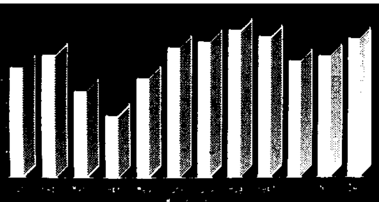
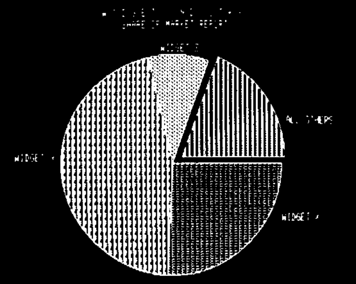
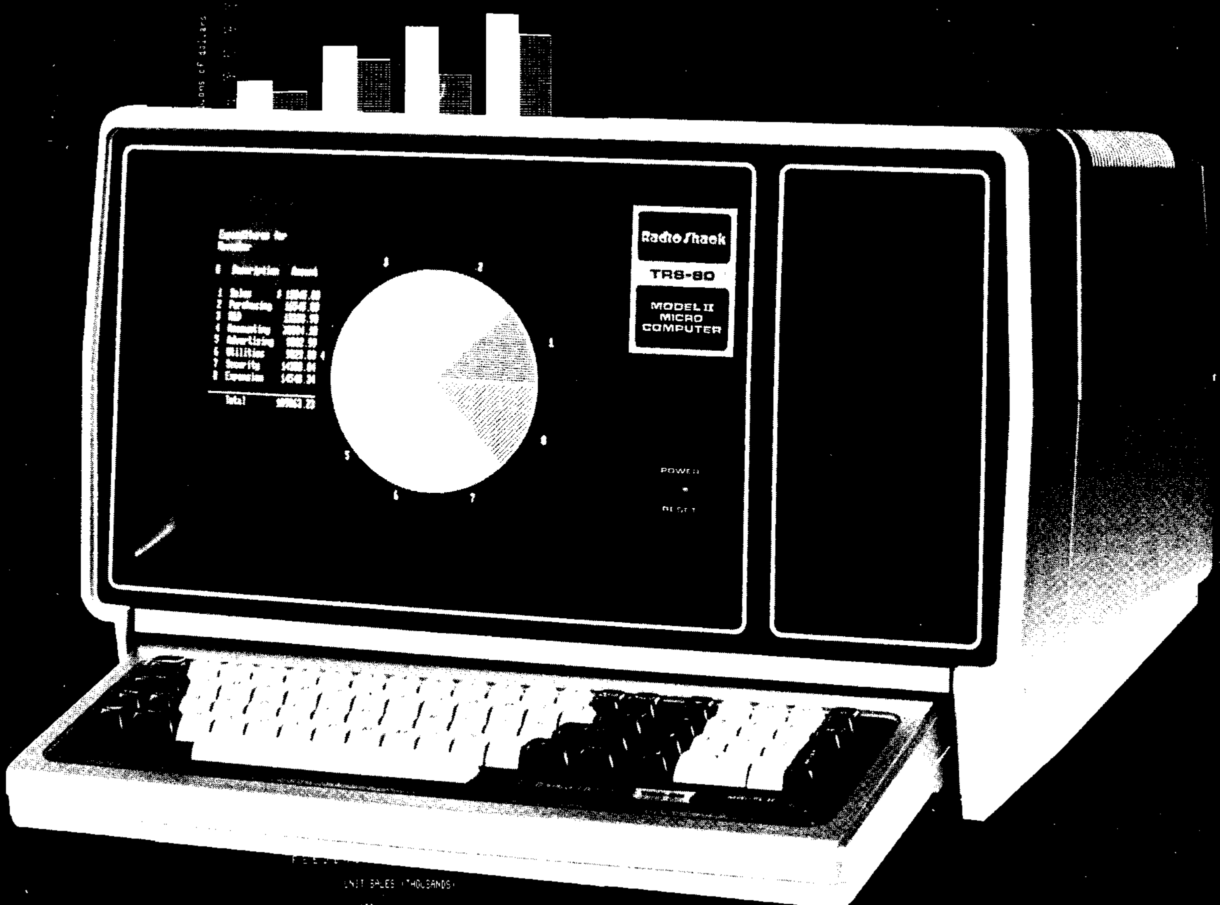
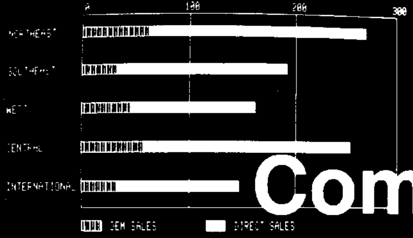




Radio Shack®

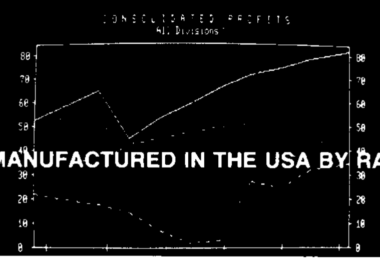
"The biggest name in little computers"





TRS-80®

Computer Graphics



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

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.

Reproduction or use without express written permission from Tandy Corporation, of any portion of this manual is prohibited. While reasonable efforts have been taken in the preparation of this manual to assure its accuracy, Tandy Corporation assumes no liability resulting from any errors or omissions in this manual, or from the use of the information obtained herein.

Contents

To Our Customers . . .

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

The TRS-8Ø Computer Graphics package includes a:

- Graphics Diskette
- Graphics Operation Manual

However, before you can use this package, your Computer must be modified by a qualified Radio Shack service technician. Your Model II must also have 64K of RAM (Random Access Memory). The Computer Graphics package will run on the TRS-8Ø® Hard Disk (Radio Shack Catalog Number 26-415Ø) if your Hard Disk is operating under either TRSDOS-HD (version 4.Ø) or TRSDOS-II (4.1).

Included on the Graphics diskette are:

- TRSDOS 2.Øa
- Model II BASIC
- Model II Graphics BASIC (BASICG)
- Model II Graphics Subroutine Library
- Graphics Utilities
- COBOL Interface Routines (2 files)
- Sample Programs in BASIC, Assembly, FORTRAN, and COBOL.

To print graphic displays, you can use any Radio Shack printer that has graphic capabilities such as Line Printer VII (26-1167) or a Line Printer VIII (26-1168).

Note that you can also utilize the Graphics Subroutine Library with several languages, including Assembly (26-47Ø2), FORTRAN (26-47Ø1), and COBOL (26-47Ø3).

About This Manual . . .

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

- Computer Graphics Overview
- Graphics BASIC (BASICG) Language Description
- Graphics Utilities

- FORTRAN Description
- Assembly Language Description
- COBOL 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 II 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 II 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 appropriate TRS-8Ø language packages. Then read Chapters 1, 2, 3, and 4 as well as Appendixes C, D, E, and F of this manual.

For Assembly Language applications, read Chapters 1, 2, 3, 4, 5, and 7; then refer to Appendixes D, E, and F.

COBOL programmers should also read Chapters 1, 2, and 3, along Chapter 6 and Appendixes D, E, and F.

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.

<KEYBOARD> — Any word or character contained within a box 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(diskette name)
Note that with TRSDOS-II, d (Drive) can be any number between Ø-7.

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.

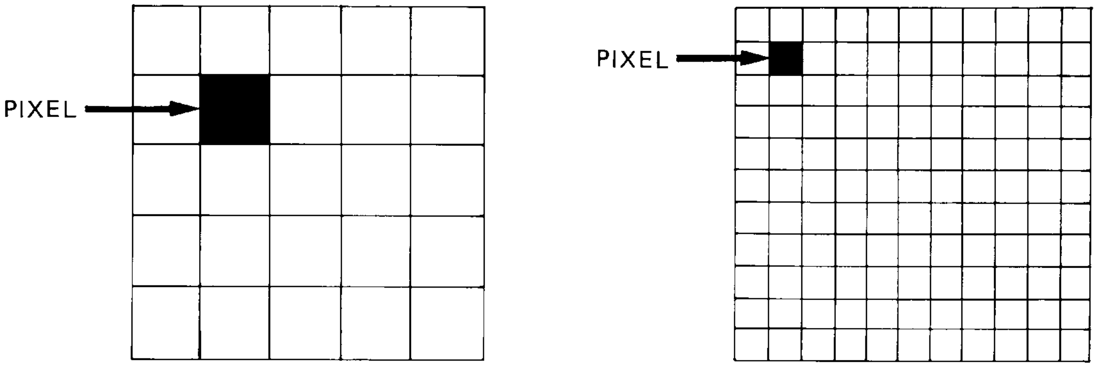
TRS-8Ø Computer Graphics has two colors:

- 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.

Excellent graphics packages, such as TRS-8Ø Computer Graphics, have 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.



Lower resolution **Higher resolution**

Figure 1. Resolution

Since the TRS-8Ø has high resolution -- 64Ø pixels on the X-axis (Ø to 639, inclusive) and 24Ø pixels on the Y-axis (Ø to 239, inclusive) -- 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 TRS-8Ø 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 or together. Note: The Graphics Screen will not automatically be cleared when you return to TRSDOS. It will be cleared when you re-enter BASICG unless you use the -G option. (See Options to Loading BASICG.)

The Graphics Screen can be displayed at the same time as the Text Screen. However, if the same pixel in Text and Graphic Screens overlay each other (i.e., both Screens turn the same pixel ON), the pixel will be turned OFF.

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 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 Graphics, you can directly reference these coordinates as you draw.

About Ranges...

Some TRS-8Ø Graphics commands accept values within the Model II 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.

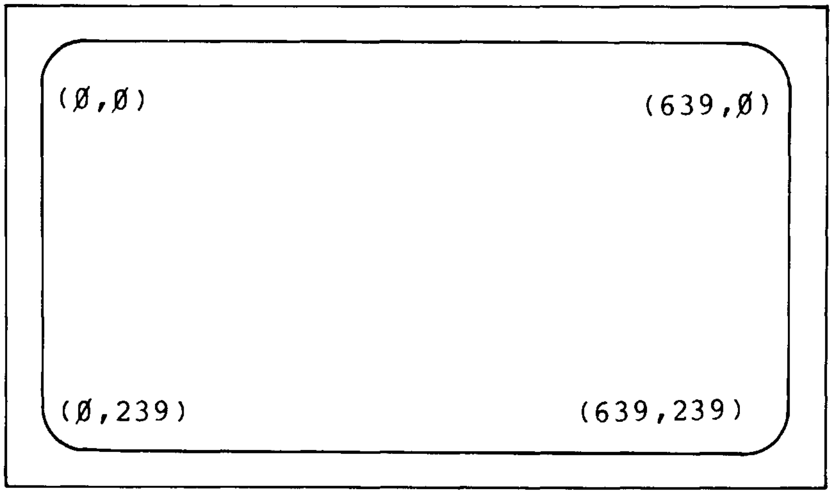


Figure 2. Graphics Visible Screen

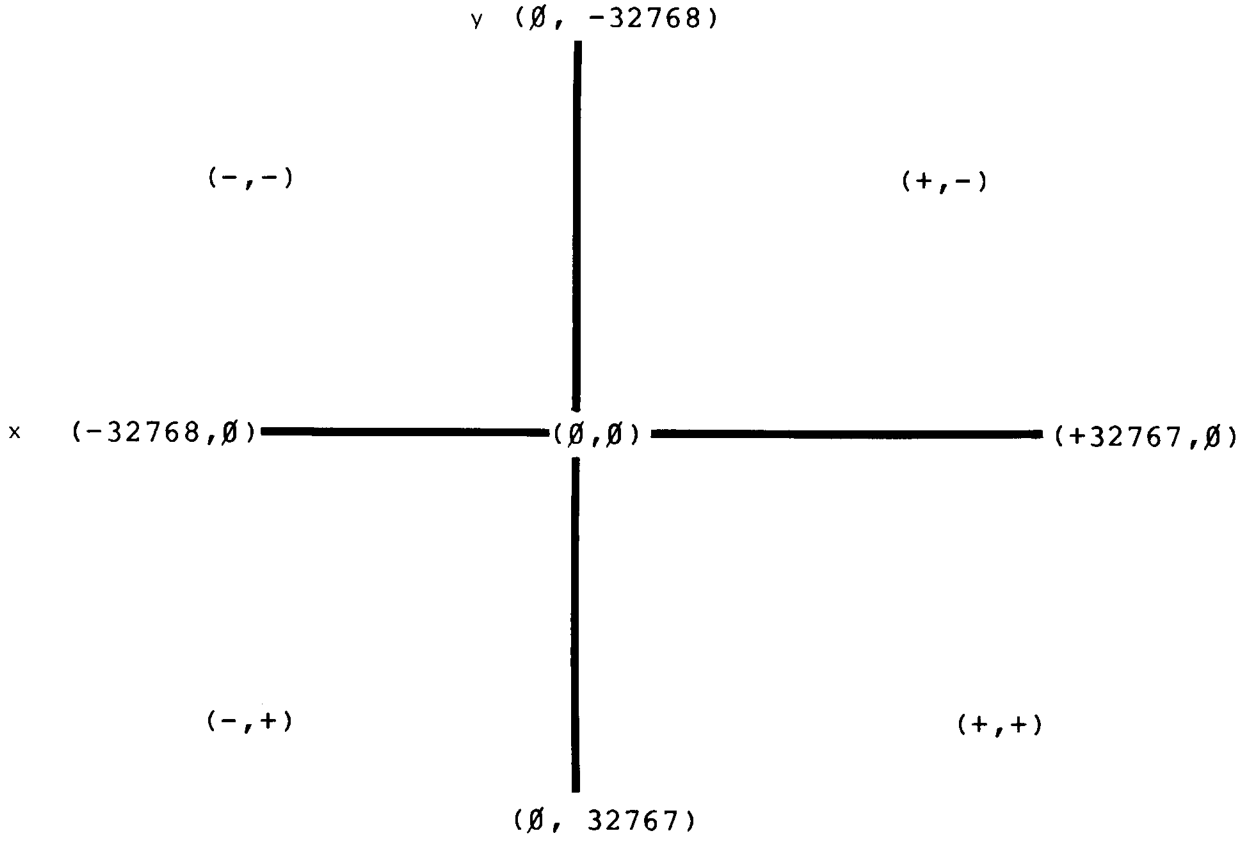


Figure 3. Graphics "Imaginary" Cartesian System

2/ Graphics BASIC

Graphics BASIC (BASICG) vs. BASIC

The Graphics BASIC file on the supplied diskette is called 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 will occur. See Appendix B for a list of error messages.

The only way to load a file from one BASIC to the other is to first save the file from either BASICG or BASIC in ASCII (**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 5K more memory than BASIC. When you enter BASIC without files (i.e., you do not use the -F: option), there are 336Ø8 bytes free. When you enter BASICG without files, there are 27784 bytes free. Some Graphics Commands use Free Memory. This means that the larger your BASIC programs are, the more limitations on your Graphic 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, semi-circle, etc.
CLS	Clears either the Text or Graphics Screen or both.
GET	Reads contents of a rectangle on the Graphics Screen into an array for future use by PUT.
LINE	Draws a line from the startpoint to 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).
PSET	Sets an individual dot (pixel) ON (or OFF).
PUT	Stores graphics from an array onto the Graphics Screen.
SCREEN	Turns Graphics Screen on or off and selects display speed.
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 II Owner's Manual for information on BACKUP.

To load BASICG:

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

```
BASICG <ENTER>
```

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

Options to Loading BASICG

There are three options you can use when loading BASICG. When you enter Graphics BASIC without an option (i.e., BASICG <ENTER>), the Graphics Screen is cleared.

```
BASICG -G: <ENTER>
```

The -G option lets you enter BASICG without clearing the Graphics Screen.

```
          BASICG  -F:files <ENTER>
```

This option works exactly like -F which is described in the Model II Owner's Manual. Refer to that manual for details.

```
          BASICG -M:address <ENTER>
```

This option also works exactly as described in the Model II Owner's Manual.

These options may be combined. For example, if you do not want to clear the Graphics Screen but you do want to allocate three files, type:

```
          BASICG -G: -F:3 <ENTER>
```

Additionally, a BASICG program name in standard format can be specified when you enter BASICG from TRSDOS. Upon entry into BASICG, the program will be loaded and executed.

Remember that Model II numeric values are as follows:

Model II Numeric Values

Numeric Type	**Range**	**Storge 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 only 16)	8 bytes	123ØØØØ.ØØ 3.1415926535897932

Table 3

See your Model II Owner's Manual for more details on Numeric Data Types.

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 II Owner's Manual for more information).

CIRCLE
Draws Circle, Semi-Circle, 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 a positive 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:

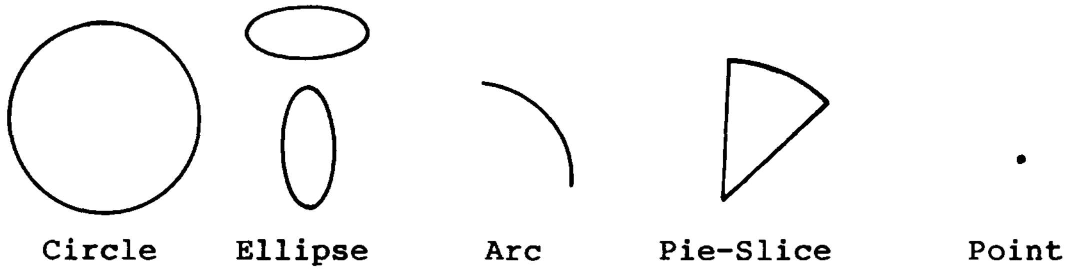


Figure 4. Types of Displays with CIRCLE

With CIRCLE, you can enter values for PI (and 2 x PI) up to 37 significant digits:

3.1415926535897932384626433832795Ø28841
6.2831853Ø7179586476925286766559ØØ57682

without getting an overflow error. 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
```

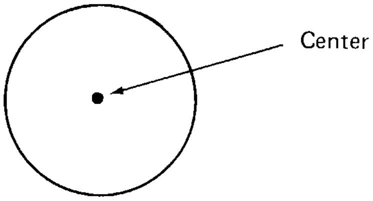


Figure 5. Center of Circle

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.

The radius is either on the X-axis or Y-axis, depending on 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

```
1Ø CIRCLE(32Ø,12Ø),1ØØ
```

This example draws a circle. The radius is 1ØØ and the centerpoint is (32Ø,12Ø).

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 Ø.

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

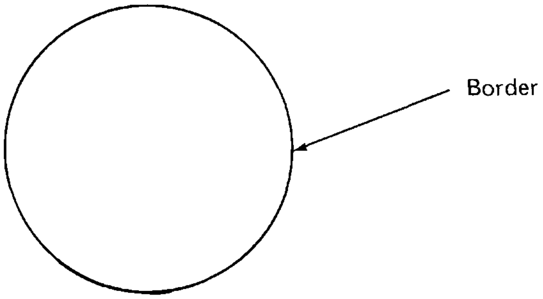


Figure 6. Border of Circle

start/end
Startpoint/Endpoint of Circle

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

If you do not enter start and end, the default values of Ø 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 -Ø. To draw a radius with start or end as Ø, you must use -Ø.ØØØ...Ø1.

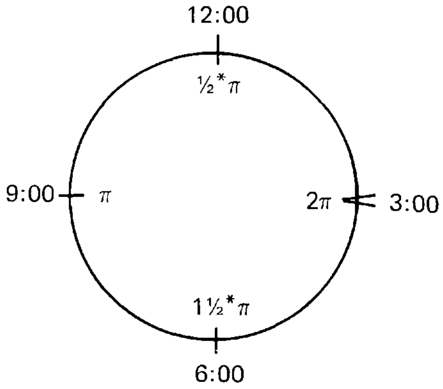


Figure 7. Clock/Radian Equivalents

Degrees	Radians	Clock Equivalent
Ø	Ø	3:ØØ
9Ø	1.57	12:ØØ
18Ø	3.14	9:ØØ
27Ø	4.71	6:ØØ
36Ø	6.28	3:ØØ

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.

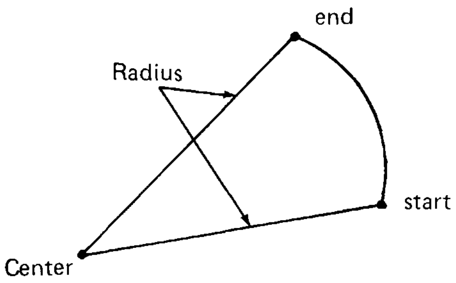


Figure 8. CIRCLE's (-) start, (-) end

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

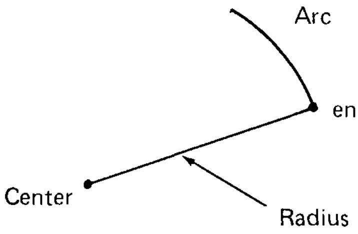


Figure 9. CIRCLE's (+) start, (-) end

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

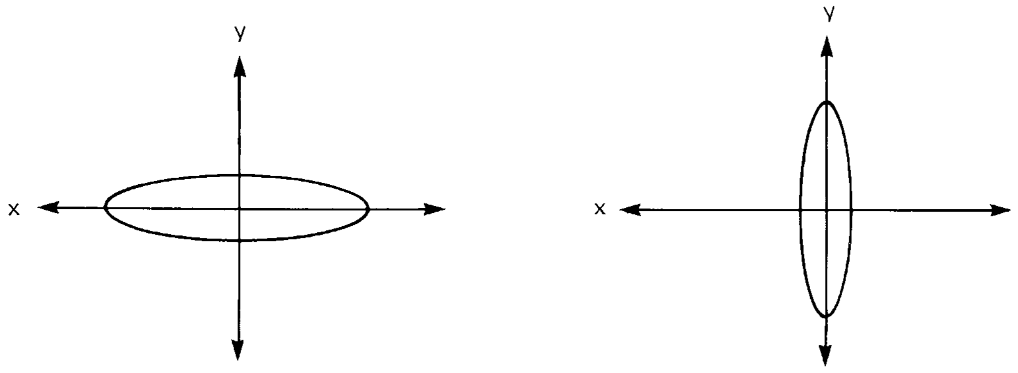
You can draw ellipses by varying the aspect ratio from the default value (.5) for a circle (and semi-circle).

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}). See your Model II Owner's Manual for more information.

Hints and Tips about aspect ratio:

- Entering .5 as the ratio produces a circle.
- Number between Ø and .5 produce an ellipse with a major axis on X.
- Number 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 semi-circle 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

```
          5 CLS 2
          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.

