOGICAL COLOR,OPTION
ØØØ3Ø       COLOR=1
ØØØ4Ø       OPTION=Ø
ØØØ5Ø       CALL GRPINI(OPTION)
ØØØ6Ø       CALL CLS
ØØØ7Ø       CALL SETXY(3ØØ,1ØØ)
ØØØ8Ø       CALL CIRCLE(1ØØ,COLOR,Ø.Ø,Ø.Ø,Ø.Ø)
ØØØ9Ø       CALL PAINT(COLOR,COLOR)
ØØ1ØØ       END
```

CLS
Clears Graphics Screen

```
CLS
```

Example

```
CALL CLS
```

Sample Program (see CIRCLE)

GET
Reads Contents of a Rectangular Pixel Area into an Array

GET (array,size)

array is any type and is the name of the array you specify.
size is of INTEGER type and specifies the size of the array in terms of bytes.

GET reads the contents of a rectangular pixel area into an array for future use by PUT. The pixel area is a group of pixels which are defined by the current x and y, and the previous X- and Y-coordinates specified by the SETXY call.

The first two bytes of array are set to the horizontal (X-axis) number of pixels in the pixel area; the second two bytes are set to the vertical (Y-axis) number of pixels in the pixel area. The remainder of array represents the status of each pixel (either ON or OFF) in the pixel area. The data is stored in a row-by-row format. The data is stored eight pixels per byte and each row starts on a byte boundary.

Array Limits

When the array is defined, space is reserved in memory for each element of the array. The size of the array is limited by the amount of memory available for use by your program -- each real number in your storage array uses four memory locations (bytes).

The array must be large enough to hold your graphic display and the rectangular area defined must include all the points you want to store.

To determine the minimum array size:

1. Divide the number of X-axis pixels by 8 and round up to the next higher integer.

2. Multiply the result by the number of Y-axis pixels.

   When counting the X-Y axis pixels, be sure to include the first and last pixel.

3. Add four to the total.

4. Divide by four (for real numbers) and two (for integers) rounding up to the next higher integer. (Note: If you're using a LOGICAL array, the result of Step #3 above will produce the desired array size.)

When using arrays, the position and size of the rectangular pixel area is determined by the current and previous (x,y) coordinates.

```
Position:          upper left corner = startpoint = (x1,y1)
                   lower left corner = endpoint = (x2,y2)

Size (in pixels):  width =  x2-x1+1
                   length = y2-y1+1
```

Example

```
CALL GET(A,4ØØØ)
```

Sample Program

This example draws a circle, saves the circle into an array, then restores the array to the graphics video.

```
ØØØ5Ø  C    SAMPLE FOR GET AND PUT
ØØ1ØØ       LOGICAL V(128),ACTION
ØØ15Ø                ACTION=1
ØØ2ØØ       CALL GRPINI(Ø)
ØØ3ØØ       CALL CLS
ØØ35Ø  C    DRAW A CIRCLE
ØØ4ØØ       CALL SETXY(3Ø,3Ø)
ØØ5ØØ       CALL CIRCLE(1Ø,1,Ø.Ø,Ø.Ø,Ø.Ø)
ØØ55Ø  C    SET COORDINATES FOR GET ARRAY
ØØ6ØØ       CALL SETXY(1Ø,1Ø)
ØØ7ØØ       CALL SETXY(4Ø,4Ø)
ØØ75Ø  C    STORE GRAPHICS INTO ARRAY WITH GET
ØØ8ØØ       CALL GET(V,128)
ØØ9ØØ       DO 1Ø I=1,5ØØØ
Ø1ØØØ  1Ø   CONTINUE
Ø1Ø5Ø  C    CLEAR SCREEN AND RESTORE GRPH FROM ARRAY
Ø11ØØ       CALL CLS
Ø12ØØ       CALL SETXY(11Ø,11Ø)
Ø13ØØ       CALL PUT(V,ACTION)
Ø14ØØ       DO 2Ø I=1,5ØØØ
Ø15ØØ  2Ø   CONTINUE
Ø16ØØ       END
```

GPRINT
Write Text Characters to the Graphics Screen

GPRINT (cnt,array)

cnt is of INTEGER type and specifies the number of characters to display.
array is a one dimensional LOGICAL array containing the characters to be displayed.

GPRINT is used to write text characters to the Graphics Screen. This is the easiest way to display textual data on the Graphics Screen. Characters are displayed starting at the current (x,y) coordinates and going in the direction specified by the most recently executed LOCATE call. If no LOCATE call was executed prior to the GPRINT call, a direction of Ø is assumed.

GPRINT will only print text characters (see Appendix C of the **Model III Operation and BASIC Language Reference Manual**). Each character displayed in the Ø or 2 direction uses an 8 X 8 pixel grid; each character displayed in the 1 or 3 direction uses a 16 X 8 grid. Executing this command will set the current (x,y) coordinates to the end of the last character that was displayed.

Displaying text in direction Ø engages a wraparound feature. If the end of a line is reached, the display will be continued on the next line. If the end of the screen is reached, the display will be continued at the beginning of the screen without scrolling. If there is not enough room to display at least one character at the current (x,y) coordinates, a GRAPHICS ERROR will result. When displaying text in other directions, an attempt to display text outside the currently defined screen will cause a GRAPHICS ERROR to be given.

GRPINI
Graphics Initialization Routine

GRPINI(option)

option is of LOGICAL type; Ø clears the Graphics Screen, non-zero does not clear the Graphics Screen.

GRPINI is the graphics initialization routine. This function must be called before any other graphics calls are made in FORTRAN.

Example

```
CALL GRPINI(1)
```

Sample Program (see CIRCLE)

LINE
Draws Line

LINE (color, style)

color is of LOGICAL type, specifies the OFF/ON color of a line and is an integer expression of either Ø (OFF, black) or 1 (ON, white).
style is of INTEGER type, specifies the pattern of the line and is a number in the integer range. -1 indicates a solid line.

LINE draws a line between the previous and current coordinates. These coordinates are set by the SETXY or SETXYR subroutines.

Example

```
CALL LINE (1,-1)
```

Sample Program

This example draws a diagonal line connected to a box, which is connected to a filled box.

```
ØØØ1Ø  C    SAMPLE FOR LINE LINEB LINEBF
ØØØ2Ø       LOGICAL COLOR
ØØØ3Ø       COLOR=1
ØØØ4Ø       CALL GRPINI(Ø)
ØØØ5Ø       CALL CLS
ØØØ6Ø       CALL SETXY(1,1)
ØØØ7Ø       CALL SETXY(21Ø,8Ø)
ØØØ8Ø       CALL LINE(COLOR,-1)
ØØØ9Ø       CALL SETXY(42Ø,16Ø)
ØØ1ØØ  C    COORDINATES ARE NOW (21Ø,8Ø) (42Ø,16Ø)
ØØ11Ø       CALL LINEB(COLOR,-1)
ØØ12Ø       CALL SETXY(639,239)
ØØ13Ø  C    COORDINATES ARE NOW (42Ø,16Ø) (639,239)
ØØ14Ø       CALL LINEBF(COLOR)
ØØ15Ø       END
```

LINEB
Draws Box

LINEB (color, style)

color is of LOGICAL type, specifies the OFF/ON color of a line and is a integer expression of either Ø (OFF, black) or 1 (ON, white).
style is of INTEGER type and specifies the pattern of the line. -1 indicates a solid line.

LINEB is the same as LINE except LINEB draws a box between the two sets of coordinates set by the SETXY or SETXYR subroutines.

Example

```
CALL LINEB (1,-1)
```

Sample Program (see LINE)

LINEBF
Draws Painted Box

LINEBF (color)

color is of LOGICAL type, specifies the OFF/ON color of a line and is an integer expression of either Ø (OFF, black) or 1 (ON, white).

LINEBF is the same as LINEB except LINEBF fills the box (colors in the box) and the argument style is not used.

Example