CLS
Clears Screen(s)

CLS n

n is a integer expression from Ø to 2 and specifies which Screen (Text or Graphics or both) is to be cleared. CLS Ø clears the Text Screen, CLS 1 clears the Graphic Screen, CLS 2 clears both the Graphics and Text Screens. n is optional; if omitted, Ø is used.

CLS clears the Screen according to the specified variable.

Examples

```
1Ø CIRCLE(32Ø,12Ø),1ØØ,1
```

This program line will draw a circle. Now type:

```
CLS <ENTER>
```

and the Text Screen will be cleared but the Graphics Screen will remain.

Type:

```
CLS 2 <ENTER>
```

and both the Graphics and Text Screen will be cleared.

Run the program again and type:

```
CLS 1 <ENTER>
```

and the Graphics Screen will be cleared but the Text Screen will remain.

GET
Reads 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 II Owner's Manual. BASIC recognizes only the GET syntax described in the Model II Owner's 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 highest 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 = (xl,yl) lower-left corner = endpoint = (x2,y2)

Size (in pixels):	width = x2-xl+l
	length = y2-yl+l

Examples

```
GET(1Ø,1Ø)-(8Ø,5Ø),V
```

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

- 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

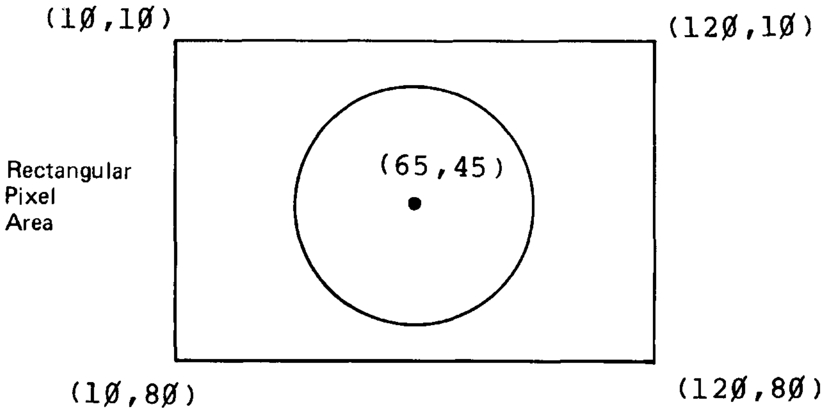


Figure 11

```
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)).

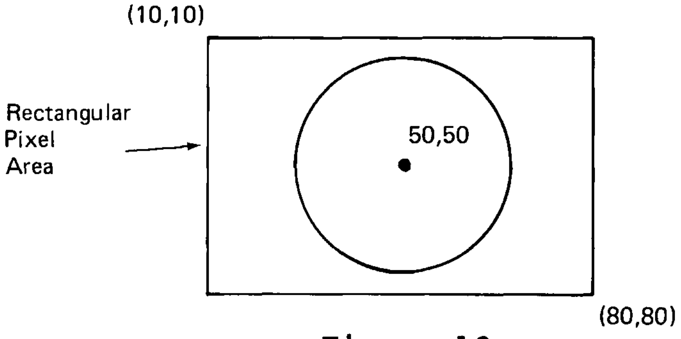


Figure 12

```
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.

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 only) 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 option, the "diagonal" line is drawn -- not the perimeter of the rectangle. When you do specify the B option, the perimeter is drawn but not the diagonal line.

```
LINE(14Ø,8Ø)-(5ØØ,2ØØ),1,B
```

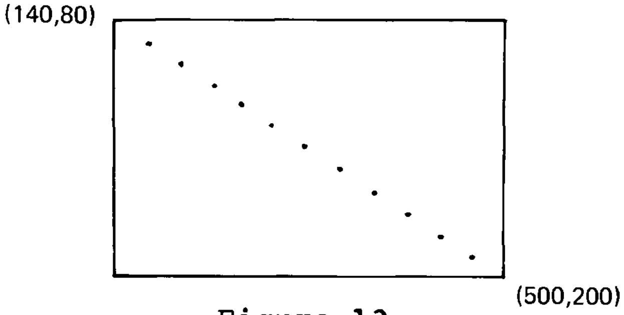


Figure 13

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

Example

```
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, Ø 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 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 x 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.

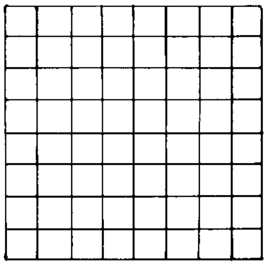


Figure 14. Example of an 8-by-8 Pixel Grid

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 tile 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 a 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 totally 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 two-digit 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	Hexadecimal
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
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 paints 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 Screen white, draws circle and paints 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.

```
1Ø TILE$(Ø)=CHR$(&H22)+CHR$(&HØØ)
2Ø TILE$(1)=CHR$(&HFF)+CHR$(&HØØ)
3Ø TILE$(2)=CHR$(&H99)+CHR$(&H66)
```

```
40 TILE$(3)=CHR$(&H99)
50 TILE$(4=CHR$(&HFF)
60 TILE$(5)=CHR$(&HF0)+CHR$(&HF0)+CHR$(&H0F)+CHR$(&H0F)
70 TILE$(6)=CHR$(&H3C)+CHR$(&H3C)+CHR$(&HFF)
80 TILE$(7)=CHR$(&H03)+CHR$(&H0C)+CHR$(&H30)+CHR$(&HC0)
90 A$=TILE$(0)+TILE$(1)+TILE$(2)+TILE$(3)+TILE$(4)+TILE$(5)
   +TILE$(6)+TILE$(7)
100 PAINT(300,100),A$,1
```

This example paints the Screen with a tiling pattern made up of eight individually defined tile strings (0-7).

POINT (function)
Returns Pixel Value

POINT(x,y)

x specifies an X-coordinate and is an integer expression.
y specifies a Y-coordinate and is an integer expression.
values returns with POINT are:
0 (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(800,500)) return a -1, signifying the pixel is off the Screen.

Example

```
10 PSET(300,100),1
20 PRINT POINT(300,100)
```

Reads and prints the value of the pixel at the point's coordinates (300,100) and displays its value: 1

```
PRINT POINT(3000,1000)
```

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

```
          GRAPHICS BASIC!
```

```
          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 CLS2
          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 1s are returned for 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, a 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.

PSET
Sets Pixel ON (or OFF)

PSET(x,y),switch

x specifies an X-coordinate and is an integer expression.
y specifies a 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.

Examples

```
1Ø A=1
2Ø PSET (5Ø,5Ø),A
```

Turns the pixel located at the specified coordinates ON.

```
1Ø PSET (RND(64Ø),RND(24Ø)),1
2Ø GOTO 1Ø
```

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

```
          PSET(-3ØØ,-2ØØ),1
```

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

```
          1Ø PSET (32Ø,12Ø),1
          2Ø A$=INKEY$: IF A$= "" THEN 2Ø
          3Ø PSET(32Ø,12Ø),Ø
```

Line 1Ø sets ("turns ON") a pixel; line 3Ø 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 expressicn 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 II Owner's Manual. BASIC recognizes only the PUT syntax described in the Model II Owner's 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.

For example:

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

After GET-ting, 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.

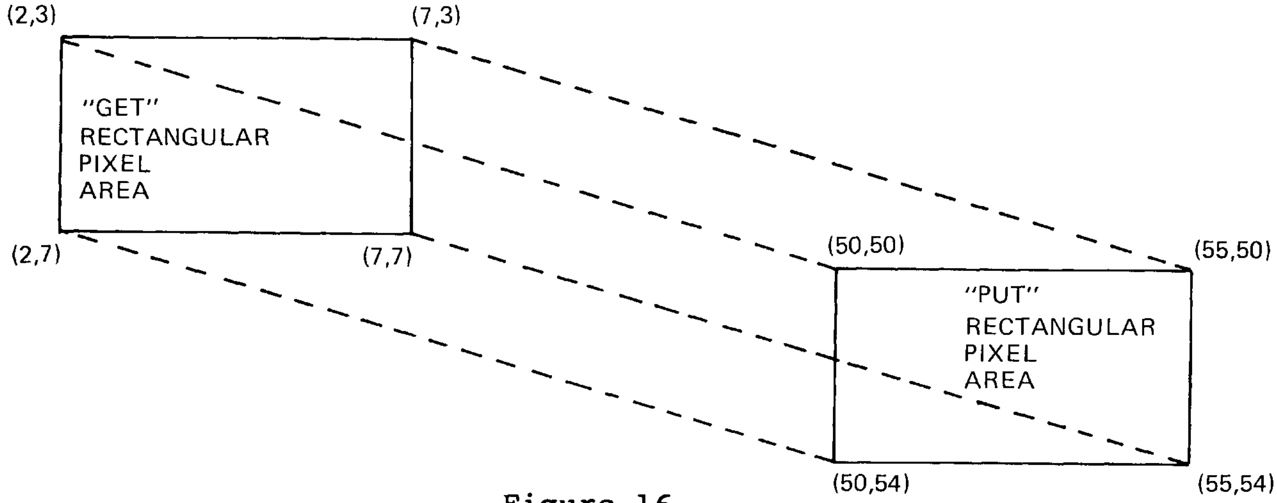


Figure 16

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

These operators are used in Graphics BASIC to test the true/false ("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 such as the Line Printer VII or VIII. See the section of this manual entitled Graphic Utilities for more details on using the Graphics package with a printer.

```
1Ø DATA "OR","AND","PRESET","PSET","XOR"
2Ø CLS 2
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$
17Ø PRINT @(R,17),A$;
18Ø PRINT @(R,45), "= ";
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 PRINT @Ø, " ";
27Ø SYSTEM "VDOGRPH"
28Ø SYSTEM "GPRINT"
```

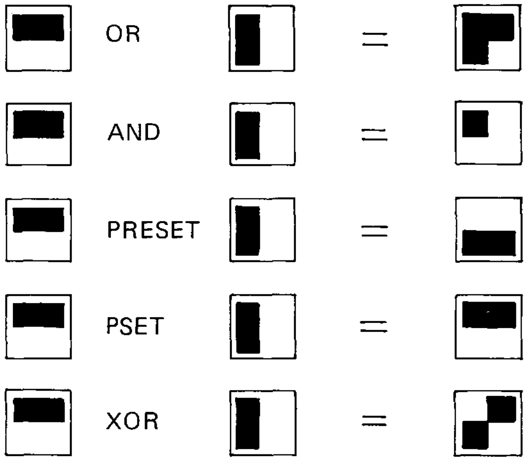


Figure 17

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)
2Ø CIRCLE (3Ø,3Ø),1Ø
3Ø GET (1Ø,1Ø)-(4Ø,4Ø),V%
4Ø FOR I=1 TO 5ØØ: NEXT I
5Ø CLS 1
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 statement), it disappears and then reappears on the Screen -- (11Ø,11Ø) -- program line 5Ø.

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 FOR X=1 TO 5
 2Ø FOR Y=1 TO 3
 3Ø PSET (1ØØ+X, 1ØØ+Y)
 4Ø NEXT Y: NEXT X
 5Ø A$=INKEY$: IF A$=""THEN 5Ø
 6Ø DIM V%(5)
 7Ø GET (1ØØ,1ØØ)-(1Ø6,1Ø4),V%
 8Ø FOR A=1Ø TO 1ØØ STEP 1Ø
 9Ø FOR B=1Ø TO 1ØØ STEP 1Ø
1ØØ PUT (A,B),V%,PSET
11Ø A$=INKEY$: IF A$=""THEN 11Ø
12Ø NEXT B: NEXT A
```

```
 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
Sets Screen/Graphics Speed

SCREEN type

type specifies which "Screen" to use and is a numeric expression from Ø to 3.

Ø = Graphics ON/ normal speed
1 = Graphics OFF/normal speed
2 = Graphics ON/ high speed
3 = Graphics OFF/high speed

SCREEN lets you set the proper Screen and Screen speed. SCREEN 2 and 3 produce graphics more rapidly than SCREEN Ø and 1. Any value greater than 3 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 add to it. With SCREEN, you can remove the graphics display, add to the program, and then return to the Graphics Screen.

Graphics can produce a "flashing" on the Screen if the high speed option is specified. With normal speed graphic presentations, however, this flashing will not occur.

Examples

```
1Ø SCREEN 3
2Ø LINE (15Ø,15Ø)-(2ØØ,2ØØ)
```

The Computer executes the short program but the Graphics Screen cannot display the graphics because of the SCREEN 3 command. To display the line, type: SCREEN Ø <ENTER>

```
1Ø CLS
2Ø SCREEN 3
3Ø LINE(1Ø,1Ø)-(255,191)
4Ø LINE(Ø,191)-(255,Ø)
```

```
50 A$=INKEY$: IF A$=""THEN 50
60 SCREEN 0
70 A$=INKEY$: IF A$=""THEN 70
80 GOTO 10
```

The Computer executes the program (draws two intersecting lines) but the Screen cannot display the graphics because of SCREEN 3. By pressing any key, the graphics are displayed because of SCREEN 0.

```
10 CIRCLE (200,100),100
20 PAINT (200,100),"44",1
```

Now run the program and type:

```
SCREEN 3 <ENTER>
```

This command turns the Graphics Screen OFF. Type:

```
SCREEN 0 <ENTER>
```

This command turns the Graphics Screen back ON.
By entering the SCREEN 3 and SCREEN 0 commands, you can alternately turn the Graphics Screen ON and OFF 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:

- CLS 1 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 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 those coordinates. For example:

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

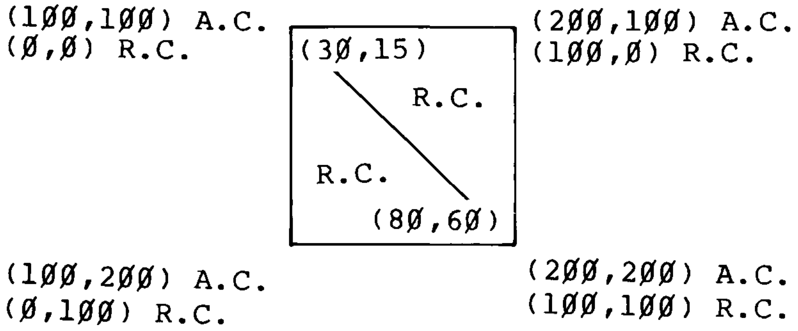


Figure 18

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).

```
1Ø VIEW (1Ø,1Ø)-(6ØØ,2ØØ),Ø,1
2Ø VIEW (5Ø,5Ø)-(1ØØ,1ØØ),Ø,1
3Ø LINE(RND(5ØØ),RND(19Ø))-(RND(5ØØ),RND(19Ø))
4Ø GOTO 3Ø
```

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.

```
1Ø VIEW(8Ø,8Ø)-(4ØØ,2ØØ),Ø,1
2Ø VIEW(1ØØ,9Ø)-(3ØØ,17Ø),Ø,1
3Ø VIEW(12Ø,1ØØ)-(2ØØ,2ØØ),Ø,1
4Ø VIEW(5Ø,5Ø)-(1ØØ,1ØØ),Ø,1
```

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

```
1Ø VIEW(21Ø,8Ø)-(42Ø,16Ø),Ø,1
2Ø CIRCLE(3ØØ,12Ø),18Ø,1
3Ø LINE(15,15)-(6Ø,6Ø),1
4Ø CIRCLE(9Ø,4Ø),5Ø,1
5Ø LINE(4Ø,3Ø)-(5ØØ,3Ø),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.

```
1Ø VIEW (19Ø,7Ø)-(44Ø,18Ø),Ø,1
2Ø CIRCLE (3ØØ,14Ø),17Ø,1
3Ø CIRCLE (1ØØ,23Ø),4ØØ,1
4Ø LINE (1Ø,1Ø)-(5ØØ,23Ø),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-axes 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 left X-coordinate
- 1 returns the left Y-coordinate
- 2 returns the right X-coordinate
- 3 returns the 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

Returns coordinates of last-defined viewport.

3/ Graphics Utilities

There are seven 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, Assembly, FORTRAN, and COBOL. 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.
- Transfer Text Screen displays (video memory) to graphics memory.
- Print graphic displays on a graphics printer.
- Turn graphics display OFF or ON.
- Clear graphics memory.

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

To use these utilities from an assembly-language program, use the supervisor call DOSCMD (function code 37) or RETCMD (function code 38) to send a command to TRSDOS. Control returns to your program if you use RETCMD.

To call these routines from FORTRAN, see the Subprogram Linkage section of your TRS-8Ø Model II FORTRAN Manual (26-47Ø1).

To call these routines from COBOL, refer to the COBOL section of this manual.

Note: These utilities load into high memory starting at FØØØ (hex); therefore, they cannot be used with SPOOL, DEBUG, HOST, 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 to printer.
GROFF	Turns Graphic Screen OFF.
GRON	Turns Graphic Screen ON.
GSAVE	Saves graphics memory to diskette.
VDOGRPH	Transfers Text Screen displays to graphics memory.

Table 6

GCLS
Clears Graphics Screen

```
          GCLS
```

GCLS clears the Graphics Screen by erasing the contents of graphics memory. 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:

```
          SYSTEM"GCLS" <ENTER>
```

or

```
          1ØØ SYSTEM"GCLS"
```

GLOAD
Loads Graphics Memory from Diskette

GLOAD <u>filename</u> <u>/ext</u> <u>.password</u> <u>:d</u> <u>(diskette name)</u>

<u>filename</u> consists of a letter followed by up to seven optional numbers or letters.
/<u>ext</u> is an optional name-extension; <u>ext</u> is a sequence of up to three numbers or letters.
<u>.password</u> is an optional password; <u>password</u> is a sequence of up to eight numbers or letters.
<u>:d</u> is an optional drive specification; <u>d</u> is one of the digits Ø through 7.
<u>(diskette name)</u> is an optional field of up to eight numbers or letters. If this field is included, it must be preceded by a drive specification.