```
CALL LINEBF (1)
```

Sample Program (see LINE)

LOCATE
Sets the Direction for Displaying Text on the Graphics Screen

LOCATE (direction)

direction is of LOGICAL type, specifies the direction that GLOCATE will use to display textual data and is an integer expression of Ø-3.

LOCATE sets the direction that GPRINT will use to display textual data. The allowable values for direction are:

Ø - zero degree angle
1 - 9Ø degree angle
2 - 18Ø degree angle
3 - 27Ø degree angle

Examples

```
CALL LOCATE (Ø)
```

This program line will cause characters to be displayed at the current (x,y) coordinates in normal left to right orientation.

```
CALL LOCATE (1)
```

This program line will cause characters to be displayed at the current (x,y) coordinates in a vertical orientation going from the top of the screen to the bottom of the screen.

```
CALL LOCATE (2)
```

This program line will cause characters to be displayed upside down starting at the right of the screen and going towards the left.

```
CALL LOCATE (3)
```

This program line will cause the characters to be displayed vertically starting at the lower portion of the screen going towards the top of the screen.

PAINT
Paints Screen in Specified Color

PAINT (color, border)

color is of LOGICAL type, specifies the OFF/ON color of painting and is an integer expression of either Ø (OFF, black) or 1 (ON, white).
border is of LOGICAL type, specifies the OFF/ON color of the border and is an integer expression of either Ø (OFF, black) or 1 (ON, white).

PAINT paints the screen in the specified OFF/ON color (black or white). It uses the current X- and Y-coordinates (see SETXY) as its startpoint.

Example

```
CALL PAINT(1,1)
```

Sample Program (see CIRCLE)

PAINTT
Paints Screen in Specified Pattern

PAINTT (arrayT, border, arrayS)

arrayT is a byte array which defines a multi-pixel pattern to be used when painting (tiling). The first byte of arrayT indicates the length of the "tile" (number of bytes).
border is of LOGICAL type and specifies the color of the border. border is an integer expression of either Ø (black) or 1 (white).
arrayS is a byte array that is used to define the background. The first byte is always set to 1; the second byte describes the background you are painting on (X'FF' = white, X'ØØ' = black).

PAINTT lets you paint a precisely defined pattern using a graphics technique called "tiling." You can paint with tiling by defining a multi-pixel grid in an array and then using that array as the paint pattern.

Example

```
CALL PAINTT (A,1,V)
```

Sample Program

```
ØØ1ØØ  C     EXAMPLE FOR PAINT WITH TILE
ØØ15Ø        LOGICAL A,B,BORDER
ØØ2ØØ        DIMENSION A(9)
ØØ3ØØ        DIMENSION B(2)
ØØ35Ø  C     DEFINE TILE ARRAY HERE
ØØ4ØØ        DATA A(1), A(2), A(3) / 8, X'81', X'42'/
ØØ5ØØ        DATA A(4),A(5),A(6)/X'24',X'18',X'18'/
ØØ6ØØ        DATA A(7),A(8),A(9)/X'24',X'42',X'81'/
ØØ65Ø  C     DEFINE BACKGROUND ARRAY HERE
ØØ7ØØ        DATA  B(1),B(2)/1,Ø/
ØØ8ØØ        CALL GRPINI(Ø)
ØØ9ØØ        CALL CLS
Ø1ØØØ        CALL SETXY(3ØØ,1ØØ)
Ø11ØØ        CALL CIRCLE(15Ø,1,Ø.Ø,Ø.Ø,Ø.Ø)
Ø12ØØ        BORDER=1
Ø13ØØ        CALL PAINTT(A,BORDER,B)
Ø14ØØ        END
```

PRESET
Sets Pixel ON/OFF

PRESET (color)

color is of LOGICAL type, specifies whether a pixel is to be set ON or OFF and is an integer expression of either Ø (OFF) or 1 (ON).

PRESET sets the pixel defined by the current (x,y) coordinates either ON or OFF.

Example