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(GRAPHICS) <ENTER>
```

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

```
SYSTEM"GLOAD PROGRAM" <ENTER>
```

or

```
1ØØ SYSTEM "GLOAD PROGRAM"
```

GPRINT
Lists Graphic Display to Printer

GPRINT

GPRINT lets you print graphics memory on a graphic (dot-addressable) printer such as Radio Shack's Line Printer VII (26-1167) or VIII (26-1168). However, distortion will occur when Graphic routines are printed on the Line Printer VII and VIII. 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 the user 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 used to set printing parameters.

Most uses will require that you set FORMS to:

```
FORMS P=66 L=6Ø W=Ø C=Ø <ENTER>
```

Then type:

```
FORMS X <ENTER>
```

See your Model II and printer owner's manual for more details on setting printing parameters.

Examples

When TRSDOS READY is displayed, type:

```
GPRINT <ENTER>
```

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

```
SYSTEM"GPRINT" <ENTER>
```

or

```
1ØØ SYSTEM"GPRINT"
```

For a complete example of using GPRINT, see Appendix D.

GROFF
Turn Graphic 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:

```
SYSTEM "GROFF" <ENTER>
```

or

```
1ØØ SYSTEM "GROFF"
```

GRON
Turn Graphic 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:

```
SYSTEM "GRON" <ENTER>
```

or

```
          1ØØ SYSTEM "GRON"
```

GSAVE
Saves Graphics Memory to Diskette

GSAVE <u>filename</u> <u>/ext</u> <u>.password</u> <u>:d</u> <u>(diskette name)</u>

<u>filename</u> consists of a letter followed by up to seven optional numbers or letters.
<u>/ext</u> is an optional name-extension; <u>ext</u> is a sequence of up to three numbers or letters.
<u>.password</u> is an optional password; <u>password</u> is a sequence of up to eight numbers or letters.
<u>:d</u> is an optional drive specification; <u>d</u> is one of the digits Ø through 7.
<u>(diskette name)</u> is an optional field of up to eight numbers or letters. If this field is included, it must be preceded by a drive specification.

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 <u>filename</u> which follow 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(GRAPHICS) <ENTER>
```

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

```
          SYSTEM"GSAVE PROGRAM" <ENTER>
```

or

```
          1ØØ SYSTEM "GSAVE PROGRAM"
```

VDOGRPH

Transfer Text Screen to Graphics Memory

```
VDOGRPH
```

VDOGRPH transfers the contents of the Text Screen (Video Display) to graphics memory. Before you can save a graphics display where text characters are an integral part of your graphics picture, VDOGRPH should be used. Use VDOGRPH in the last line of your program and, as you run the program, the Video Display will be transferred.

If you do not make the video-to-graphics transfer before you save the graphics memory, the file will contain the Graphics Screen contents only and not the Text Screen contents. As a result, for example, a bar graph which does not have the graph's numeric calibrations would be saved.

Examples

When TRSDOS READY is displayed, type:

```
VDOGRPH
```

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

```
SYSTEM"VDOGRPH" <ENTER>
```

or

```
1ØØ SYSTEM"VDOGRPH"
```

For a complete example of using VDOGRPH, see Appendix D, Sample Sessions.