```
CALL PRESET(Ø)
```

Sample Program

```
ØØ1ØØ   C       PRESET EXAMPLE
ØØ2ØØ           LOGICAL COLOR
ØØ3ØØ           COLOR=1
ØØ4ØØ           CALL GRPINI(Ø)
ØØ5ØØ           CALL CLS
ØØ6ØØ   C       SET PIXEL TO ON
ØØ6ØØ           CALL SETXY(3ØØ,12Ø)
ØØ8ØØ           CALL PRESET(COLOR)
ØØ9ØØ   C       TEST PIXEL WHETHER ON OR OFF
Ø1ØØØ           K=POINT(M)
Ø11ØØ   3Ø      WRITE (3,35)K
Ø12ØØ   35      FORMAT ('2','PIXEL VALUE IS',I4)
Ø13ØØ           END
```

PSET
Sets Pixel ON/OFF

PSET (color)

color is of LOGICAL type, specifies whether a pixel is to be set ON or OFF and is an integer expression of either Ø (OFF) or 1 (ON).

PSET sets the pixel defined by the current (x,y) coordinates either ON or OFF.

Example

```
CALL PSET(Ø)
```

Sample Program

```
ØØ1ØØ  C     PSET EXAMPLE
ØØ2ØØ        LOGICAL COLOR
ØØ3ØØ        LOGICAL POINT
ØØ4ØØ        COLOR=1
ØØ5ØØ        CALL GRPINI(Ø)
ØØ6ØØ        CALL CLS
ØØ7ØØ  C     SET PIXEL TO ON
ØØ8ØØ        CALL SETXY(3ØØ,12Ø)
ØØ9ØØ        CALL PSET(COLOR)
Ø1ØØØ  C     TEST PIXEL WHETHER ON OR OFF
Ø11ØØ        K=POINT(M)
Ø12ØØ        WRITE (3,35)K
Ø13ØØ  35    FORMAT ('2','PIXEL VALUE IS',I4)
Ø14ØØ        END
```

PUT
Puts Stored Array onto Screen

PUT (array, action)

array is usually LOGICAL type, although any type is permissible. Specifies the array (stored with GET) to be restored.
action is of LOGICAL type and specifies how the data is to be written to the video. Action may be one of the following:

1 = OR	3 = PRESET
2 = AND	4 = PSET
	5 = XOR

PUT takes a rectangular pixel area that has been stored by GET and puts it on the screen at current x and y coordinates set by calling SETXY.

Example

```
CALL PUT (V,1)
```

Sample Program (see GET)

SCREEN
Selects Screen

SCREEN (switch)

switch is of LOGICAL type and specifies the type of screen display and may be one of the following:
Ø = Graphics Screen
1 = Text Screen

SCREEN lets you select the proper screen.

Example

```
CALL SCREEN(Ø)
```

Sample Program

This example turns off the graphics display, draws a circle, then turns on the graphics display. The circle is then visible.

```
ØØØ1Ø  C     EXAMPLE FOR SCREEN
ØØØ2Ø        LOGICAL CMD
ØØØ4Ø        CMD=1
ØØØ5Ø        CALL GRPINI(Ø)
ØØØ6Ø        CALL CLS
ØØØ7Ø        CALL SCREEN(CMD)
ØØØ8Ø        CALL SETXY(3ØØ,12Ø)
ØØØ9Ø        CALL CIRCLE(1ØØ,1,Ø.Ø,Ø.Ø,Ø.Ø)
ØØ1ØØ        CALL PAINT(1,1)
ØØ11Ø        DO 2Ø I=1,1ØØØØ
ØØ12Ø  2Ø    CONTINUE
ØØ13Ø        CMD=Ø
ØØ14Ø        CALL SCREEN(CMD)
ØØ15Ø        END
```

SETXY
Sets Coordinates

SETXY(x,y)

(x,y) are INTEGER type and represent coordinates on the Graphics Screen.

SETXY sets and holds both current and previous X- and Y-coordinates. When a new coordinate is given, it is designated as the "current coordinate" and the last coordinate is designated as the "previous coordinate." If a new coordinate is specified, the "previous coordinate" is lost and the "current coordinate" becomes the "previous coordinate."

Example