Graphic Utilities Source Code Listings

```
; GCLS -- Clear graphics screen
;
        PSECT    ØFØØØH
GCLS    PUSH     HL           ;Save registers
        PUSH     DE
        PUSH     BC
        LD       A,INCY       ;Set graphics status:
        OUT      (STATUS),A   ; Graphics off, waits off, inc Y
        XOR      A
        OUT      (X),A        ;Set X & Y address to Ø
        OUT      (Y),A
        LD       B,8Ø         ;8Ø X addresses
OUTER   LD       C,B
        LD       B,239        ;239 Y addresses. 24Øth done after loop.
INNER   OUT      (WRITE),A    ;Zero graphics memory
        DJNZ     INNER        ;Go clear next Y
        LD       A,INCXY      ;Set status to inc X & Y after write
        OUT      (STATUS),A
        XOR      A
        OUT      (WRITE),A    ;and clear last (24Øth) Y address
        OUT      (Y),A        ;Set Y back to zero
        LD       A,INCY       ;Reset status to inc Y only
        OUT      (STATUS),A
        XOR      A
        LD       B,C
        DJNZ     OUTER        ;Go clear next X
        LD       A,ØFFH       ;Set status to graphics, waits, no incs.
        OUT      (STATUS),A
        POP      BC           ;Restore registers
        POP      DE
        POP      HL
        XOR      A
        RET                   ;All done.  Go back to caller.
INCY    EQU      7ØH
INCXY   EQU      3ØH
X       EQU      8ØH
Y       EQU      81H
WRITE   EQU      82H
STATUS  EQU      83H
        END      GCLS
```

```
ØØ1 ; GRON --  Turn on graphics display with waits on
ØØ2 ;
ØØ3            PSECT     ØFØØØH
ØØ4 GRON       LD        A,ØFFH
ØØ5            OUT       (STATUS),A
ØØ6            XOR       A
ØØ7            RET
ØØ8 STATUS     EQU       83H
ØØ9            END       GRON
```

```
ØØ1 ; GROFF --  Turn graphics display off with waits off
ØØ2 ;
ØØ3            PSECT     ØFØØØH
ØØ4 GROFF      LD        A,ØFCH
ØØ5            OUT       (STATUS),A
ØØ6            XOR       A
ØØ7            RET
ØØ8 STATUS     EQU       83H
ØØ9            END       GROFF
```

```
; VDOGRPH --  Convert video text screen to graphics
;
        PSECT   ØFØØØH
VDOGRPH PUSH    HL          ;Save registers
        PUSH    DE
        PUSH    BC
        XOR     A
        OUT     (8ØH),A     ;Init X and Y contents in graphics board
        OUT     (81H),A
        LD      A,73H       ;Status = inc Y after write
        OUT     (83H),A
        LD      BC,ØØH      ;Init BC for X and Y contents of vdo
        LD      HL,CHAR
        LD      D,1
        LD      A,1Ø
        RST     8           ;Home cursor to Ø,Ø
LOOP    LD      HL,CHAR     ;Read a vdo character into buffer area
        LD      D,Ø1
        PUSH    BC
        LD      A,11
        RST     8
        LD      A,(CHAR)
        CP      2ØH         ;Check for a blank on vdo screen
        CALL    NZ,CONV     ;If not blank then convert to graphics
        POP     BC
        INC     C           ;Next character.  Add 1 to X value
        LD      A,C
        LD      (X),A
        CP      8Ø          ;End of row?
        JP      NZ,LOOP
        XOR     A
        LD      C,A         ;Reset X to zero
        LD      (X),A
        INC     B           ; and inc. Y screen address
        LD      A,24
        CP      B           ;End of screen?
        JR      Z,EXIT
        LD      HL,Y        ;Inc. Y graphics location.
        LD      A,1Ø
        ADD     A,(HL)
        LD      (HL),A
        JP      LOOP
;
; End of screen.
EXIT    LD      A,ØFFH      ;Set status = graphics, waits, no incs.
        OUT     (83H),A
        LD      B,18H
        LD      A,8
        RST     8           ;Clear vdo screen
```

```
Ø5Ø         POP      BC
Ø51         POP      DE
Ø52         POP      HL           ;Restore registers
Ø53         XOR      A
Ø54         RET                   ;All done.  Return to caller.
Ø55 ;
Ø56 ;  Convert character to graphics
Ø57 CONV    LD       E,A          ;Save character in E
Ø58         SLA      A            ;Multiply char by 2 dropping sign bit
Ø59         LD       C,A          ;Put in BC  ( = char * 2)
Ø6Ø         LD       B,Ø
Ø61         LD       HL,BC        ; and HL
Ø62         SLA      L
Ø63         RL       H
Ø64         SLA      L
Ø65         RL       H            ;HL = BC*4 = char*2 * 4 = char*8
Ø66         ADD      HL,BC        ;HL = HL + BC = char*8 + char*2 = char*1Ø
Ø67         LD       BC,TBL
Ø68         ADD      HL,BC        ;HL = Character table + offset
Ø69         LD       A,(Y)
Ø7Ø         OUT      (81H),A
Ø71         LD       A,(X)
Ø72         OUT      (8ØH),A      ;Set X & Y on graphics board
Ø73         LD       B,1Ø         ;1Ø rows per character
Ø74 CLOOP   IN       A,(82H)      ;Get graphics board contents
Ø75         LD       D,A          ; and save in D
Ø76         LD       A,E
Ø77         AND      8ØH          ;Reverse video?
Ø78         LD       A,(HL)
Ø79         JR       Z,NRML
Ø8Ø         CPL
Ø81 NRML    XOR      D            ;Graphics = graphics XOR character bits
Ø82         OUT      (82H),A      ;Send to graphics board
Ø83         INC      HL           ;Move to next table byte
Ø84         DJNZ     CLOOP
Ø85         RET
Ø86 ;
Ø87 CHAR    DEFB     ØFBH         ;Char buffer. Init value homes cursor
Ø88 X       DEFB     ØØ
Ø89 Y       DEFB     ØØ
Ø9Ø ;
Ø91 ;  CHARACTER GEN TABLE ======================
Ø92         RADIX    1ØH     ;All numbers base 16 (hex)
Ø93 ;
Ø94 TBL     DEFB     ØØ,ØØ,ØØ,ØØ,3F,3F,3C,3C,3C,3C           ;ØØ
Ø95         DEFB     ØØ,ØØ,ØØ,ØØ,ØFC,ØFC,3C,3C,3C,3C         ;Ø1
Ø96         DEFB     3C,3C,3C,3C,ØFC,ØFC,ØØ,ØØ,ØØ,ØØ         ;Ø2
Ø97         DEFB     3C,3C,3C,3C,3F,3F,ØØ,ØØ,ØØ,ØØ           ;Ø3
Ø98         DEFB     ØØ,ØØ,ØØ,ØØ,ØFF,ØFF,3C,3C,3C,3C         ;Ø4
```

```
Ø99          DEFB      3C,3C,3C,3C,ØFC,ØFC,3C,3C,3C,3C               ;Ø5
1ØØ          DEFB      3C,3C,3C,3C,ØFF,ØFF,ØØ,ØØ,ØØ,ØØ               ;Ø6
1Ø1          DEFB      3C,3C,3C,3C,3F,3F,3C,3C,3C,3C                 ;Ø7
1Ø2          DEFB      ØØ,ØØ,ØØ,ØØ,ØFF,ØFF,18,18,18,18               ;Ø8
1Ø3          DEFB      3C,3C,3C,3C,ØFC,3C,3C,3C,3C,3C                ;Ø9
1Ø4          DEFB      18,18,18,18,ØFF,ØFF,ØØ,ØØ,ØØ,ØØ               ;ØA
1Ø5          DEFB      3C,3C,3C,3C,3F,3C,3C,3C,3C,3C                 ;ØB
1Ø6          DEFB      3C,3C,3C,3C,ØFF,ØFF,3C,3C,3C,3C               ;ØC
1Ø7          DEFB      3C,3C,3C,3C,ØFF,3C,3C,3C,3C,3C                ;ØD
1Ø8          DEFB      18,18,18,18,ØFF,ØFF,18,18,18,18               ;ØE
1Ø9          DEFB      18,18,18,18,ØFF,18,18,18,18,18                ;ØF
11Ø          DEFB      ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,3C,3C                 ;1Ø
111          DEFB      ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,3C,3C,3C,3C                 ;11
112          DEFB      ØØ,ØØ,ØØ,ØØ,3C,3C,3C,3C,3C,3C                 ;12
113          DEFB      ØØ,ØØ,3C,3C,3C,3C,3C,3C,3C,3C                 ;13
114          DEFB      3C,3C,3C,3C,3C,3C,3C,3C,3C,3C                 ;14
115          DEFB      18,18,18,18,18,18,18,18,18,18                 ;15
116          DEFB      ØØ,ØØ,ØØ,ØØ,ØFF,ØFF,ØØ,ØØ,ØØ,ØØ               ;16
117          DEFB      ØØ,ØØ,ØØ,ØØ,ØFF,ØØ,ØØ,ØØ,ØØ,ØØ                ;17
118          DEFB      ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØFF,ØFF               ;18
119          DEFB      ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØFF,ØFF,ØFF,ØFF             ;19
12Ø          DEFB      ØØ,ØØ,ØØ,ØØ,ØFF,ØFF,ØFF,ØFF,ØFF,ØFF           ;1A
121          DEFB      ØØ,ØØ,ØFF,ØFF,ØFF,ØFF,ØFF,ØFF,ØFF,ØFF         ;1B
122          DEFB      ØCØ,ØCØ,ØCØ,ØCØ,ØCØ,ØCØ,ØCØ,ØCØ,ØCØ,ØCØ       ;1C
123          DEFB      ØFØ,ØFØ,ØFØ,ØFØ,ØFØ,ØFØ,ØFØ,ØFØ,ØFØ,ØFØ       ;1D
124          DEFB      ØFC,ØFC,ØFC,ØFC,ØFC,ØFC,ØFC,ØFC,ØFC,ØFC       ;1E
125          DEFB      ØØ,Ø8,1C,2A,Ø8,Ø8,Ø8,Ø8,ØØ,ØØ                 ;1F
126  ; End of graphics characters ================================
127          DEFB      ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ                 ;2Ø (space)
128          DEFB      ØØ,Ø8,Ø8,Ø8,Ø8,Ø8,ØØ,Ø8,ØØ,ØØ                 ;21 !
129          DEFB      ØØ,24,24,24,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ                 ;22 "
13Ø          DEFB      ØØ,24,24,7E,24,7E,24,24,ØØ,ØØ                 ;23 #
131          DEFB      ØØ,Ø8,1E,28,1C,ØA,3C,Ø8,ØØ,ØØ                 ;24 $
132          DEFB      ØØ,ØØ,62,64,Ø8,1Ø,26,46,ØØ,ØØ                 ;25 %
133          DEFB      ØØ,3Ø,48,48,3Ø,4A,44,3A,ØØ,ØØ                 ;26 &
134          DEFB      ØØ,Ø4,Ø8,1Ø,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ                 ;27 '
135          DEFB      ØØ,Ø4,Ø8,1Ø,1Ø,1Ø,Ø8,Ø4,ØØ,ØØ                 ;28 (
136          DEFB      ØØ,2Ø,1Ø,Ø8,Ø8,Ø8,1Ø,2Ø,ØØ,ØØ                 ;29 )
137          DEFB      ØØ,Ø8,2A,1C,3E,1C,2A,Ø8,ØØ,ØØ                 ;2A *
138          DEFB      ØØ,ØØ,Ø8,Ø8,3E,Ø8,Ø8,ØØ,ØØ,ØØ                 ;2B +
139          DEFB      ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,Ø8,Ø8,1Ø,ØØ                 ;2C ,
14Ø          DEFB      ØØ,ØØ,ØØ,ØØ,7E,ØØ,ØØ,ØØ,ØØ,ØØ                 ;2D -
141          DEFB      ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,Ø8,ØØ,ØØ                 ;2E .
142          DEFB      ØØ,ØØ,Ø2,Ø4,Ø8,1Ø,2Ø,4Ø,ØØ,ØØ                 ;2F /
143          DEFB      ØØ,3C,42,46,5A,62,42,3C,ØØ,ØØ                 ;3Ø Ø
144          DEFB      ØØ,Ø8,18,28,Ø8,Ø8,Ø8,3E,ØØ,ØØ                 ;31 1
145          DEFB      ØØ,3C,42,Ø2,ØC,3Ø,4Ø,7E,ØØ,ØØ                 ;32 2
146          DEFB      ØØ,3C,42,Ø2,1C,Ø2,42,3C,ØØ,ØØ                 ;33 3
147          DEFB      ØØ,Ø4,ØC,14,24,7E,Ø4,Ø4,ØØ,ØØ                 ;34 4
```

```
148       DEFB     ØØ,7E,4Ø,78,Ø4,Ø2,44,38,ØØ,ØØ              ;35  5
149       DEFB     ØØ,1C,2Ø,4Ø,7C,42,42,3C,ØØ,ØØ              ;36  6
15Ø       DEFB     ØØ,7E,42,Ø4,Ø8,1Ø,1Ø,1Ø,ØØ,ØØ              ;37  7
151       DEFB     ØØ,3C,42,42,3C,42,42,3C,ØØ,ØØ              ;38  8
152       DEFB     ØØ,3C,42,42,3E,Ø2,Ø4,38,ØØ,ØØ              ;39  9
153       DEFB     ØØ,ØØ,ØØ,Ø8,ØØ,ØØ,Ø8,ØØ,ØØ,ØØ              ;3A  :
154       DEFB     ØØ,ØØ,ØØ,Ø8,ØØ,ØØ,Ø8,Ø8,1Ø,ØØ              ;3B  ;
155       DEFB     ØØ,Ø6,ØC,18,3Ø,18,ØC,Ø6,ØØ,ØØ              ;3C  <
156       DEFB     ØØ,ØØ,ØØ,7E,ØØ,7E,ØØ,ØØ,ØØ,ØØ              ;3D  =
157       DEFB     ØØ,6Ø,3Ø,18,ØC,18,3Ø,6Ø,ØØ,ØØ              ;3E  >
158       DEFB     ØØ,3C,42,Ø2,ØC,1Ø,ØØ,1Ø,ØØ,ØØ              ;3F  ?
159       DEFB     ØØ,1C,22,4A,56,4C,2Ø,1E,ØØ,ØØ              ;4Ø  @
16Ø       DEFB     ØØ,18,24,42,7E,42,42,42,ØØ,ØØ              ;41  A
161       DEFB     ØØ,7C,22,22,3C,22,22,7C,ØØ,ØØ              ;42  B
162       DEFB     ØØ,1C,22,4Ø,4Ø,4Ø,22,1C,ØØ,ØØ              ;43  C
163       DEFB     ØØ,78,24,22,22,22,24,78,ØØ,ØØ              ;44  D
164       DEFB     ØØ,7E,4Ø,4Ø,78,4Ø,4Ø,7E,ØØ,ØØ              ;45  E
165       DEFB     ØØ,7E,4Ø,4Ø,78,4Ø,4Ø,4Ø,ØØ,ØØ              ;46  F
166       DEFB     ØØ,1C,22,4Ø,4E,42,22,1C,ØØ,ØØ              ;47  G
167       DEFB     ØØ,42,42,42,7E,42,42,42,ØØ,ØØ              ;48  H
168       DEFB     ØØ,1C,Ø8,Ø8,Ø8,Ø8,Ø8,1C,ØØ,ØØ              ;49  I
169       DEFB     ØØ,ØE,Ø4,Ø4,Ø4,Ø4,44,38,ØØ,ØØ              ;4A  J
17Ø       DEFB     ØØ,42,44,48,7Ø,48,44,42,ØØ,ØØ              ;4B  K
171       DEFB     ØØ,4Ø,4Ø,4Ø,4Ø,4Ø,4Ø,7E,ØØ,ØØ              ;4C  L
172       DEFB     ØØ,42,66,5A,5A,42,42,42,ØØ,ØØ              ;4D  M
173       DEFB     ØØ,42,62,52,4A,46,42,42,ØØ,ØØ              ;4E  N
174       DEFB     ØØ,3C,42,42,42,42,42,3C,ØØ,ØØ              ;4F  O
175       DEFB     ØØ,7C,42,42,7C,4Ø,4Ø,4Ø,ØØ,ØØ              ;5Ø  P
176       DEFB     ØØ,3C,42,42,42,4A,44,3A,ØØ,ØØ              ;51  Q
177       DEFB     ØØ,7C,42,42,7C,48,44,42,ØØ,ØØ              ;52  R
178       DEFB     ØØ,3C,42,4Ø,3C,Ø2,42,3C,ØØ,ØØ              ;53  S
179       DEFB     ØØ,3E,Ø8,Ø8,Ø8,Ø8,Ø8,Ø8,ØØ,ØØ              ;54  T
18Ø       DEFB     ØØ,42,42,42,42,42,42,3C,ØØ,ØØ              ;55  U
181       DEFB     ØØ,42,42,42,24,24,18,18,ØØ,ØØ              ;56  V
182       DEFB     ØØ,42,42,42,5A,5A,66,42,ØØ,ØØ              ;57  W
183       DEFB     ØØ,42,42,24,18,24,42,42,ØØ,ØØ              ;58  X
184       DEFB     ØØ,22,22,22,1C,Ø8,Ø8,Ø8,ØØ,ØØ              ;59  Y
185       DEFB     ØØ,7E,Ø2,Ø4,18,2Ø,4Ø,7E,ØØ,ØØ              ;5A  Z
186       DEFB     ØØ,3C,2Ø,2Ø,2Ø,2Ø,2Ø,3C,ØØ,ØØ              ;5B  [
187       DEFB     ØØ,ØØ,4Ø,2Ø,1Ø,Ø8,Ø4,Ø2,ØØ,ØØ              ;5C  \
188       DEFB     ØØ,3C,Ø4,Ø4,Ø4,Ø4,Ø4,3C,ØØ,ØØ              ;5D  ]
189       DEFB     ØØ,Ø8,14,22,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ              ;5E  ^
19Ø       DEFB     ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØFF,ØØ             ;5F  _
191       DEFB     ØØ,1Ø,Ø8,Ø4,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ              ;6Ø
192       DEFB     ØØ,ØØ,ØØ,38,Ø4,3C,44,3A,ØØ,ØØ              ;61  a
193       DEFB     ØØ,4Ø,4Ø,5C,62,42,62,5C,ØØ,ØØ              ;62  b
194       DEFB     ØØ,ØØ,ØØ,3C,42,4Ø,42,3C,ØØ,ØØ              ;63  c
195       DEFB     ØØ,Ø2,Ø2,3A,46,42,46,3A,ØØ,ØØ              ;64  d
196       DEFB     ØØ,ØØ,ØØ,3C,42,7E,4Ø,3C,ØØ,ØØ              ;65  e
```

```
197          DEFB     ØØ,ØC,12,1Ø,7C,1Ø,1Ø,1Ø,ØØ,ØØ              ;66 f
198          DEFB     ØØ,ØØ,ØØ,3A,46,46,3A,Ø2,3C,ØØ              ;67 g
199          DEFB     ØØ,4Ø,4Ø,5C,62,42,42,42,ØØ,ØØ              ;68 h
2ØØ          DEFB     ØØ,Ø8,ØØ,18,Ø8,Ø8,Ø8,1C,ØØ,ØØ              ;69 i
2Ø1          DEFB     ØØ,Ø4,ØØ,ØC,Ø4,Ø4,Ø4,44,38,ØØ              ;6A j
2Ø2          DEFB     ØØ,4Ø,4Ø,44,48,5Ø,68,44,ØØ,ØØ              ;6B k
2Ø3          DEFB     ØØ,18,Ø8,Ø8,Ø8,Ø8,Ø8,1C,ØØ,ØØ              ;6C l
2Ø4          DEFB     ØØ,ØØ,ØØ,76,49,49,49,49,ØØ,ØØ              ;6D m
2Ø5          DEFB     ØØ,ØØ,ØØ,5C,62,42,42,42,ØØ,ØØ              ;6E n
2Ø6          DEFB     ØØ,ØØ,ØØ,3C,42,42,42,3C,ØØ,ØØ              ;6F o
2Ø7          DEFB     ØØ,ØØ,ØØ,5C,62,62,5C,4Ø,4Ø,ØØ              ;7Ø p
2Ø8          DEFB     ØØ,ØØ,ØØ,3A,46,46,3A,Ø2,Ø2,ØØ              ;71 q
2Ø9          DEFB     ØØ,ØØ,ØØ,5C,62,4Ø,4Ø,4Ø,ØØ,ØØ              ;72 r
21Ø          DEFB     ØØ,ØØ,ØØ,3E,4Ø,3C,Ø2,7C,ØØ,ØØ              ;73 s
211          DEFB     ØØ,1Ø,1Ø,7C,1Ø,1Ø,12,ØC,ØØ,ØØ              ;74 t
212          DEFB     ØØ,ØØ,ØØ,42,42,42,46,3A,ØØ,ØØ              ;75 u
213          DEFB     ØØ,ØØ,ØØ,42,42,42,24,18,ØØ,ØØ              ;76 v
214          DEFB     ØØ,ØØ,ØØ,41,49,49,49,36,ØØ,ØØ              ;77 w
215          DEFB     ØØ,ØØ,ØØ,42,24,18,24,42,ØØ,ØØ              ;78 x
216          DEFB     ØØ,ØØ,ØØ,42,42,46,3A,Ø2,3C,ØØ              ;79 y
217          DEFB     ØØ,ØØ,ØØ,7E,Ø4,18,2Ø,7E,ØØ,ØØ              ;7A z
218          DEFB     ØØ,ØC,1Ø,1Ø,2Ø,1Ø,1Ø,ØC,ØØ,ØØ              ;7B {
219          DEFB     ØØ,Ø8,Ø8,Ø8,ØØ,Ø8,Ø8,Ø8,ØØ,ØØ              ;7C |
22Ø          DEFB     ØØ,3Ø,Ø8,Ø8,Ø4,Ø8,Ø8,3Ø,ØØ,ØØ              ;7D }
221          DEFB     ØØ,3Ø,49,Ø6,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ              ;7E ~
222          DEFB     ØØ,Ø8,Ø8,3E,Ø8,Ø8,ØØ,3E,ØØ,ØØ              ;7F + and _
223 ;
224          END      VDOGRPH
```

```
ØØ1 ; GSAVE --  Save graphics display to disk
ØØ2 ;
ØØ3         PSECT    ØFØØØH
ØØ4 GSAVE   PUSH     HL            ;Save registers
ØØ5         PUSH     DE
ØØ6         PUSH     BC
ØØ7         PUSH     HL
ØØ8         CALL     NOBRK
ØØ9         LD       (PBRK),HL     ;Save address of previous break routine
Ø1Ø         LD       HL,DCBEE      ;Zero DCB buffer
Ø11         LD       DE,DCBEE+1
Ø12         LD       BC,59
Ø13         LD       (HL),ØØH
Ø14         LDIR
Ø15         POP      HL
Ø16         LD       C,(HL)        ;Get command length
Ø17         LD       B,Ø
Ø18         INC      HL
Ø19         LD       A,' '
Ø2Ø         CPIR
Ø21         JP       NZ,ERROR      ;Error if blank not found
Ø22         LD       DE,DCBEE
Ø23         LDIR                   ;DCBEE now has filespec...
Ø24         LD       HL,PARM
Ø25         LD       DE,DCBEE
Ø26         LD       A,4Ø
Ø27         RST      8             ;Open file
Ø28         JP       NZ,BOMB
Ø29         XOR      A
Ø3Ø         LD       (OPNFLG),A    ;Set flag: file is open
Ø31 ;
Ø32         LD       HL,BRKHIT     ;set up break handling routine
Ø33         LD       A,3
Ø34         RST      8
Ø35         LD       A,ØE3H        ;status = inc X after read
Ø36         OUT      (STATUS),A
Ø37         XOR      A
Ø38         OUT      (X),A         ;init X & Y to zero
Ø39         OUT      (Y),A
Ø4Ø         LD       E,A           ;counter for X values
Ø41         LD       D,8Ø          ;8Ø X values
Ø42         LD       B,75          ;75 disk records for entire screen
Ø43 NXTREC  LD       HL,BUFFER
Ø44         LD       C,B
Ø45         LD       B,Ø           ;256 bytes per record
Ø46 NGRPH   IN       A,(GRAPH)     ;Get next graphics byte
Ø47         LD       (HL),A        ; and put in buffer
Ø48         INC      HL
Ø49         INC      E
```

```
Ø5Ø        LD      A,E
Ø51        CP      D
Ø52        JR      NZ,EGRPH       ;Same row?
Ø53        XOR     A
Ø54        LD      E,A
Ø55        OUT     (X),A          ;Next row. Set X to zero
Ø56        LD      A,(YPOS)
Ø57        INC     A
Ø58        JP      Z,DOBRK        ;Stop & kill file if break hit
Ø59        LD      (YPOS),A
Ø6Ø        OUT     (Y),A
Ø61 EGRPH  DJNZ    NGRPH          ;Go get next graphics byte
Ø62        PUSH    DE
Ø63        LD      DE,DCBEE
Ø64        LD      A,43
Ø65        RST     8              ;Write disk record
Ø66        POP     DE
Ø67        JR      NZ,BOMB
Ø68        LD      B,C
Ø69        DJNZ    NXTREC         ;Go fill buffer for next record
Ø7Ø ;
Ø71 EXIT   CALL    CLOSE
Ø72        LD      A,ØFFH         ;Status = graphics, waits, no incs
Ø73        OUT     (STATUS),A
Ø74        CALL    NOBRK
Ø75        LD      HL,(PBRK)          ;Restore previous break routine
Ø76        LD      A,3
Ø77        RST     8
Ø78        POP     BC
Ø79        POP     DE
Ø8Ø        POP     HL
Ø81        LD      A,(EFLAG)
Ø82        CP      Ø
Ø83        RET                    ;All done.  Return to caller.
Ø84 ;
Ø85 ; Subroutines
Ø86 ;
Ø87 CLOSE  LD      A,(OPNFLG)
Ø88        OR      A
Ø89        RET     NZ             ;Return if file not open
Ø9Ø        LD      DE,DCBEE
Ø91        LD      A,42
Ø92        RST     8
Ø93        LD      A,1
Ø94        LD      (OPNFLG),A     ;Set flag: file is closed.
Ø95        RET
Ø96 ;
Ø97 NOBRK  LD      HL,Ø
Ø98        LD      A,3
```

```
Ø99          RST      8            ;Inhibit break
1ØØ          RET
1Ø1 ;
1Ø2 BRKHIT   PUSH     AF
1Ø3          LD       A,ØFFH       ;Signal break has been hit
1Ø4          LD       (YPOS),A     ;By making next Y be zero
1Ø5          POP      AF
1Ø6          RET
1Ø7 ;
1Ø8 ; Error and break exits
1Ø9 ;
11Ø DOBRK    CALL     CLOSE        ;Process break.
111          LD       DE,DCBEE
112          LD       A,41
113          RST      8            ;Kill file
114          LD       HL,BRKMSG
115          LD       B,PBRK-BRKMSG
116          LD       C,ØDH
117          LD       A,9
118          LD       (EFLAG),A
119          RST      8
12Ø          JP       EXIT
121 ;
122 ERROR    LD       A,47         ;Required Command Parameter Not Found
123 ;
124 BOMB     LD       (EFLAG),A
125          LD       B,A
126          LD       A,39
127          RST      8            ;Print "ERROR nn" message
128          JP       EXIT
129 ;
13Ø X        EQU      8ØH
131 Y        EQU      81H
132 GRAPH    EQU      82H
133 STATUS   EQU      83H
134 EFLAG    DEFB     Ø
135 YPOS     DEFB     Ø
136 BRKMSG   DEFM     '** BREAK **.  File killed'
137 PBRK     DEFS     2
138 OPNFLG   DEFB     1
139 PARM     DEFW     BUFFER
14Ø          DEFW     ØØ,ØØ
141          DEFB     'W',Ø,'F',2,Ø          ;Write from graphics to disk
142 DCBEE    DEFS     6Ø
143 BUFFER   DEFS     256
144          END      GSAVE
```

```
; GLOAD --  Save graphics display to disk
;
        PSECT   ØFØØØH
GLOAD   PUSH    HL           ;Save registers
        PUSH    DE
        PUSH    BC
        PUSH    HL
        CALL    NOBRK
        LD      (PBRK),HL    ;Save address of previous break routine
        LD      HL,DCBEE     ;Zero DCB buffer
        LD      DE,DCBEE+1
        LD      BC,59
        LD      (HL),ØØH
        LDIR
        POP     HL
        LD      C,(HL)       ;Get command length
        LD      B,Ø
        INC     HL
        LD      A,' '
        CPIR
        JP      NZ,ERROR     ;Error if blank not found
        LD      DE,DCBEE
        LDIR                 ;DCBEE now has filespec...
        LD      HL,PARM
        LD      DE,DCBEE
        LD      A,4Ø
        RST     8            ;Open file
        JP      NZ,BOMB
        XOR     A
        LD      (OPNFLG),A   ;Set flag: file is open
;
        LD      HL,BRKHIT    ;Set up break handling routine
        LD      A,3
        RST     8
        LD      A,ØB3H       ;status = inc X after write
        OUT     (STATUS),A
        XOR     A
        OUT     (X),A        ;init X & Y to zero
        OUT     (Y),A
        LD      E,A          ;counter for X values
        LD      D,8Ø         ;8Ø X values
        LD      B,75         ;75 disk records for entire screen
NXTREC  PUSH    DE
        LD      DE,DCBEE
        LD      A,34
        RST     8            ;Read record from disk
        POP     DE
        JR      NZ,BOMB
        LD      HL,BUFFER
```

```
Ø5Ø         LD       C,B
Ø51         LD       B,Ø           ;256 bytes per record
Ø52 NGRPH   LD       A,(HL)
Ø53         OUT      (GRAPH),A
Ø54         INC      HL
Ø55         INC      E
Ø56         LD       A,E
Ø57         CP       D
Ø58         JR       NZ,EGRPH      ;Same row?
Ø59         XOR      A
Ø6Ø         LD       E,A
Ø61         OUT      (X),A         ;Next row. Set X to zero
Ø62         LD       A,(YPOS)
Ø63         INC      A
Ø64         JP       Z,DOBRK       ;Stop if break hit
Ø65         LD       (YPOS),A
Ø66         OUT      (Y),A
Ø67 EGRPH   DJNZ     NGRPH         ;Go get next graphics byte
Ø68         LD       B,C
Ø69         DJNZ     NXTREC        ;Go read next disk record
Ø7Ø ;
Ø71 EXIT    CALL     CLOSE
Ø72         LD       A,ØFFH        ;Status = graphics, waits, no incs.
Ø73         OUT      (STATUS),A
Ø74         CALL     NOBRK
Ø75         LD       HL,(PBRK)
Ø76         LD       A,3
Ø77         RST      8             ;Restore previous break routine
Ø78         POP      BC
Ø79         POP      DE
Ø8Ø         POP      HL
Ø81         LD       A,(EFLAG)
Ø82         CP       Ø
Ø83         RET
Ø84 ;
Ø85 ; Subroutines
Ø86 ;
Ø87 CLOSE   LD       A,(OPNFLG)
Ø88         OR       A
Ø89         RET      NZ            ;Return if file not open
Ø9Ø         LD       DE,DCBEE
Ø91         LD       A,42
Ø92         RST      8
Ø93         RET
Ø94 ;
Ø95 NOBRK   LD       HL,Ø
Ø96         LD       A,3
Ø97         RST      8             ;Inhibit break
Ø98         RET
```

```
Ø99 ;
1ØØ BRKHIT  PUSH     AF
1Ø1         LD       A,ØFFH       ;Signal break has been hit
1Ø2         LD       (YPOS),A     ;by making next Y be zero
1Ø3         POP      AF
1Ø4         RET
1Ø5 ;
1Ø6 ; Error and break exits
1Ø7 ;
1Ø8 DOBRK   LD       A,ØFFH       ;Process break
1Ø9         LD       (EFLAG),A
11Ø         JP       EXIT         ;Return with error code set
111 ;
112 ERROR   LD       A,47         ;Required Command Parameter Not Found
113 ;
114 BOMB    LD       (EFLAG),A
115         LD       B,A
116         LD       A,39
117         RST      8            ;Print "ERROR nn" message
118         JP       EXIT
119 ;
12Ø X       EQU      8ØH
121 Y       EQU      81H
122 GRAPH   EQU      82H
123 STATUS  EQU      83H
124 EFLAG   DEFB     Ø
125 YPOS    DEFB     Ø
126 PBRK    DEFS     2
127 OPNFLG  DEFB     1
128 PARM    DEFW     BUFFER
129         DEFW     ØØ,ØØ
13Ø         DEFB     'R',Ø,'F',Ø,Ø          ;Read from disk to graphics
131 DCBEE   DEFS     6Ø
132 BUFFER  DEFS     256
133         END      GLOAD
```

```
; GPRINT --  Print graphics screen to graphics printer
;
        PSECT    ØFØØØH
GPRINT  PUSH     HL            ;Save registers
        PUSH     DE
        PUSH     BC
        PUSH     IX
        OR       ØDBH          ;Output a Control byte to cause
        OUT      (STATUS),A    ; Y to automatically dec. on a read
        CALL     INITBF
;
        XOR      A             ;Set A to Ø
        OUT      (X),A         ;Initialize the X position
        LD       (BPOS),A      ;   "       "  bit position
        LD       (XLOC),A      ;   "       "   "  location counter
        LD       HL,BGMODE
        LD       B,1
        LD       C,ØDH
        LD       A,19
        RST      8             ;Begin graphics print mode
;
LOOP1   LD       IX,BUFFER     ;point IX at the printer buffer
        LD       B,24Ø         ;go through a whole column of bytes
        LD       A,B           ;Put value in A and decrement
        DEC      A             ;  so it can be put out as
        OUT      (Y),A         ;    the Y position
COLUMN  LD       HL,MASK       ;point HL at the mask byte
        IN       A,(GRAPH)     ;input a graphics byte
        AND      (HL)          ;chop off all but proper bit
        CALL     PO,SETØ       ;if result is odd parity set bit Ø
                               ; otherwise bit A is Ø
        LD       HL,BPOS       ;point HL at the bit position
        PUSH     BC            ;save register B (for DJNZ loop)
        LD       B,(HL)        ;get count
        INC      B             ;increment (in case it is Ø)
DECJ    DEC      B             ;move bit left BPOS number of times
        JR       Z,PAST        ;if done, move on...
        RLC      A             ;move bit left one position
        JR       DECJ          ;repeat loop
PAST    POP      BC            ;get loop counter back
        OR       (IX)          ;merge A with byte of printer buffer
        LD       (IX),A        ;put merged result in buffer
        INC      IX            ;increment buffer pointer
        DJNZ     COLUMN        ;continue loop
-----------------------------------------------------------------------
        LD       A,7           ;See if BPOS has gotten to 8.
        INC      (HL)          ;  If it has (printer uses 7 bits)
        CP       (HL)          ;    print the buffer and reset
        CALL     Z,PRNDRS      ;      BPOS to Ø
```

```
Ø5Ø ;
Ø51         LD      HL,MASK        ;After getting a vertical row of bits
Ø52         RRC     (HL)           ; rotate the mask right one position
Ø53         LD      A,8ØH          ;Check to see if its back to
Ø54         CP      (HL)           ; it's original value, if not
Ø55         JR      NZ,LOOP1       ; go get another row of bits
Ø56         LD      A,(XLOC)       ;If so, get X pos (to increment it)
Ø57         CP      79             ;Check to see if we are at the end...
Ø58         JP      Z,BYE
Ø59         INC     A              ;otherwise increment the X counter
Ø6Ø         LD      (XLOC),A       ;and store it back
Ø61         OUT     (X),A          ;also update the port value
Ø62         JR      LOOP1          ;now go get another row of bits
Ø63 ;------------------------------------------------------------------------
Ø64 SETØ    LD      A,1            ;set A to binary ØØØØ ØØØ1
Ø65         RET                    ; and return
Ø66 ;
Ø67 PRNDRS  LD      HL,BUFFER      ;Set up the
Ø68         LD      B,24Ø          ;  PRLINE SVC and
Ø69         LD      C,ØDH          ;    send the buffer
Ø7Ø         LD      A,19
Ø71         RST     8
Ø72         JP      NZ,ERROR
Ø73         XOR     A              ;clear A
Ø74         LD      (BPOS),A       ;reset bit position counter
Ø75 ;
Ø76 INITBF  LD      HL,BUFFER      ;Initialize the printer buffer
Ø77         LD      DE,BUFFER+1    ; with all 8ØH
Ø78         LD      BC,239
Ø79         LD      A,8ØH
Ø8Ø         LD      (HL),A
Ø81         LDIR
Ø82         RET
Ø83 ;----------------------------------------------------------------------
Ø84 ERROR   LD      B,A            ;Error routine
Ø85         LD      A,39
Ø86         RST     8
Ø87         RST     Ø
Ø88 ;----------------------------------------------------------------------
Ø89 BYE     CALL    PRNDRS
Ø9Ø         LD      HL,EGMODE
Ø91         LD      B,1
Ø92         LD      C,ØDH
Ø93         LD      A,19
Ø94         RST     8              ;End graphics print mode
Ø95         POP     IX             ;Restore registers
Ø96         POP     BC
Ø97         POP     DE
Ø98         POP     HL
```

```
Ø99           XOR       A
1ØØ           RET
1Ø1 X         EQU       8ØH
1Ø2 Y         EQU       81H
1Ø3 GRAPH     EQU       82H
1Ø4 STATUS    EQU       83H
1Ø5 MASK      DEFB      8ØH               ;Mask to use in extracting bits
1Ø6 BGMODE    DEFB      12H               ;Control byte: start graphics mode
1Ø7 BUFFER    DEFS      24Ø               ;Printer data buffer
1Ø8 EGMODE    DEFB      1EH               ;Control byte: end graphics mode
1Ø9 BPOS      DEFB      Ø                 ;Bit position in printer buffer
11Ø XLOC      DEFB      Ø                 ;Current X location value
111 ;
112           END       GPRINT
```

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 II FORTRAN (26-47Ø1). 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 3 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 2 corresponding library subroutines: PAINT and PAINTT. PAINT is for painting solid black or white, and PAINTT is for using 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 and *GRPHSAM,GRPLIB-S,FORLIB-S,-U is the program name. *-E sends the routine.

Note: If there are unresolved external references, then 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.

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

Table 7

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

Table 8

CIRCLE
Draws a Circle, Arc, Semi-Circle, Point or Ellipse

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

radius is 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 REAL type and specifies the startpoint of the circle.
end is REAL type and specifies the endpoint of the circle.
ar is the aspect ratio, is 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, semi-circles, 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,CLGRPH,OPTION
ØØØ3Ø     COLOR=1
ØØØ4Ø     CLGRPH=1
ØØØ5Ø     OPTION=Ø
ØØØ6Ø     CALL GRPINI(OPTION)
ØØØ7Ø     CALL CLS(CLGRPH)
ØØØ8Ø     CALL SETXY(3ØØ,1ØØ)
ØØØ9Ø     CALL CIRCLE(1ØØ,COLOR,Ø.Ø,Ø.Ø,Ø.Ø)
ØØ1ØØ     CALL PAINT(COLOR,COLOR)
ØØ11Ø     END
```

CLS
Clears Screen(s)

CLS (n)

n is of LOGICAL type, clears the Screen(s) and is a integer expression between Ø and 2:

Ø = clears only Text Screen
1 = clears only Graphics Screen
2 = clears both Text and Graphics

CLS clears Screen(s) according to the specified variable.
Note: Any value greater than 2 gives you an error.

Example

```
CALL CLS(2)
```

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 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 highest 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 #2 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 = (xl,yl) lower left corner = endpoint = (x2,y2)

Size (in pixels):	width = x2-xl+1
	length = y2-yl+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(125),ACTION
ØØ15Ø     ACTION=1
ØØ2ØØ     CALL GRPINI(Ø)
ØØ3ØØ     CALL CLS(2)
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,125)
ØØ9ØØ     DO 1Ø I=1,5ØØØ
Ø1ØØØ 1Ø  CONTINUE
Ø1Ø5Ø C   CLEAR SCREEN AND RESTORE GRPH FROM ARRAY
Ø11ØØ     CALL CLS(1)
Ø12ØØ     CALL SETXY(11Ø,11Ø)
Ø13ØØ     CALL PUT(V,ACTION)
Ø14ØØ     DO 2Ø I=1,5ØØØ
Ø15ØØ 2Ø  CONTINUE
Ø16ØØ     END
```

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 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(2)
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 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)

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 by 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(2)
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(2)
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(2)
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 LOGICAL type and specifies how the data is to be written to the video. Action may be one of the following:

1 = OR
2 = AND
3 = PRESET
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
Sets Screen

SCREEN (switch)

switch is of LOGICAL type and specifies the type of Screen display and may be one of the following:

Ø = Graphics ON/ normal speed
1 = Graphics OFF/normal speed
2 = Graphics ON/ high speed
3 = Graphics OFF/high speed

SCREEN lets you set the proper Screen and screen speed. SCREEN 2 and 3 display graphics more rapidly on your Screen than SCREEN Ø and 1. Any value greater than 3 with SCREEN gives you a error.

Example

```
        CALL SCREEN(2)
```

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(2)
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=2
ØØ14Ø           ALL 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(2)
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 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 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(2)
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=1,12
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 left x coordinate of viewport
    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/ Assembly Language

The Graphics Subroutine Library (GRPLIB/REL) included on the Graphics Diskette can be linked to any program to access the Graphics Subroutines. The FORTRAN Assembly Subroutine Library (FORLIB/REL) must also be linked (using the L8Ø Linker) to any program that will access the Graphics Subroutines.

Note: To use the Computer Graphics package with Assembly language, you'll need the Editor Assembler (26-47Ø2).

Before any calls are made to the Graphics Subroutines, the FORTRAN Subroutine Library must be initialized. This can be done by having the following as the first executable statements in your assembly program:

```
                LD      BC,L1
                JP      $INIT       ; FORTRAN INIT ROUTINE
        L1:                         ; YOUR PROGRAM STARTS HERE
```

Note: When you jump to $INIT, the Stack Pointer will be set to the contents of register pair DE.

Additionally, the Graphics Subroutine Library must be initialized. This is done by inserting a call to GRPINI before attempting to access the Graphics Subroutines Library.

Any errors resulting from incorrect use of the Graphics Subroutines will cause a GRAPHICS ERROR, and control will return to TRSDOS READY.

A program that demonstrates how assembly-language can be used to exercise the Graphics library is included in Appendix D.

You must link the FORTRAN subroutine library as well as the Graphics library to the object code of your graphics program in order to produce an executable load module. A description of the various FORTRAN Library Subroutines such as $CA, and the Assembler linkage conventions for them can be found in the Editor Assembler User's Manual.

All of the subroutines described in this section have a corresponding subroutine in FORTRAN. If more information is needed to understand a given routine, see the FORTRAN interface section of this manual. In the examples that follow, the Assembler code will define and describe how the given graphics functions are invoked as well as describe the size and format of the parameters.

Important Note: Free memory (above your program) is utilized by the Graphics Subroutines for temporary storage area. Extreme care should be exercised if your program accesses this memory.

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 3 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 2 corresponding library subroutines: PAINT and PAINTT. PAINT is for painting solid black or white, and PAINTT is for using 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.

Important Note: All graphics routines utilize the AF, BC, DE, and HL register pairs. It is the user's responsiblity to save these registers (if needed) before a call to a graphics routine.

CIRCLE

Draws a circle, arc, or ellipse using the current x and y coordinates as the center.

Example

```
RADIUS: DS      2           ; RADIUS OF CIRCLE
                            ;  INTEGER
COLOR:  DS      1           ; Ø->BLACK, 1->WHITE
START:  DS      4           ; SNGL PRECISION FLOATING
                            ;  POINT. Ø=CENTER OF
                            ;  RIGHT SIDE
END:    DS      4           ; SNGL PRECISION FLOATING
                            ;  POINT. IF IT IS = Ø
                            ;  2*PI IS USED.
RATIO:  DS      4           ; SNGL PRECISION FLOATING
                            ;  POINT. IF IT IS Ø, .5
                            ;  IS USED (CIRCLE).
P3:     DS      6           ; PARAMETERS 3 - 5
        LD      HL,START
        LD      (P3),HL
        LD      HL,END
        LD      (P3+2),HL
        LD      HL,RATIO
        LD      (P3+4),HL
        LD      HL,RADIUS
        LD      DE,COLOR
        LD      BC,P3
        CALL    CIRCLE
```

CLS

Clears the screen according to the specified variable.

Example

```
N:      DS      1           ; Ø->CLEAR ONLY TEXT
                            ; 1->CLEAR ONLY GRAPHICS
                            ; 2->CLEAR BOTH TEXT AND
                            ;    GRAPHICS
        LD      HL,N
        CALL    CLS
```

GET

Reads the contents of a pixel block into memory for future use by PUT.

Example

```
ARRAY:  DS      9ØØ       ; SPACE TO STORE PIXELS
SIZE:   DW      9ØØ       ; SIZE OF STORAGE AREA
        LD      HL,ARRAY
        LD      DE,SIZE
        CALL    GET
```

GRPINI

Graphics initialization routine. This function must be called before any other graphics calls are made.

Example

```
OPTION: DS      1            ; Ø -> CLEAR GRAPHICS
                             ;  SCREEN.
                             ; NOT ZERO -> DO NOT
                             ;  CLEAR GRAPHICS
                             ;  SCREEN.
        LD      HL,OPTION
        CALL    GRPINI
```

LINE

Draws a line between the previous and the current coordinates.

Example

```
        COLOR:  DS      1           ; Ø->BLACK, 1->WHITE
        STYLE:  DS      2           ; ANY 16-BIT PATTERN
                                    ;  (ØFFFFH = SOLID LINE)
                LD      HL,COLOR
                LD      DE,STYLE
                CALL    LINE
```

LINEB

Same as LINE, except LINEB draws a box between the two sets of coordinates.

Example

```
        COLOR:  DS      1           ; Ø->BLACK, 1->WHITE
        STYLE:  DS      2           ; ANY 16-BIT PATTERN
                                    ;  (ØFFFFH = SOLID LINE)
                LD      HL,COLOR
                LD      DE,STYLE
                CALL    LINEB
```

LINEBF

Same as LINEB, except LINEBF fills the box (colors in the box).

Example

```
        COLOR:  DS      1           ; Ø->BLACK, 1->WHITE
                LD      HL,COLOR
                CALL    LINEBF
```

PAINT

Paints your screen in the specified color (black or white).

Example

```
COLOR:  DS      1          ; Ø->BLACK, 1->WHITE
BORDER: DS      1          ; Ø->BLACK, 1->WHITE
        LD      HL,COLOR
        LD      DE,BORDER
        CALL    PAINT
```

PAINTT

This routine allows you to paint with a precise pattern by using a technique called 'tiling'.

Example

```
ARRAYT: DS      1Ø         ; DEFINES PATTERN
BORDER: DS      1          ; Ø->BLACK, 1->WHITE
ARRAYS: DS      2          ; DESCRIBES BACKGROUND OF
                           ;  AREA BEING PAINTED
        LD      HL,ARRAYT
        LD      DE,BORDER
        LD      BC,ARRAYS
        CALL    PAINTT
```

PSET

Sets a pixel either ON or OFF.

Example

```
COLOR:  DS      1          ; Ø->OFF, 1->ON
        LD      HL,COLOR
        CALL    PSET
```

PRESET

Same as PSET.

PUT

The given array (stored by GET) is put on the video screen at the current x and y coordinates set by calling SETXY.

Example

```
        ARRAY:  DS      9ØØ         ; STORAGE FOR PIXELS
        ACTION: DS      1           ; 1->OR, 2->AND,
                                    ; 3->PRESET,4->PSET,5->XOR
                LD      HL,ARRAY
                LD      DE,ACTION
                CALL    PUT
```

SCREEN

Allows you to set the screen mode.

Example

```
        N:      DS      1           ; Ø->GRAPHICS ON/NORMAL
                                    ;    SPEED
                                    ; 1->GRAPHICS OFF/NORMAL
                                    ;    SPEED
                                    ; 2->GRAPHICS ON/HIGH
                                    ;    SPEED
                                    ; 3->GRAPHICS OFF/HIGH
                                    ;    SPEED
                LD      HL,N
                CALL    SCREEN
```

SETXY

Sets both the current and previous x and y coordinates.

Example

```
X:      DS      2       ; X COORDINATE
Y:      DS      2       ; Y COORDINATE
        LD      HL,X
        LD      DE,Y
        CALL    SETXY
```

SETXYR

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

Example

```
X:      DS      2       ; X RELATIVE COORDINATE
Y:      DS      2       ; Y RELATIVE COORDINATE
        LD      HL,X
        LD      DE,Y
        CALL    SETXYR
```

VIEW

Allows you to designate specific areas of your screen where the graphics will be displayed.

Example

```
LEFTX:  DS      2          ; Ø<=LEFTX<=639
LEFTY:  DS      2          ; Ø<=LEFTY<=239
RIGHTX: DS      2          ; Ø<=RIGHTX<=639
RIGHTY: DS      2          ; Ø<=RIGHTY<=239
COLOR:  DS      1          ; Ø->BLACK, 1->WHITE,
                           ; -1 -> DON'T SHADE IT.
BORDER: DS      1          ; Ø->BLACK, 1->WHITE
                           ; -1 -> BORDER NOT DRAWN.
P3:     DS      8          ; PARAMETERS 3 - 6
        LD      HL,RIGHTX
        LD      (P3),HL
        LD      HL,RIGHTY
        LD      (P3+2),HL
        LD      HL,COLOR
        LD      (P3+4),HL
        LD      HL,BORDER
        LD      (P3+6),HL
        LD      HL,LEFTX
        LD      DE,LEFTY
        LD      BC,P3
        CALL    VIEW
```

POINT

Returns the pixel value at the current x and y coordinate.

Example

```
        CALL    POINT      ; PUTS VALUE IN A
```

FVIEW

Returns the specified viewport parameter.

Example

```
N:      DS      1       ; Ø->LEFT X COORDINATE
                        ;  1->LEFT Y COORDINATE
                        ;  2->RIGHT X COORDINATE
                        ;  3->RIGHT Y COORDINATE
        LD      HL,N
        CALL    FVIEW   ; PUTS VALUE IN HL
```

6/ COBOL Interface

The Graphics diskette contains two files for use with COBOL programs:

- CBLGRAPH/CPY -- A Cobol source file containing the definitions for the Cobol parameters to use with the graphics routines.
- CBLGRAPH/CMD -- The graphics subroutine to be called from Cobol programs.

To use Graphics from a COBOL program, the following steps should be taken:

1. In the WORKING-STORAGE SECTION of the COBOL program the following statement should appear:

   ```
   COPY "CBLGRAPH/CPY".
   ```

   This statement should be placed after any 77 level items that may be defined in the program.

2. In the PROCEDURE DIVISION the following statement should appear:

   ```
   CALL GRAPH-SUB USING GRAPHICS-PARAMETERS.
   ```

   This statement gives the Graphics subroutine the address in memory of the parameters to be used by all further Graphics routine calls.

3. The Graphics library and board must be initialized before any other Graphics routines may be done. To initialize the Graphics library and board use the following statement:

   ```
   CALL GRAPH-SUB USING GRPINI-CMD.
   ```

4. Assign values to the required parameters in GRAPHICS-PARAMETERS (using MOVE or COMPUTE) and call the graphics routine using one of the options defined in GRAPHICS-OPTIONS. The options and parameters are described on the following pages.

5. Compile the program as usual. (RSCOBOL).

6. To run the program add the parameter {T=BA3B} to the end of the RUNCOBOL command line.

   Example: RUNCOBOL PROGRAM {T=BA3B}

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 3 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 2 corresponding library subroutines: PAINT and PAINTT. PAINT is for painting solid black or white, and PAINTT is for using 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.

Example of a COBOL program using Graphics routines:

```
IDENTIFICATION DIVISION.
. . .                                  (any statements)
ENVIRONMENT DIVISION.
. . .
DATA DIVISION.
. . .
WORKING-STORAGE SECTION.
77  VARIABLE . . .                     (any 77 level variables)
. . .
    COPY "CBLGRAPH/CPY"
. . .
. . .
PROCEDURE DIVISION.
START-PROGRAM.
    CALL GRAPH-SUB USING GRAPHICS-PARAMETERS.
    CALL GRAPH-SUB USING GRPINI-CMD.
CLEAR-SCREENS.
    MOVE 2 TO CLEAR-KEY.
    CALL GRAPH-SUB USING CLS-CMD.      (clear text & graphics screens)
SPECIFY-X-AND-Y.
    MOVE 2ØØ TO X-COORD.
    MOVE 1ØØ TO Y-COORD.
    CALL GRAPH-SUB USING SETXY-CMD.    (current point: X,Y = 2ØØ,1ØØ)
    MOVE 5Ø TO X-COORD, Y-COORD.
    CALL GRAPH-SUB USING SETXYR-CMD.   (previous point: X,Y = 2ØØ,1ØØ
                                        current point: X,Y = 25Ø,15Ø)
DRAW-A-BOX.
    MOVE 1 TO COLOR.                   (color on -- white)
    MOVE -1 TO STYLE.                  (solid line pattern)
    CALL GRAPH-SUB USING LINEB-CMD.    (draw a box with corners
                                        2ØØ,1ØØ and 25Ø,15Ø)

     . . .
     . . .                             (more program here)
ALL-DONE.
    STOP RUN.
END PROGRAM.
```

```
CIRCLE-CMD --  Draws a circle, arc, or ellipse using the
               current x and y coordinates as the center.

    COMPUTE or MOVE a value to:
        COLOR = The color of the circle's border.
               Ø=off 1=on.
        RADIUS = The radius of the circle in pixels.
        START-CIR = The startpoint of the arc.  Absolute
               value between Ø and 6.2831 (2 * PI).
               Negative means draw a radius line.
        END-CIR = The endpoint of the arc.  Same range as
               START-CIR.  A zero value means use default
               value of 2 * PI.
        RATIO-CIR = The aspect ratio of the circle/ellipse.
               A zero value is interpreted as Ø.5. If
               RATIO-CIR is Ø.5, a circle will be drawn.
               Other values are for ellipses.

    CALL GRAPH-SUB USING CIRCLE-CMD.


CLS-CMD --     Clears the screen according to the specified
               variable.

    COMPUTE or MOVE a value to:
        CLEAR-KEY = Ø to clear text screen, 1 to clear
               graphics screen, or 2 to clear both screens.

    CALL GRAPH-SUB USING CLS-CMD.


FVIEW-CMD --   Returns the specified viewport parameter.

    COMPUTE or MOVE a value to:
        VIEW-KEY = Ø to return the starting X coordinate,
                   1 to return the starting Y coordinate,
                   2 to return the ending X coordinate,
                   3 to return the ending Y coordinate.

    CALL GRAPH-SUB USING FVIEW-CMD.

        VIEW-VALUE now contains the value of the coordinate
               requested by VIEW-KEY.


GET-CMD --     Reads the contents of a pixel block into
```

memory for future use by PUT. The previous and current X and Y coordinates define the corners of the graphics block to be read into memory. Sufficient memory must be reserved in WORKING-STORAGE for the graphics data and the name of the storage area must be passed to the graphics routine before GET-CMD may be used. (See GPBUF-CMD.)