```
CALL SETXY(1ØØ,1ØØ)
```

Sample Program (see LINE)

SETXYR
Sets Relative Coordinates

SETXYR(p1,p2)

(p1,p2) are INTEGER type and represent Relative Coordinates on the Graphics Screen.

SETXYR sets the current (x,y) coordinates relative to the previously set (x,y) coordinates. For example, if the "current" coordinates are (1ØØ,1ØØ), CALL SETXYR(1Ø,1Ø) will set the "current" coordinates to (11Ø,11Ø); the "previous" coordinates will then be (1ØØ,1ØØ).

Example

```
CALL SETXYR(3Ø,3Ø)
```

Sample Program

```
ØØØ1Ø  C    DRAW TWO INTERSECTING CIRCLES
ØØØ2Ø       CALL GRPINI(1)
ØØØ3Ø       CALL CLS
ØØØ4Ø       CALL SETXY(1ØØ,1ØØ)
ØØØ5Ø       CALL CIRCLE(5Ø,1,Ø.Ø,Ø.Ø,Ø.Ø)
ØØØ6Ø  C    DRAW SECOND CIRCLE WITH CENTER 2Ø
ØØØ7Ø  C    PIXELS TO THE RIGHT OF FIRST CIRCLE
ØØØ8Ø       CALL SETXYR(2Ø,Ø)
ØØØ9Ø       CALL CIRCLE(5Ø,1,Ø.Ø,Ø.Ø,Ø.Ø)
ØØ1ØØ      END
```

VIEW
Sets Viewport

VIEW(leftX,leftY,rightX,rightY,color,border)

leftX, leftY, rightX, rightY are INTEGER type and specify the viewport's parameters. leftX and rightX are numeric expressions from Ø to 639 and specify viewport's corner X-coordinates. leftY and rightY are numeric expressions from Ø to 239 and specify the viewport's corner Y-coordinates.

color is of LOGICAL type, specifies the OFF/ON color code and is a numeric expression of either Ø (OFF, black), 1 (ON, white), or -1 (viewport is not shaded).

border is of LOGICAL type, specifies the border color for the viewport and is an integer expression of either Ø (OFF, black), 1 (ON, white), or -1 (border is not drawn).

VIEW draws viewports on your screen. Graphics is displayed only in the last defined viewport.

The upper-left corner of viewport is read as (Ø,Ø) (the "relative origin") when creating items inside the viewport. All the other coordinates are read relative to this origin. However, the "absolute coordinates" of the viewport, as they are actually defined on the Graphics Cartesian system, are retained in memory and can be read using VIEW as a function.

Example