Define an area in WORKING-STORAGE to hold the graphics data. The buffer area must be at least as large as:

(XP/8 * YP) + 4 bytes

where XP = the number of X pixels to get and
YP = the number of Y pixels to get.

```
    CALL GRAPH-SUB USING GPBUF-CMD.
    CALL GRAPH-SUB USING BUFFER.   ("BUFFER" = name of area)

    COMPUTE or MOVE a value to:
        GET-SIZE = The size of the buffer (in bytes) which was
                   passed after a call using GPBUF-CMD.

    CALL GRAPH-SUB USING GET-CMD.
```

Example use of GPBUF-CMD, GET-CMD, and PUT-CMD:

```
    . . .
  WORKING-STORAGE SECTION
    . . .
        COPY "CBLGRAPH/CPY".
  *  Reserves 524 bytes of memory for GET and PUT.
    Ø1  STORAGE
        Ø2 FILLER PIC X(24).
        Ø2 FILLER PIC X(1ØØ) OCCURS 5 TIMES.
  . . .
PROCEDURE DIVISION.
START-PROGRAM.
    CALL GRAPH-SUB USING GRAPHICS-PARAMETERS.
    CALL GRAPH-SUB USING GRPINI-CMD.
  * Draw a design and set (X,Y) to (1ØØ,5Ø) then (199,89)
    . . .
  * Pass name of storage area to graphics routine:
    CALL GRAPH-SUB USING GPBUF-CMD.
    CALL GRAPH-SUB USING STORAGE.
  * Size of area = 1ØØ/8 * 4Ø + 4
    MOVE 524 TO GET-SIZE.
    CALL GRAPH-SUB USING GET-CMD.
  * Set (X,Y) to a new point
    . . .
    CALL GRAPH-SUB USING PUT-CMD.
```

GPBUF-CMD -- Tells graphics routine that next call will specify the buffer for GET-CMD and PUT-CMD.

```
    CALL GRAPH-SUB USING GPBUF-CMD.
    CALL GRAPH-SUB USING STORAGE.
        where "STORAGE" is the name of the storage area
               defined in WORKING-STORAGE to be used for
               GET-CMD and PUT-CMD.
```

These two calls MUST be together. No other calls to any programs should be made between these calls. The buffer can be re-specified at any time by calling GRAPH-SUB using GPBUF-CMD followed by another call specifying the new buffer. Once a buffer is specified it will be used for all subsequent calls with GET-CMD or PUT-CMD until another buffer is specified.

GRPINI-CMD -- Graphics initialization routine. This function must be called before any other graphics calls are made.

```
    COMPUTE or MOVE a value to:
        INIT-KEY = Ø to Clear the Graphics Screen; anything
              else will not Clear the Screen.

    CALL GRAPH-SUB USING GRPINI-CMD.
```

LINE-CMD -- Draws a line between the previous and the current coordinates.

COMPUTE or MOVE a value to:
COLOR = The color of the line. Ø=off 1=on.
STYLE = The pattern of the line. The binary value of STYLE indicates a 16-pixel pattern for the line. A zero bit in the pattern means no change. A one bit means set that pixel according to COLOR. For a solid line. STYLE should be -1 (since the binary representation of -1 is all bits are ones).

```
CALL GRAPH-SUB USING LINE-CMD.
```

LINEB-CMD -- Same as LINE, except LINEB draws a box between the two sets of coordinates.

COMPUTE or MOVE a value to:
COLOR = The color of the box. Ø=off 1=on.
STYLE = The pattern of the box. See LINE-CMD for a description of STYLE.

```
CALL GRAPH-SUB USING LINEB-CMD.
```

LINEBF-CMD -- Same as LINEB, except LINEBF fills the box (colors in the box).

COMPUTE or MOVE a value to:
COLOR = The color to use for the filled box. Ø=off 1=on.

```
CALL GRAPH-SUB USING LINEBF-CMD.
```

PAINT-CMD -- Paints your screen in the specified color (black or white).

COMPUTE or MOVE a value to:
COLOR = The color to paint with. Ø=off 1=on.
BORDER = The color of the border where painting should stop. Ø=off 1=on.

```
CALL GRAPH-SUB USING PAINT-CMD.
```

PAINTT-CMD -- This routine allows you to paint with a precise pattern by using a technique called 'tiling'.

COMPUTE or MOVE a value to:

BORDER = The color of the border where painting should stop. Ø=off 1=on.

BACKGROUND = One byte specifying what the background is in the area to be painted. This value will normally be Ø for painting in an area that is already off or 255 (all bits = ones) for painting in an area that is already on.

NUM-TILES = The number of "tiles" in the painting pattern.

TILE array = The pattern to be used for painting. Each TILE should be a number from Ø to 255. The binary value of each TILE specifies the on/off status of a row of 8 pixels.

CALL GRAPH-SUB USING PAINTT-CMD.

POINT-CMD -- Returns the pixel value at the current x and y coordinates.

CALL GRAPH-SUB USING POINT-CMD.

POINT-VAL now contains Ø if the point was off, 1 if the point was on, or -1 if the point was not on the Screen or not in the current viewport.

PSET-CMD -- Sets a pixel defined by the current x and y coordinates either ON or OFF.

COMPUTE or MOVE a value to:

COLOR = The color to set the point. Ø=off 1=on.

CALL GRAPH-SUB USING PSET-CMD.

PRESET-CMD -- Same as PSET.

COMPUTE or MOVE a value to:

COLOR = The color to set the point. Ø=off 1=on.

CALL GRAPH-SUB USING PRESET-CMD.

PUT-CMD -- The pixel pattern (stored by GET) is put on the video screen at the current x and y coordinates set by calling SETXY or SETXYR.

```
    COMPUTE or MOVE a value to:
        ACTION = A number from 1 to 5 specifying how the
             pixels in the buffer are to be combined with
             the pixels already on the screen.  1=OR,
             2=AND, 3=PRESET, 4=PSET, and 5=XOR.

    CALL GRAPH-SUB USING PUT-CMD.
```

SCREEN-CMD -- Allows you to set the screen mode.

```
    COMPUTE or MOVE a value to:
        SCREEN-MODE = A number from Ø to 3 specifying how to
             set the graphics screen:
                       Ø = graphics on, normal speed
                       1 = graphics off, normal speed
                       2 = graphics on, high speed
                       3 = graphics off, high speed

    CALL GRAPH-SUB USING SCREEN-CMD.
```

SETXY-CMD -- Sets the previous X and Y coordinates to the current X and Y coordinates and sets new current X and Y coordinates.

```
    COMPUTE or MOVE a value to:
        X-COORD = The X coordinate
        Y-COORD = The Y coordinate

    CALL GRAPH-SUB USING SETXY-CMD.
```

SETXYR-CMD -- Sets the current x,y coordinates relative to the previously set x,y coordinates. For example, if the "current" coordinates are (1ØØ,1ØØ), SETXYR with x equal to 1Ø and y equal to 1Ø will set the "current" coordinates to (11Ø,11Ø); the "previous" coordinates will then be (1ØØ,1ØØ).

```
    COMPUTE or MOVE a value to:
        X-COORD = The X offset.  This number will be added
                 to the current X address for the new X
                 address.
        Y-COORD = The Y offset.

    CALL GRAPH-SUB USING SETXYR-CMD.
```

VIEW-CMD -- Allows you to designate specific areas of your screen where the graphics will be displayed.

```
    COMPUTE or MOVE a value to:
        X-START = The X coordinate for the start of the
                 viewport.
        Y-START = The Y coordinate for the start of the
                 viewport.
        X-END = The X coordinate for the end of the
                 viewport.
        Y-END = The Y coordinate for the end of the
                 viewport.
        COLOR = The color of the interior of the viewport.
                 Ø=off 1=on, -1 = don't color the viewport.
        BORDER = The color of the border of the viewport.
                 Ø=off 1=on, -1 = border is not drawn.

    CALL GRAPH-SUB USING VIEW-CMD.
```

Calling Graphics Utilities from a COBOL Program

The graphics utility programs GLOAD, GPRINT, GSAVE, and VDOGRPH may be called from a Cobol program by calling GRAPH-SUB using one of the "-UTIL" options. When any of these options are called no Cobol files should be open as the system will automatically close any open files when one of the utility programs is loaded.

GLOAD-UTIL

Loads graphics memory from a disk file previously written by GSAVE.

```
          MOVE the filespec to GFILE.
          CALL GRAPH-SUB USING GLOAD-UTIL.
```

GPRINT-UTIL

Prints graphics memory on a graphics printer.

```
          CALL GRAPH-SUB USING GPRINT-UTIL.
```

GSAVE-UTIL

Writes graphics memory to a disk file.

```
          MOVE the filespec to GFILE.
          CALL GRAPH-SUB USING GSAVE-UTIL.
```

VDOGRPH-UTIL

Converts the video text display to graphics memory.

```
          CALL GRAPH-SUB USING VDOGRPH-UTIL.
```

COBOL Copy Source Code Listing

```
ØØØ1ØØ*  CBLGRAPH/CPY -- COBOL graphics parameter definitions.
ØØØ11Ø*
ØØØ12Ø*    This file should be included in the source for any
ØØØ13Ø*    Cobol program that will use Graphics Subroutines.
ØØØ14Ø*    To do this put this statement in the WORKING-STORAGE SECTION
ØØØ15Ø*    after any 77 level items:
ØØØ16Ø*
ØØØ17Ø*    COPY "CBLGRAPH/CPY".
ØØØ18Ø*
ØØØ19Ø Ø1  GRAPH-SUB.
ØØØ2ØØ*  Name of subroutine to be called is "CBLGRAPH/CMD".
ØØØ21Ø*  Use "CALL GRAPH-SUB USING ......." to call graphics.
ØØØ22Ø*
ØØØ23Ø     Ø2  FILLER  PIC X(12)   VALUE "CBLGRAPH/CMD".
ØØØ24Ø*
ØØØ25Ø*
ØØØ26Ø*
ØØØ27Ø Ø1  GRAPHICS-PARAMETERS.
ØØØ28Ø*  Parameters for graphics routines defined here.
ØØØ29Ø*  First call to graphics MUST be:
ØØØ3ØØ*
ØØØ31Ø*    CALL GRAPH-SUB USING GRAPHICS-PARAMETERS.
ØØØ32Ø*
ØØØ33Ø*  ARGS-KEY must be zero.  Do NOT change this value.
ØØØ34Ø     Ø2  ARGS-KEY    COMP-1  PIC 99 VALUE Ø.
ØØØ35Ø*
ØØØ36Ø*  Init key for GRPINI-CMD (Ø=clear, >Ø=don't clear Graphics)
ØØØ37Ø     Ø2  INIT-KEY            PIC 9  VALUE Ø.
ØØØ38Ø*
ØØØ39Ø*  X and Y Coordinates (relative or absolute)
ØØØ4ØØ     Ø2  X-COORD     COMP-1  PIC S9(5)  VALUE Ø.
ØØØ41Ø     Ø2  Y-COORD     COMP-1  PIC S9(5)  VALUE Ø.
ØØØ42Ø*
ØØØ43Ø*  Color and Border (Ø=off, 1=on; -1=none for VIEW-CMD)
ØØØ44Ø     Ø2  COLOR       COMP-1  PIC S9  VALUE 1.
ØØØ45Ø     Ø2  BORDER      COMP-1  PIC S9  VALUE 1.
ØØØ46Ø*
ØØØ47Ø*  Point value returned by POINT-CMD:
ØØØ48Ø*    Ø=point is off, 1=point is on, -1=point is not on the screen
ØØØ49Ø Ø2  POINT-VAL       COMP-1  PIC S9.
ØØØ5ØØ*
ØØØ51Ø*  Screen clear key (Ø=text, 1=graphics, 2=both)
ØØØ52Ø     Ø2  CLEAR-KEY           PIC 9  VALUE 2.
ØØØ53Ø*
ØØØ54Ø*  Line style: 16 bit pattern (-1 = solid line)
ØØØ55Ø     Ø2  STYLE       COMP-1  PIC S9(5)  VALUE -1.
```

```
ØØØ56Ø*
ØØØ57Ø*  Screen mode:  (Must be Ø, 1, 2, or 3)
ØØØ58Ø*    Ø = graphics on,  normal speed
ØØØ59Ø*    1 = graphics off, normal speed
ØØØ6ØØ*    2 = graphics on,  high speed
ØØØ61Ø*    3 = graphics off, high speed
ØØØ62Ø     Ø2  SCREEN-MODE          PIC 9  VALUE Ø.
ØØØ63Ø*
ØØØ64Ø*  Circle parameters
ØØØ65Ø     Ø2  RADIUS      COMP-1   PIC 999  VALUE Ø.
ØØØ66Ø     Ø2  START-CIR   COMP     PIC S9V9(4)  VALUE Ø.
ØØØ67Ø     Ø2  END-CIR     COMP     PIC S9V9(4)  VALUE Ø.
ØØØ68Ø     Ø2  RATIO-CIR   COMP     PIC 9(4)V9(4)  VALUE Ø.5.
ØØØ69Ø*
ØØØ7ØØ*  Viewport parameters
ØØØ71Ø     Ø2  X-START     COMP-1   PIC S9(5)  VALUE Ø.
ØØØ72Ø     Ø2  X-END       COMP-1   PIC S9(5)  VALUE 639.
ØØØ73Ø     Ø2  Y-START     COMP-1   PIC S9(5)  VALUE Ø.
ØØØ74Ø     Ø2  Y-END       COMP-1   PIC S9(5)  VALUE 239.
ØØØ75Ø     Ø2  VIEW-KEY             PIC 9  VALUE Ø.
ØØØ76Ø     Ø2  VIEW-VALUE  COMP-1   PIC 999.
ØØØ77Ø*
ØØØ78Ø*  Size of get/put buffer in bytes:
ØØØ79Ø*    Must be greater than or equal to
ØØØ8ØØ*        number of X pixels / 8 * number of Y pixels + 4
ØØØ81Ø*
ØØØ82Ø*  Get/Put buffer should be defined separately in WORKING-STORAGE.
ØØØ83Ø*  Before using GET-CMD or PUT-CMD tell graphics routine where
ØØØ84Ø*  the get/put storage buffer is by the following calls:
ØØØ85Ø*
ØØØ86Ø*    CALL GRAPH-SUB USING GPBUF-CMD.
ØØØ87Ø*    CALL GRAPH-SUB USING STORAGE.
ØØØ88Ø*
ØØØ89Ø*  where "STORAGE" is the name of the storage area for
ØØØ9ØØ*  Get & Put.
ØØØ91Ø*
ØØØ92Ø     Ø2  GET-SIZE    COMP-1   PIC 9(5).
ØØØ93Ø*
ØØØ94Ø*  Action key for PUT-CMD.  Must be 1, 2, 3, 4, or 5.
ØØØ95Ø*    1 = OR    2 = AND    3 = PRESET    4 = PSET    5 = XOR
ØØØ96Ø     Ø2  ACTION               PIC 9  VALUE 4.
ØØØ97Ø*
ØØØ98Ø*  Filespec for GLOAD-UTIL and GSAVE-UTIL
ØØØ99Ø     Ø2  GFILE                PIC X(33)  VALUE SPACE.
ØØ1ØØØ*
ØØ1Ø1Ø*  Background tile for PAINTT-CMD (Ø=black, 255= white)
ØØ1Ø2Ø     Ø2  BACKGROUND  COMP-1   PIC 999  VALUE Ø.
ØØ1Ø3Ø*
ØØ1Ø4Ø*  Tiling for PAINTT-CMD.  Each tile specifies 8 pixels across.
```

```
ØØ1Ø5Ø     Ø2  NUM-TILES   COMP-1  PIC 99.
ØØ1Ø6Ø     Ø2  TILE  OCCURS 1 TO 64 TIMES DEPENDING ON NUM-TILES
ØØ1Ø7Ø               INDEXED BY TILE-NO
ØØ1Ø8Ø                     COMP-1  PIC 999.
ØØ1Ø9Ø*
ØØ11ØØ*
ØØ111Ø*
ØØ112Ø Ø1  GRAPHICS-OPTIONS    COMP-1.
ØØ113Ø*  Use one of the following for parameter with "CALL GRAPH-SUB".
ØØ114Ø*  Example:
ØØ115Ø*
ØØ116Ø*    CALL GRAPH-SUB USING CLS-CMD.
ØØ117Ø*  clears screen(s) depending on value of CLEAR-KEY.
ØØ118Ø*
ØØ119Ø*        Option             Variables used
ØØ12ØØ*        ------             --------------
ØØ121Ø     Ø2  CIRCLE-CMD                              PIC 99 VALUE  1.
ØØ122Ø*                        COLOR       RADIUS
ØØ123Ø*                        START-CIR   END-CIR
ØØ124Ø*                        RATIO-CIR
ØØ125Ø*
ØØ126Ø     Ø2  CLS-CMD                                 PIC 99 VALUE  2.
ØØ127Ø*                        CLEAR-KEY
ØØ128Ø*
ØØ129Ø     Ø2  FVIEW-CMD                               PIC 99 VALUE  3.
ØØ13ØØ*                        VIEW-KEY
ØØ131Ø*                        VIEW-VALUE returned
ØØ132Ø*
ØØ133Ø     Ø2  GET-CMD                                 PIC 99 VALUE  4.
ØØ134Ø*                        GET-SIZE
ØØ135Ø*                        Buffer passed after GPBUF-CMD call
ØØ136Ø*
ØØ137Ø     Ø2  GPBUF-CMD                               PIC 99 VALUE  5.
ØØ138Ø*                        Next call passes GET/PUT Buffer
ØØ139Ø*
ØØ14ØØ     Ø2  GRPINI-CMD                              PIC 99 VALUE  6.
ØØ141Ø*                        INIT-KEY
ØØ142Ø*
ØØ143Ø     Ø2  LINE-CMD                                PIC 99 VALUE  7.
ØØ144Ø*                        COLOR       STYLE
ØØ145Ø*
ØØ146Ø     Ø2  LINEB-CMD                               PIC 99 VALUE  8.
ØØ147Ø*                        COLOR       STYLE
ØØ148Ø*
ØØ149Ø     Ø2  LINEBF-CMD                              PIC 99 VALUE  9.
ØØ15ØØ*                        COLOR
ØØ151Ø*
ØØ152Ø     Ø2  PAINT-CMD                               PIC 99 VALUE 1Ø.
ØØ153Ø*                        COLOR       BORDER
```

```
ØØ154Ø*
ØØ155Ø     Ø2  PAINTT-CMD                              PIC 99 VALUE 11.
0/Ø156Ø*                        NUM-TILES   TILE array
ØØ157Ø*                        BORDER      BACKGROUND
ØØ158Ø*
ØØ159Ø     Ø2  POINT-CMD                               PIC 99 VALUE 12.
ØØ16ØØ*                        POINT-VAL returned
ØØ161Ø*
ØØ162Ø     Ø2  PRESET-CMD                              PIC 99 VALUE 13.
ØØ163Ø*                        COLOR
ØØ164Ø*
ØØ165Ø     Ø2  PSET-CMD                                PIC 99 VALUE 14.
ØØ166Ø*                        COLOR
ØØ167Ø*
ØØ168Ø     Ø2  PUT-CMD                                 PIC 99 VALUE 15.
ØØ169Ø*                        ACTION
ØØ17ØØ*                        Buffer passed after GPBUF-CMD call
ØØ171Ø*
ØØ172Ø     Ø2  SCREEN-CMD                              PIC 99 VALUE 16.
ØØ173Ø*                        SCREEN-MODE
ØØ174Ø*
ØØ175Ø     Ø2  SETXY-CMD                               PIC 99 VALUE 17.
ØØ176Ø*                        X-COORD     Y-COORD
ØØ177Ø*
ØØ178Ø     Ø2  SETXYR-CMD                              PIC 99 VALUE 18.
ØØ179Ø*                        X-COORD     Y-COORD
ØØ18ØØ*
ØØ181Ø     Ø2  VIEW-CMD                                PIC 99 VALUE 19.
ØØ182Ø*                        X-START     X-END
ØØ183Ø*                        Y-START     Y-END
ØØ184Ø*                        COLOR       BORDER
ØØ185Ø*
ØØ186Ø*
ØØ187Ø*  Graphics utilities
ØØ188Ø*
ØØ189Ø     Ø2  GLOAD-UTIL                              PIC 99 VALUE 2Ø.
ØØ19ØØ*                        GFILE
ØØ191Ø*
ØØ192Ø     Ø2  GPRINT-UTIL                             PIC 99 VALUE 21.
ØØ193Ø*                        none
ØØ194Ø*
ØØ195Ø     Ø2  GSAVE-UTIL                              PIC 99 VALUE 22.
ØØ196Ø*                        GFILE
ØØ197Ø*
ØØ198Ø     Ø2  VDOGRPH-UTIL                            PIC 99 VALUE 23.
ØØ199Ø*                        none
```

COBOL Graphics Interface Source Listing

```
        NAME     ('CBLGRAPH')
        ENTRY    START
        .SALL
;
;  Macro definitions
;
GETARG  MACRO                 ;Put address of Cobol arg in HL
        LD       H,(IX+3)
        LD       L,(IX+2)
        ENDM
GETB    MACRO    XR,XT,XA     ;Pass byte arg for subroutine
        LD       A,(IY+XA)
        SUB      'Ø'
        LD       XR,XT
        LD       (XR),A
        ENDM
GETB2   MACRO    XR,XT,XA     ;Pass 2nd byte of integer for sub.
        LD       A,(IY+XA+1)
        LD       XR,XT
        LD       (XR),A
        ENDM
GETI    MACRO    XR,XT,XA     ;Pass integer arg for subroutine
        LD       A,(IY+XA)
        LD       (XT+1),A
        LD       A,(IY+XA+1)
        LD       XR,XT
        LD       (XR),A
        ENDM
;
;  Permanent storage.  Must be retained between calls.
;
CBLARY  EQU      START-4
GPBUF   EQU      START-2
;
;*******************************************************************
;  Program starts here.
;*******************************************************************
;
START:  LD       (KEEPSP),SP  ;Save Stack pointer for COBOL
        LD       A,(TESTI)    ;Has $INIT been done?
        OR       A
        JR       NZ,FIRST
        LD       SP,(TOPSTK)  ;Restore Fortran's stack
        JP       READY        ;  and begin
;
;  Storage for Cobol values.
```

```
Ø47 ,
Ø48 TOPSTK: DEFS      2
Ø49 TESTI:  DEFB      1
Ø5Ø KEEPSP: DEFS      2
Ø51 ;
Ø52 ;  Arguments for circle
Ø53 ;
Ø54 CIRARG: DEFW      STCF
Ø55         DEFW      ECF
Ø56         DEFW      RATF
Ø57 ;
Ø58 ;  Command names & lengths for utilities
Ø59 ;
Ø6Ø CGLOAD: DEFB      5
Ø61         DEFM      'GLOAD'
Ø62 CGPRNT: DEFB      6
Ø63         DEFM      'GPRINT'
Ø64 CGSAVE: DEFB      5
Ø65         DEFM      'GSAVE'
Ø66 CVDOG:  DEFB      7
Ø67         DEFM      'VDOGRPH'
Ø68 ;
Ø69 ;  GET/PUT buffer flag
Ø7Ø ;
Ø71 GPFLAG: DEFB      Ø
Ø72 ;
Ø73 ;  Temporary storage area
Ø74 ;
Ø75 ARG1:   DEFS      2
Ø76 ARG2:   DEFS      2
Ø77 TEMP:   DEFS      3Ø
Ø78 STCF:   DEFS      4
Ø79 ECF:    DEFS      4
Ø8Ø RATF:   DEFS      4
Ø81 CPAR:   DEFS      1Ø
Ø82 ;
Ø83 FIRST:  XOR       A             ;First call: initialize Fortran
Ø84         LD        (TESTI),A
Ø85         LD        DE,ØEFFFH
Ø86         LD        BC,READY
Ø87         JP        $INIT##
Ø88 ;
Ø89 ;  Initialization done.  Begin execution here.
Ø9Ø ;
Ø91 READY:  LD        IY,(CBLARY)   ;IY points to Cobol parameters
Ø92         GETARG                  ;Get address of subroutine number
Ø93         LD        A,(GPFLAG)    ;Was last call GPBUF?
Ø94         LD        B,A
Ø95         XOR       A
```

```
Ø96          CP       B
Ø97          JR       Z,GOCMD
Ø98          LD       (GPFLAG),A     ;Last call was GPBUF.
Ø99          LD       (GPBUF),HL     ;Argument is address of GET/PUT buffer
1ØØ          JP       DONE
1Ø1 ;
1Ø2 GOCMD:   INC      HL             ;Subroutine number is in second byte
1Ø3          LD       A,(HL)
1Ø4          ADD      A,A            ;Offset = subroutine number * 2
1Ø5          LD       C,A
1Ø6          LD       HL,JMPTBL
1Ø7          ADD      HL,BC          ;Add offset to jump table
1Ø8          LD       E,(HL)         ;Get jump address
1Ø9          INC      HL
11Ø          LD       D,(HL)
111          EX       DE,HL
112          JP       (HL)           ;And go to subroutine
113 ;
114 ;  Convert 5 byte Ascii string at (HL) to floating point
115 ;
116 CFLT:    PUSH     HL
117          LD       B,1
118          LD       A,21
119          RST      8
12Ø          EX       DE,HL
121          CALL     $CA##
122          LD       HL,$AC##
123          POP      DE
124          LD       BC,4
125          LDIR
126          RET
127 ;
128 ;  Convert Cobol COMP PIC S9V9(4) to floating point
129 ;
13Ø FLOAT1:  PUSH     IY
131          POP      HL
132          ADD      HL,BC
133          LD       DE,CPAR+4
134          LD       BC,6
135          LDIR
136          LD       HL,Ø
137          LD       (CPAR),HL
138          LD       (CPAR+2),HL
139          JR       FLOAT
14Ø ;
141 ;  Convert Cobol COMP PIC S9(4)V9(4) to floating point
142 ;
143 FLOAT2:  PUSH     IY
144          POP      HL
```

```
          ADD       HL,BC
          LD        DE,CPAR+1
          LD        BC,8
          LDIR
          XOR       A
          LD        (CPAR),A
          LD        (CPAR+9),A
;
;  Convert to floating point from Cobol COMP PIC S9(5)V9(5)
;
FLOAT:    LD        B,9
          LD        C,'Ø'
          LD        HL,CPAR
CDISP:    LD        A,(HL)         ;Convert COMP to Ascii
          OR        C
          LD        (HL),A
          INC       HL
          DJNZ      CDISP
          LD        HL,CPAR        ;Convert left of dec. to float
          CALL      CFLT
          LD        A,'Ø'
          LD        HL,CPAR+4      ;Convert right of dec. to float
          LD        (HL),A
          CALL      CFLT
          LD        HL,1ØØØØ       ;Divide fraction part by 1Ø,ØØØ
          CALL      $DA##
          LD        HL,CPAR        ;And add to whole number part
          CALL      $AB##
          LD        A,(CPAR+9)
          CP        ØDH            ;Negative number ?
          JR        NZ,POS
          LD        HL,-1          ;Multiply by -1 if negative
          CALL      $MA##
POS:      LD        HL,$AC         ;Set up for move (LDIR)
          LD        BC,4
          RET
;
;  Pack array from Cobol COMP-1 to bytes
;
PACKA:    PUSH      DE
          INC       HL
          LD        C,(HL)
          INC       C
          XOR       A
          LD        B,A
LOOPP:    LDI
          INC       HL
          CP        C
          JR        NZ,LOOPP
```

```
        POP      HL
        RET
;
;  GET COLOR FROM COBOL INTO ARG1, ADDRESS IN HL
;
GCOLOR: GETB2    HL,ARG1,COLOR
        RET
;
;  SET UP FOR CALL TO LINE (B,BF)
;
SETLIN: CALL     GCOLOR
        GETI     DE,ARG2,STYLE
        RET
;
;  SET UP X & Y COORDINATE ARGUMENTS FOR SETXY (R)
;
GCOORD: GETI     HL,ARG1,XCOORD
        GETI     DE,ARG2,YCOORD
        RET
;
;  Move command to buffer
;
MVCMD:  PUSH     HL
        LD       HL,ARG1
        LD       A,' '
        LD       (HL),A
        LD       DE,ARG1+1
        LD       BC,38
        LDIR                  ;Fill buffer with blanks
        POP      HL
        LD       C,(HL)       ;Get command length
        INC      HL
        LD       B,Ø
        LD       DE,ARG1
        LDIR                  ;Move command to buffer
        RET
;
;  Jump table.  Address of procedure for each command.
;
JMPTBL: DEFW     JARGS
        DEFW     JCIRCL
        DEFW     JCLS
        DEFW     JFVIEW
        DEFW     JGET
        DEFW     JGPBUF
        DEFW     JGRPIN
        DEFW     JLINE
        DEFW     JLINEB
        DEFW     JLINEF
        DEFW     JPAINT
        DEFW     JPANTT
        DEFW     JPOINT
        DEFW     JPRSET
        DEFW     JPSET
        DEFW     JPUT
        DEFW     JSCREN
        DEFW     JSETXY
        DEFW     JSTXYR
        DEFW     JVIEW
        DEFW     GLOAD
        DEFW     GPRINT
        DEFW     GSAVE
        DEFW     VDOGRP
;
;  Offsets into Cobol parameter structure.
;
;  Init key (Ø=clear, >Ø=don't clear Graphics)
INITKY  EQU      Ø
;  X and Y coordinates (Relative or absolute)
XCOORD  EQU      INITKY+1
YCOORD  EQU      XCOORD+2
;  Color, border, point value (Ø=off 1=on -1=neither)
COLOR   EQU      YCOORD+2
BORDER  EQU      COLOR+2
PVAL    EQU      BORDER+2
CLEAR   EQU      PVAL+2       ;Ø=text, 1=graphics, 2=both
STYLE   EQU      CLEAR+1      ;-1 = solid line
SCMODE  EQU      STYLE+2      ;Screen mode (Ø-3)
;  Circle parameters
RADIUS  EQU      SCMODE+1
STCIR   EQU      RADIUS+2
ECIR    EQU      STCIR+6
RATIO   EQU      ECIR+6
;  Parameters for view-port
LEFTX   EQU      RATIO+8
RIGHTX  EQU      LEFTX+2
LEFTY   EQU      RIGHTX+2
RIGHTY  EQU      LEFTY+2
FVCTL   EQU      RIGHTY+2
FVRTN   EQU      FVCTL+1
;  Parameters for get & put
GSIZE   EQU      FVRTN+2
ACTION  EQU      GSIZE+2
;  Filespec for GLOAD & GSAVE
GFILE   EQU      ACTION+1
;  Parameters for PAINTT
BACGND  EQU      GFILE+33     ;Background tile
NUMTIL  EQU      BACGND+2     ;Number of tiles
```

```
;
;  Define Cobol parameters address
;
JARGS:  GETARG
        INC      HL
        INC      HL
        LD       (CBLARY),HL
        JP       DONE
;
;  Circle
;
JCIRCL: LD       BC,STCIR     ;Convert params to float
        CALL     FLOAT1
        LD       DE,STCF
        LDIR
        LD       BC,ECIR
        CALL     FLOAT1
        LD       DE,ECF
        LDIR
        LD       BC,RATIO
        CALL     FLOAT2
        LD       DE,RATF
        LDIR
        CALL     GCOLOR
        GETI     DE,ARG2,RADIUS
        EX       DE,HL
        LD       BC,CIRARG
        CALL     CIRCLE##
        JP       DONE
;
;  Clear screen(s)
;
JCLS:   GETB     HL,ARG1,CLEAR
        CALL     CLS##
        JP       DONE
;
;  Return X or Y coordinate of view-port
;
JFVIEW: GETB     HL,ARG1,FVCTL
        CALL     FVIEW##
        LD       (IY+FVRTN),H
        LD       (IY+FVRTN+1),L
        JP       DONE
;
;  Get pixel block
;
JGET:   LD       HL,(GPBUF)
        GETI     DE,ARG1,GSIZE
        CALL     GET##
        JP       DONE
;
;  Get address of GET/PUT buffer (will be passed next call)
;
JGPBUF: LD       A,1
        LD       (GPFLAG),A
        JP       DONE
;
;  Initialize Graphics board and subroutines
;
JGRPIN: GETB     HL,ARG1,INITKY
        CALL     GRPINI##
        JP       DONE
;
;  Draw a line from previous X,Y to current
;
JLINE:  CALL     SETLIN
        CALL     LINE##
        JP       DONE
;
;  Draw a box
;
JLINEB: CALL     SETLIN
        CALL     LINEB##
        JP       DONE
;
;  Draw a filled box
;
JLINEF: CALL     SETLIN
        CALL     LINEBF##
        JP       DONE
;
;  Paint an area
;
JPAINT: CALL     GCOLOR
        GETB2    DE,ARG2,BORDER
        CALL     PAINT##
        JP       DONE
;
;  Paint with tiling
;
JPANTT: LD       DE,TEMP
        PUSH     IY
        POP      HL
        LD       BC,NUMTIL
        ADD      HL,BC        ;(HL) is address of tiling array
        CALL     PACKA
        GETB2    DE,ARG1,BORDER
        LD       A,(IY+BACGND+1)
        LD      (ARG2+1),A
        LD      A,1
        LD      BC,ARG2
        LD      (BC),A
        CALL    PAINTT##
        JP      DONE
;
;  Return on/off status of current X,Y point
;
JPOINT: CALL    POINT##       ;Returns Ø, 1, or -1
        LD      (IY+PVAL+1),A
        SRA     A
        LD      (IY+PVAL),A
        JP      DONE
;
;  Turn pixel at current X,Y point on or off
;
JPRSET: CALL    GCOLOR
        CALL    PRESET##
        JP      DONE
;
;  Turn pixel at current X,Y point on or off
;
JPSET:  CALL    GCOLOR
        CALL    PSET##
        JP      DONE
;
;  Display pixel array at current X,Y
;
JPUT:   LD      HL,(GPBUF)
        GETB    DE,ARG1,ACTION
        CALL    PUT##
        JP      DONE
;
;  Change screen mode
;
JSCREN: GETB    HL,ARG1,SCMODE
        CALL    SCREEN##
        JP      DONE
;
;  Set X,Y absolute
;
JSETXY: CALL    GCOORD
        CALL    SETXY##
        JP      DONE
;
;  Set X,Y relative
;
JSTXYR: CALL    GCOORD
```

```
        CALL    SETXYR##
        JP      DONE
;
;  Create a view-port
;
JVIEW:  GETI    HL,TEMP,RIGHTX
        LD      (CPAR),HL
        GETI    HL,TEMP+2,RIGHTY
        LD      (CPAR+2),HL
        GETB2   HL,TEMP+4,COLOR
        LD      (CPAR+4),HL
        GETB2   HL,TEMP+5,BORDER
        LD      (CPAR+6),HL
        GETI    HL,ARG1,LEFTX
        GETI    DE,ARG2,LEFTY
        LD      BC,CPAR
        CALL    VIEW##
        JP      DONE
;
;  Graphics utilities
;
GLOAD:  LD      HL,CGLOAD
        JR      FILCMD
;
GPRINT: LD      HL,CGPRNT
        JR      NCMD
;
GSAVE:  LD      HL,CGSAVE
        JR      FILCMD
;
VDOGRP: LD      HL,CVDOG
        JR      NCMD
;
;  Execute TRSDOS command with filespec
;
FILCMD: CALL    MVCMD
        PUSH    IY
        POP     HL
        LD      BC,GFILE
        ADD     HL,BC
        LD      DE,ARG1+6
        LD      BC,33
        LDIR
        JR      EXCMD
;
;  Execute TRSDOS command without filespec
;
NCMD:   CALL    MVCMD
EXCMD:  LD      HL,ARG1
```

```
488          LD      B,39
489          LD      A,38
49Ø          RST     8
491 ;
492 ;  Done with command.   Return to Cobol.
493 ;
494 DONE:    LD      (TOPSTK),SP   ;Save stack pointer for next call
495          LD      SP,(KEEPSP)   ;Restore COBOL's stack pointer
496          XOR     A             ;A reg must be zero for COBOL
497          RET
498 ;
499          EXTRN   $IOERR        ;Fortran routines missed on first
5ØØ          EXTRN   $IOINI        ;  pass of loader.  Declared here
5Ø1          EXTRN   $LUNTB        ;  to force them be loaded
5Ø2 ;
5Ø3          END     START
```

7/ Programming the Graphics Board

The Graphics Board provides 64Ø X 24Ø byte addressable pixels on a TRS-8Ø Model II. The Graphics Board contains 32K of screen RAM to store video data. Regular alphanumeric data is stored in the static RAM on the Video board. The Graphics Board uses the Video board's circuitry as much as possible to minimize the hardware.

I/O port mapping is used to read and write data to the board. A DIP switch selects a 16-byte boundary (ØØH, 1ØH, 2ØH...FØH) in the entire I/O space. The use of port mapping allows the board to reside transparent to TRSDOS.

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 79) for data write only.
2. **Y-Position** - Y-address (Ø to 239) for data write only.
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
- x1 - Y-Register Write.
- x2 - Video data read or write.
- x3 - options write.

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.

An option lets the Graphics Board insert WAIT STATES any time the graphics RAM is not accessed during a retrace. This prevents "flashing" of the display. The worse case access time for a read or write would be 64 uS, as opposed to about 12 uS without wait states. Another way to prevent flashing is to blank out the graphics display until all drawing is complete, then turn the graphics on. The hardware is such that the alphanumeric video data and the graphics data are overlaid. When you try to overlay solid white graphics directly over alphanumerics, the alphanumerics will appear as Reverse Video so they can be read.

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 must not allow the counters to go past 79 and 239 or unpredictable results may occur.

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,1Ø11ØØØ1B ;OPTIONS:INCREMENT X AFTER WRITE
     OUT (83H),A      ;AND NO WAITS
     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 ON and OFF graphics. "1" turns graphics ON.
1	WAITS ON/OFF*	If WAITS are /ON the screen does not "flash" when Reading or Writing to graphics. A "1" selects WAITS.
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

Appendix A/ BASICG/Utilities Reference Summary

Utilities are shaded like this.

Argument ranges are indicated below by special letters and words:

ar is a single-precision floating point number > Ø.Ø (to 1* 1Ø^{38}).

b	is an integer expression of either Ø or 1.
B	specifies a box.
BF	specifies a shaded box.
c	is an integer expression of Ø or 1.
n	is an integer expression from Ø to 2.
p	is an integer expression from Ø to 3.
r	is an integer expression from Ø to 639.
x	is an integer expression from Ø to 639.
y	is an integer expression from Ø to 239.
action	is either AND, PSET, PRESET, OR, or XOR.
background	is a string.
border	is an integer expression of either Ø or 1.
end	is an expression from -6.283185 to 6.283185.
start	is an expression from -6.283185 to 6.283185.
switch	is an integer expression of Ø or 1.
tiling	is a string or an integer expression of Ø or 1.
type	is an integer expression from Ø to 3.