```
CALL VIEW(1ØØ,1ØØ,2ØØ,2ØØ,Ø,1)
```

Sample Program

```
ØØ1ØØ C     SAMPLE VIEW PROGRAM
ØØ2ØØ       LOGICAL COLOR,BORDER,K
ØØ3ØØ       INTEGER FVIEW
ØØ4ØØ       CALL GRPINI(1)
ØØ5ØØ       CALL CLS
ØØ5ØØ C     SET UP VIEW PORT
ØØ7ØØ       COLOR=Ø
ØØ8ØØ       BORDER=1
ØØ9ØØ       CALL VIEW(21Ø,8Ø,42Ø,16Ø,COLOR,BORDER)
Ø1ØØØ C     DRAW MULTIPLE CIRCLES
Ø11ØØ       CALL SETXY(1Ø5,4Ø)
Ø12ØØ       DO 2Ø I=1Ø,15Ø,1Ø
Ø13ØØ       CALL CIRCLE(I,1,Ø.Ø,Ø.Ø,Ø)
Ø14ØØ 2Ø    CONTINUE
Ø15ØØ C     DISPLAY VIEWPORT COORDINATES
Ø16ØØ       DO 4Ø I=1,4
Ø17ØØ       K=I-1
Ø18ØØ       J=FVIEW(K)
Ø19ØØ       WRITE (3,35)I,J
Ø2ØØØ 35    FORMAT ('2','VIEW PORT COORDINATE ',I4,' IS AT',I4)
Ø21ØØ 4Ø    CONTINUE
Ø22ØØ C     PRINT EMPTY LINES
Ø23ØØ       DO 6Ø I=1,6
Ø24ØØ       WRITE (3,5Ø)
Ø25ØØ 5Ø    FORMAT (1H1)
Ø26ØØ 6Ø    CONTINUE
Ø27ØØ       END
```

The following two descriptions are functions in the Graphics Subroutine Library and must be declared as LOGICAL and INTEGER, respectively, in any routine that uses them.

Functions

POINT
Reads Pixel Value at Current Coordinates

```
V=POINT(X)

  X is a dummy variable needed to set up the proper
   FORTRAN linkage to the POINT routine.
```

POINT returns the OFF/ON pixel value at current x and y coordinate as specified by SETXY or SETXYR. If the point is not in the current viewport, POINT returns -1.

Example

```
K=POINT(M)
```

Sample Program (see PSET)

FVIEW
Reads Viewport's Parameters

```
FVIEW (n)

  n is of LOGICAL type and is an integer expression
   from Ø to 3.
```

FVIEW returns the specified viewport parameter:

Ø = returns the left X-coordinate
1 = returns the left Y-coordinate
2 = returns the right X-coordinate
3 = returns the right Y-coordinate

Example

```
I=FVIEW(Ø)
```

Sample Program (see VIEW)

5/ Programming the Graphics Board

The Graphics Board provides 64Ø X 24Ø byte addressable pixels on a TRS-8Ø Model III. The Graphics Board contains 32K of screen RAM to store video data consisting of four 64K RAMs which are double accessed for 8 bytes of data. Regular alphanumeric data is stored in the static RAM on the Video Board. The Graphics Board uses separate hardware to generate a 64Ø X 24Ø display, so only one screen may be displayed at a time. If the video is switched from Text to Graphics Screen very rapidly, the Video display may lose horizontal/vertical synchronization.

I/O port mapping is used to read and write data to the board. The Board is addressable at 8Ø-83 Hex.

There are four internal registers which can be written to or read on the board. They are as follows:

1. **X-Position** - X-address (Ø to 127) for data write only. (Ø to 79 for display.)
2. **Y-Position** - Y-address (Ø to 255) for data write only. (Ø to 238 for display.)
3. **Data** - Graphics data in "byte" form. Each byte turns on or off 8 consecutive horizontal dots.
4. **Options** - 8 flags which turn on or off the user programmable options (Write only).

The I/O port mapping of the board is:

- xØ - X-Register Write. (8Ø)
- x1 - Y-Register Write. (81)
- x2 - Video data read or write. (82)
- x3 - Options write. (83)

where x denotes the upper nibble of the I/O boundary as set by the DIP Switches. They are set by the factory at 8ØH.

The Graphics Board uses X-Y addressing to locate the start of a Graphics data byte. The upper-left of the screen is (Ø,Ø) while the lower-right is (Ø79,239). If the bit is a 1, the dot will be ON. For example, if you wanted to turn

on the 5th dot on the top row, the registers would contain: X POSITION=Ø, Y POSITION=Ø, DATA=(ØØØØ1ØØØ)=Ø8H. Note that in calculating points to plot, the Y-position is correct for a single dot. Only the X-position must be corrected to compensate for the byte addressing. This can be accomplished in a simple subroutine.

Line Drawing Options

There are two 8-bit counters which act as latches for the X- and Y-address. You may select, through the options register, if they are to automatically count after a read or write to graphic memory. Also, the counters may increment or decrement independently. These counters do not count to their respective endpoints and reset. Instead, they will overflow past displayable video addresses. Therefore, the software should not allow the counters to go past 79 and 239. However, these extra memory locations may be used for data storage.

Examples

The following are brief examples on how to use the Graphics Board.

Read the video byte at X=Ø, Y=Ø

```
        XOR     A               ;CLEAR A
        OUT     (8ØH),A         ;OUTPUT X ADDRESS
        OUT     (81H),A         ;OUTPUT Y ADDRESS
        IN      A,(82H)         ;READ VIDEO BYTE
```

Draw a line from X=Ø,Y=Ø to X=639, Y=Ø using the hardware line drawing

```
        LD      B,79            ;B HAS CHARACTER COUNT
        LD      A,ØB1H          ;OPTIONS:INCREMENT X AFTER WRITE
                                ;1Ø11ØØØ1 Binary
        OUT     (83H),A
        XOR     A
        OUT     (8ØH),A         ;OUT X ADDRESS STARTING
        OUT     (81H),A         ;OUTPUT Y ADDRESS
        LD      A,ØFFH          ;LOAD A WITH ALL DOTS ON
LOOP    OUT     (82H),A         ;OUTPUT DOTS
        DJNZ    LOOP            ;OUTPUT NUMBER IN B REGISTER
```

Options Programming

No.	Option	Description
Ø	GRAPHICS/ALPHA*	Turns graphics ON and OFF. "1" turns graphics ON.
1	NOT USED	
2	XREG DEC/INC*	Selects whether X decrements or increments. "1" selects decrement.
3	YREG DEC/INC*	Selects whether Y decrements or increments. "1" selects decrement.
4	X CLK RD*	If address clocking is desired, a "Ø" clocks the X address up or down AFTER a Read depending on the status of BIT 2.
5	Y CLK RD*	If address clocking is desired, a "Ø" clocks the Y address up or down AFTER a Read depending on the status of BIT 3.
6	X CLK WR*	A "Ø" clocks AFTER a Write.
7	Y CLK WR*	A "Ø" clocks AFTER a Write.

Table 9. Options Programming

SERVICE POLICY

Radio Shack's nationwide network of service facilities provides quick, convenient, and reliable repair services for all of its computer products, in most instances. Warranty service will be performed in accordance with Radio Shack's Limited Warranty. Non-warranty service will be provided at reasonable parts and labor costs.

Because of the sensitivity of computer equipment, and the problems which can result from improper servicing, the following limitations also apply to the services offered by Radio Shack:

1. If any of the warranty seals on any Radio Shack computer products are broken, Radio Shack reserves the right to refuse to service the equipment or to void any remaining warranty on the equipment.
2. If any Radio Shack computer equipment has been modified so that it is not within manufacturer's specifications, including, but not limited to, the installation of any non-Radio Shack parts, components, or replacement boards, then Radio Shack reserves the right to refuse to service the equipment, void any remaining warranty, remove and replace any non-Radio Shack part found in the equipment, and perform whatever modifications are necessary to return the equipment to original factory manufacturer's specifications.
3. The cost for the labor and parts required to return the Radio Shack computer equipment to original manufacturer's specifications will be charged to the customer in addition to the normal repair charge.

RADIO SHACK, A DIVISION OF TANDY CORPORATION

U.S.A.: FORT WORTH, TEXAS 76102
CANADA: BARRIE, ONTARIO L4M 4W5

TANDY CORPORATION

AUSTRALIA	BELGIUM	U. K.
91 KURRAJONG ROAD MOUNT DRUITT, N.S.W. 2770	PARC INDUSTRIEL DE NANINNE 5140 NANINNE	BILSTON ROAD WEDNESBURY WEST MIDLANDS WS10 7JN

8749382-03/83-SP

Printed in U.S.A.