CIRCLE(x,y)r,c,start,end,ar Draws circle, ellipse, semi-circle, arc, or point.

```
  CIRCLE(1ØØ,1ØØ),25,1          CIRCLE(15Ø,15Ø),4Ø,1,,,.6
  CIRCLE(1ØØ,1ØØ),1ØØ,PI,2*PI,5         CIRCLE(-5Ø,-5Ø),2ØØ
```

CLS Clears the Text Screen and video memory.

```
  CLS          SYSTEM"CLS"
```

CLS n Clears Screen(s).

```
  CLS          CLS2
```

GCLS Clears the Graphics Screen and memory.

```
  GCLS      SYSTEM"GCLS"       1ØØ SYSTEM"GCLS"
```

GET(xl,yl)-(x2,y2),array name Reads the contents of a rectangular pixel area into an array.

```
  GET(1Ø,1Ø)-(5Ø,5Ø),V
```

GLOAD filename /ext .password :d (diskette name) Loads graphics memory.

```
  GLOAD PROG    SYSTEM"GLOAD PROG"
```

GPRINT Dumps graphic display to printer.

```
  GPRINT      SYSTEM"GPRINT"    1ØØ SYSTEM"GPRINT"
```

GSAVE filename /ext .password :d (diskette name) Saves graphics memory.

```
  GSAVE PROG       SYSTEM"GSAVE PROG"
```

GROFF Turn Graphic Display OFF.

```
  GROFF   SYSTEM "GROFF"
```

GRON Turn Graphic Display ON.

```
  GRON  SYSTEM "GRON"
```

LINE(xl,yl)-(x2,y2),c,B or BF, style Draws a line/box.

```
  LINE -(1ØØ,1ØØ)            LINE(1ØØ,1ØØ)-(2ØØ,2ØØ),1,B,45
  LINE(Ø,Ø)-(1ØØ,1ØØ),1,BF      LINE(-2ØØ,-2ØØ)-(1ØØ,1ØØ)
```

PAINT(x,y),tiling,border,background Paints Screen.

```
  PAINT(32Ø,12Ø),1,1          PAINT(32Ø,12Ø),"DDDDD",1
  PAINT(32Ø,12Ø),A$,1
  PAINT(32Ø,12Ø),CHR$(Ø)+CHR$(&HFF),Ø,CHR$(&HØØ)
  PAINT(32Ø,12Ø),CHR$(E)+CHR$(77)+CHR$(3)
```

POINT(x,y) A function. Tests graphics point.

```
  PRINT POINT(32Ø,12Ø)      IF POINT(32Ø,12Ø)=1 THEN . . .
  PRINT POINT(32Ø,12Ø),-1
```

PRESET(x,y),switch Sets pixel OFF or ON.

```
  PRESET(1ØØ,1ØØ),Ø
```

PSET(x,y),switch Sets pixel ON or OFF.

```
  PSET(1ØØ,1ØØ),1
```

PUT(xl,yl),array name,action Puts graphics from an array onto the Screen.

```
  PUT(1ØØ,1ØØ),A,PSET         PUT(1ØØ,1ØØ),A,AND
  PUT(A,B),B
```

SCREEN _type_ Selects Screen/graphics speed.

```
  SCREEN 2
```

VDOGRPH Transfers video memory to graphics memory.

```
  VDOGRPH          SYSTEM"VDOGRPH"          1ØØ SYSTEM"VDOGRPH"
```

VIEW(_x1,y1_**)-(**_x2,y2_**),**_c_**,**_b_ Redefines Screen and creates a viewport.

```
  VIEW(1ØØ,1ØØ)-(15Ø,15Ø)      VIEW(1ØØ,1ØØ)-(15Ø,15Ø),Ø,1
```

VIEW(_p_**)** A function. Returns viewport's coordinates.

```
  PRINT VIEW(1)
```

Appendix B/ BASICG Error Messages

Code	Abbre-viation	Explanation
1	NF	NEXT without FOR. NEXT is used without a matching FOR statement. This error may also occur if NEXT variables are reversed in a nested loop.
2	SN	Syntax. This is usually the result of incorrect punctuation, an illegal character or a misspelled command.
3	RG	RETURN without GOSUB. A RETURN statement was executed with insufficient data available. The DATA statement may have been left out or all data may have been read.
4	OD	Out of data. A READ statement was executed with insufficient data available. The DATA statement may have been left out or all data may have been read.
5	FC	Illegal function call. An attempt was made to executed an operation using an illegal parameter. Graphic examples: PUTing a display that is partially off the Screen, GETing an array that is not properly dimensioned, or using more than two OFF tiles or two ON tiles in a strings when tiling (with PAINT).
6	OV	Overflow. The magnitude of the number derived or input is too large for the data storage type assigned to it. The integer range is (-32768 to 32767) for BASICG.
7	OM	Out of memory. All available memory has been used or reserved. This may occur with large array dimensions and

		nested branches such as GOSUB and FOR/NEXT loops.
8	UL	Undefined line. An attempt was made to reference a non-existent line.
9	BS	Bad subscript. An attempt was made to assign an array element with a subscript beyond the dimensioned range.
1Ø	DD	Double-dimensioned array. An attempt was made to dimension an array which had previously been created with DIM or by default statements. ERASE must be used first.
11	/Ø	Division by zero. An attempt was made to use a value of zero in the denominator. Note: If you can't find an obvious division by zero, check for division by numbers smaller than allowable ranges (see OV above).
12	ID	Illegal direct. An attempt was made to use a program-only statement like INPUT in an immediate (non-program) line.
13	TM	Type mismatch. An attempt was made to assign a number to a string variable or a string to a numeric variable.
14	OS	Out of string space. The amount of string space allocated was exceeded. Use CLEAR to allocate more string space. 1ØØ bytes is the default string space allocation.
15	LS	Long string. A string variable was assigned a string which exceeded 255 characters in length.
16	ST	String too complex. A string operation was too complex to handle. The operation must be broken into shorter steps.
17	CN	Can't continue. A CONT command was given at a point where the command can't be carried out, e.g., directly after the

		program has been edited.
18	UF	Undefined user function. An attempt has been made to call a USR function without first defining its entry point via a DEFUSR statement.
19	NR	No RESUME. During an error-trapping routine, BASIC has reached the end of the program without encountering a RESUME.
2Ø	RW	RESUME without error. A RESUME was encountered when no error was present. You need to insert END or GOTO in front of the error-handling routine.
21	UE	Undefined error. Reserved for future use.
22	MO	Missing operand. An operation was attempted without providing one of the required operands.
23	BO	Buffer overflow. An attempt was made to input a data line which has too many characters to be held in the line buffer.
24	NB	Files not compatible. An attempt was made to load a BASIC file (in compressed format) into BASICG.
25-49	UE	Undefined error. Reserved for future use.
5Ø	FO	Field overflow. An attempt was made to Field more characters than the direct-access file record length allows. The record length is assigned when the file is first opened. The default length is 256.
51	IE	Internal error. Also indicates an attempt to use EOF on a file which is not open.
52	BN	Bad file number. An attempt was made to use a file number which specifies a file that is not open or that is greater than

		the number of files specified when BASICG was started up.
53	FF	File not found. Reference was made in a LOAD, KILL or OPEN statement to a file which did not exist on the diskette specified.
54	BM	Bad file mode. Program attempted to perform direct access on a file opened for sequential access or vice-versa.
55	AO	File already Open. An attempt was made to open a file that was already open. This error is also output if KILL, LOAD, SAVE, etc., is given for an open file.
56	IO	Disk I/O error. An error has been detected during a disk access.
57	FE	Undefined in Model II BASIC.
58	UE	Undefined error. Reserved for future use.
59	DF	Diskette full. All storage space on the diskette has been used. KILL unneeded files or use a formatted, non-full diskette.
6Ø	EF	End of file. An attempt was made to read past the end of file.
61	RN	Bad record number. In a PUT or GET statement, the record number is either greater than the allowable maximum, equal to zero, or negative.
62	NM	Bad file name.
63	MM	Mode mismatch. A sequential OPEN was executed for a file that already existed on the diskette as a direct access file, or vice versa.
64	UE	Undefined error. Reserved for future use.

65	DS	Direct statement. A direct statement was encountered during a load of a program in ASCII format. The load is terminated.
66	FL	Too many files.

Appendix C/ Subroutine Language Reference Summary

CIRCLE (radius,color,start,end,ar) Draws circle, ellipse, semi-circle, arc, or point. (x,y) coordinates set by SETXY.

```
CALL CIRCLE(1ØØ,1,Ø,Ø,Ø)
```

CLS (n) Clears Screen.

```
CALL CLS(2)
```

FVIEW (n) Returns viewport parameter.

```
I=FVIEW(Ø)
```

GET (array,size) Reads the contents of a rectangular pixel area into an array for future use by PUT.

```
CALL GET(A,4ØØØ)
```

GRPINI(option) Graphics initialization routine.

```
CALL GRPINI(Ø)
```

LINE (color,style) Draws a line. Coordinates set by SETXY or SETXYR.

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

LINEB (color,style) Draws a box. Coordinates set by SETXY or SETXYR.

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

LINEBF (color) Draws a filled box. Coordinates set by SETXY or SETXYR.

```
CALL LINEBF (1)
```

PAINT (color,border) Paints Screen.

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

PAINTT (arrayT,border,arrayS) Paints Screen with defined paint style.

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

POINT Returns pixel value at current coordinates.

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

PRESET (color) Sets pixel ON or OFF.

```
CALL PRESET(Ø)
```

PSET (<u>color</u>) Sets pixel ON or OFF.

```
     CALL PSET(Ø)
```

SCREEN (<u>n</u>) Sets Screen/graphics speed.

```
     CALL SCREEN(2)
```

SETXY(<u>X,Y</u>) Sets coordinates (absolute).

```
     CALL SETXY(1ØØ,1ØØ)
```

SETXYR(<u>X,Y</u>) Sets coordinates (relative).

```
     CALL SETXYR(5Ø,5Ø)
```

VIEW(<u>leftX</u>,<u>leftY</u>,<u>rightX</u>,<u>rightY</u>,<u>color</u>,<u>border</u>)
Sets viewport.

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

Appendix D/ Sample Programs

BASICG

```
1Ø '
2Ø ' Pie Graph Program ("PECANPIE/GRA")
3Ø '
4Ø ' Object
5Ø 'The object of this program is to draw a pie graph of the
6Ø 'expenses for a given month of eight departments of a
65 ' company,
7Ø ' along with the numerical value of each pie section
8Ø ' representation.
9Ø '
1ØØ '
11Ø ' Running the program
12Ø 'The month and the amounts spent by each department are
13Ø ' input, and the program takes over from there.
14Ø '
15Ø ' Special features
16Ø 'The amounts spent by each account as well as the total
17Ø 'amount spent are stored in strings.  The program will
18Ø 'standardize each string so that it is 9 characters long
19Ø 'and includes two characters to the right of the decimal
2ØØ 'point.  This allows for input of variable length and an
21Ø 'optional decimal point.
22Ø '
23Ø 'The various coordinates used in the program are found
24Ø ' based on the following equations:
25Ø '
26Ø 'x = r * cos(theta)
27Ø 'y = r * sin(theta)
28Ø '
29Ø 'where x and y are the coordinates, r is the radius,
295 'and theta is the angle.
3ØØ '(Note:  The y-coordinates are always multiplied
31Ø 'by Ø.5.  This is because the y pixels are twice the
315 'size of the x pixels.)
33Ø '
34Ø 'If an angle theta is generated by a percent less than
345' 1%, the section is not graphed, and the next theta is
35Ø' calculated.
36Ø 'However, the number will still be listed under the key.
37Ø '
```

```
38Ø ' Variables
39Ø 'ACCT$(i)Description of the account
4ØØ 'BUD$(i) Amount spent by the account
41Ø 'DS$ Dollar sign (used in output)
42Ø 'HXCOLColumn number for the pie section number
43Ø 'HYRW Row number for the pie section number
44Ø ' I Counter
45Ø ' MN$ Month
46Ø ' PER(i) Percent value of BUD$(i)
47Ø '    R Radius of circle
48Ø '    TØ Angle value line to be drawn
49Ø '    T1 Angle value of the next line
5ØØ '    TBUD$ Total of all the BUD$(i)'s
51Ø '    THALF Angle halfway between T1 and TØ (used for
52Ø '  location position for section number)
53Ø ' TILE$(i) Paint style for each section
54Ø ' TWOPI Two times the value of pi
55Ø ' XØ X-coordinate for drawing the line represented
56Ø ' by TØ
57Ø ' XP X-coordinate for painting a section
58Ø ' YØ Y-coordinate for drawing the line represented
59Ø ' by TØ
6ØØ ' YP Y-coordinate for painting a section
61Ø '
62Ø ' Set initial values
63Ø '
64Ø CLEAR 1ØØØ
65Ø DIM THALF(15),BUD$(15),ACCT$(15),PER(16)
66Ø TWOPI=2*3.14159
67Ø R=18Ø
68Ø DS$="$"
69Ø ACCT$(1) = "Sales"
7ØØ ACCT$(2) = "Purchasing"
71Ø ACCT$(3) = "R&D"
72Ø ACCT$(4) = "Accounting"
73Ø ACCT$(5) = "Construction"
74Ø ACCT$(5) = "Advertising"
75Ø ACCT$(6) = "Utilities"
76Ø ACCT$(7) = "Security"
77Ø ACCT$(8) = "Expansion"
78Ø TILE$(Ø)=CHR$(&H22)+CHR$(&HØØ)
79Ø TILE$(1)=CHR$(&HFF)+CHR$(&HØØ)
8ØØ TILE$(2)=CHR$(&H99)+CHR$(&H66)
81Ø TILE$(3)=CHR$(&H99)
82Ø TILE$(4)=CHR$(&HFF)
83Ø TILE$(5)=CHR$(&HFØ)+CHR$(&HFØ)+CHR$(&HØF)+CHR$(&HØF)
84Ø TILE$(6)=CHR$(&H3C)+CHR$(&H3C)+CHR$(&HFF)
85Ø TILE$(7)=CHR$(&HØ3)+CHR$(&HØC)+CHR$(&H3Ø)+CHR$(&HCØ)
86Ø '
```

```
87Ø ' Enter values to be graphed, standardize them, and
calculate
88Ø ' the percent they represent
89Ø '
9ØØ CLS2
91Ø PRINT @(1,Ø),"Enter month __________"
92Ø PRINT @(3,Ø),"Enter amount spent by"
93Ø PRINT @(4,Ø),"$__________"
94Ø PRINT @(Ø,Ø),""
95Ø LINE INPUT "Enter month ";MN$
96Ø FOR I=1 TO 8
97Ø PRINT @(3,22),ACCT$(I);"              "
98Ø PRINT @(4,Ø),"$__________"
99Ø PRINT @(3,Ø),""
1ØØØ LINE INPUT "$";BUD$(I)
1Ø1Ø IF INSTR(BUD$(I),".") = Ø THEN BUD$(I)=BUD$(I)+".ØØ"
1Ø2Ø IF LEN(BUD$(I))<9 THEN BUD$(I)=" "+BUD$(I):GOTO 1Ø2Ø
1Ø3Ø TBUD$=STR$(VAL(TBUD$)+VAL(BUD$(I)))
1Ø4Ø NEXT I
1Ø5Ø IF INSTR(TBUD$,".")=Ø THEN TBUD$=TBUD$+".ØØ"
1Ø6Ø IF LEN(TBUD$)<9 THEN TBUD$=" "+TBUD$:GOTO 1Ø6Ø
1Ø7Ø FOR I=1 TO 8
1Ø8Ø PER(I)=VAL(BUD$(I))/VAL(TBUD$)*1ØØ
1Ø9Ø NEXT I
11ØØ CLS2
111Ø '
112Ø ' Draw the circle and calculate the location of the
lines and
113Ø ' the line numbers
114Ø '
115Ø CIRCLE(41Ø,12Ø),R
116Ø FOR I=Ø TO 8
117Ø TØ=TWOPI/1ØØ*PER(I)+TØ
118Ø XØ=41Ø+R*COS(TØ)
119Ø YØ=12Ø-R*SIN(TØ)*Ø.5
12ØØ T1=TWOPI/1ØØ*PER(I+1)+TØ
121Ø THALF(I)=(TØ+T1)/2
122Ø HXCOL=(41Ø+R*1.15*COS(THALF(I)))*8Ø/64Ø
123Ø HYRW=(12Ø-R*1.15*SIN(THALF(I))*Ø.5)*24/24Ø
124Ø IF PER(I)>1 THEN LINE (41Ø,12Ø)-(XØ,YØ)
125Ø IF I<8 AND PER(I+1)>1 THEN PRINT @(HYRW,HXCOL),I+1
126Ø NEXT I
127Ø '
128Ø ' Paint the appropriate sections of the pie
129Ø '
13ØØ FOR I=Ø TO 7
131Ø XP=41Ø+R*Ø.5*COS(THALF(I))
132Ø YP=12Ø-R*Ø.5*SIN(THALF(I))*Ø.5
133Ø IF PER(I+1) >1 THEN PAINT (XP,YP),TILE$(I),1
```

```
134Ø NEXT I
135Ø '
136Ø ' Print the key for the graph
137Ø '
138Ø PRINT @(Ø,Ø),"Expenditures for"
139Ø PRINT @(1,Ø),MN$
14ØØ PRINT @(3,Ø),"#   Description   Amount"
141Ø FOR I=1 TO 8
142Ø PRINT @(4+I,Ø),I
143Ø PRINT @(4+I,4),ACCT$(I)
144Ø PRINT @(4+I,15),DS$;BUD$(I)
145Ø DS$=" "
146Ø NEXT I
147Ø PRINT STRING$(25,"_")
148Ø PRINT @(14,4),"Total"
149Ø PRINT @(14,16),TBUD$
15ØØ GOTO 15ØØ'Break to end program
```

```
1Ø '"THREEDEE/GRA"  (NOTE:  You must open BASICG with at
2Ø 'least one file, e.g. BASICG -F:1, in order to run this
3Ø 'program)
4Ø '
5Ø ' Object
6Ø '    The object of this program is to produce a three
7Ø ' dimensional bar graph representation of the gross
8Ø ' income for a company over a one year period.
9Ø '
1ØØ ' Variables
11Ø ' Vertical alphanumeric character
12Ø 'BMSG$ Bottom message
13Ø 'CHAR$ Disk file input field
14Ø 'GI$ Gross income
15Ø 'I Counter
16Ø 'J Counter
17Ø 'MN$ Month
18Ø 'REC Record number of vertical character
19Ø 'S1$ Single character of vertical message
2ØØ 'TILE$ Tile pattern for painting
21Ø 'TTINC Total income for the year
22Ø 'X X-coordinate of bar
23Ø 'Y(i) Y-coordinate of bar
24Ø '
25Ø 'Input/output
26Ø 'The program prompts you to enter the gross income, in
27Ø ' millions for each month.  The program requires these
275 ' values to be between one and nine.
28Ø 'Part of the output uses a data file called
285 '"VERTCHAR/DAT".
29Ø 'This file contains the dot-matrix pattern of the
3ØØ 'vertical character set.
31Ø '
32Ø 'Set initial values
33Ø '
34Ø CLS2
35Ø OPEN "D",1,"VERTCHAR/DAT",2
36Ø FIELD 1, 2 AS CHAR$
37Ø DIM Y(12),A(8),MN$(12)
38Ø DEFINT A
39Ø VMSG$=" Millions of dollars "
4ØØ TMSG$="G r o s s  I n c o m e  F o r  1 9 8 Ø "
41Ø BMSG$="M o n t h"
42Ø MN$(1)="January"
43Ø MN$(2)="February"
44Ø MN$(3)="March"
45Ø MN$(4)="April"
46Ø MN$(5)="May"
47Ø MN$(6)="June"
```

```
48Ø MN$(7)="July"
49Ø MN$(8)="August"
5ØØ MN$(9)="September"
51Ø MN$(1Ø)="October"
52Ø MN$(11)="November"
53Ø MN$(12)="December"
54Ø TILE$=CHR$(&H99)+CHR$(&H66)
55Ø X=-1Ø
56Ø '
57Ø 'Input gross income, and calculate the Y-coordinate
58Ø '
59Ø FOR I=1 TO 12
6ØØ CLS
61Ø PRINT "Enter gross income in millions (1-9) for ";MN$(I)
62Ø PRINT "$________"
63Ø PRINT @(Ø,Ø),""
64Ø LINE INPUT "$";GI$
65Ø Y(I)=2Ø5-2Ø*VAL(GI$)
66Ø TTINC=TTINC+VAL(GI$)
67Ø NEXT I
68Ø CLS2
69Ø '
7ØØ 'Draw the graph and bars
71Ø '
72Ø LINE (35,Ø)-(35,2Ø5)
73Ø LINE -(639,2Ø5)
74Ø FOR I=1 TO 12
75Ø CLS
76Ø X=X+5Ø
77Ø LINE (X,Y(I))-(X+2Ø,2Ø5),1,BF
78Ø LINE -(X+4Ø,195)
79Ø LINE -(X+4Ø,Y(I)-1Ø)
8ØØ LINE -(X+2Ø,Y(I)-1Ø)
81Ø LINE -(X,Y(I))
82Ø LINE (X+2Ø,Y(I))-(X+4Ø,Y(I)-1Ø)
83Ø PAINT(X+21,Y(I)+2),TILE$,1
84Ø NEXT I
85Ø '
86Ø 'Fetch the dot patterns for the vertical message from
87Ø '"VERTCHAR/DAT"
88Ø '
89Ø FOR J=2 TO LEN(VMSG$)-1
9ØØ S1$=MID$(VMSG$,J,1)
91Ø REC=(ASC(S1$)-1)*8+1
92Ø FOR I=Ø TO 7
93Ø GET 1,REC+I
94Ø A(I)=CVI(CHAR$)
95Ø NEXT I
96Ø PUT (Ø,14Ø-J*5),A
```

```
97Ø NEXT J
98Ø '
99Ø 'Print out the other display messages
1ØØØ '
1Ø1Ø PRINT @(21,5),"Jan   Feb   Mar    Apr    May    June
July   Aug   Sept  Oct   Nov   Dec"
1Ø2Ø PRINT @(22,36),BMSG$
1Ø3Ø FOR I=1 TO 1Ø
1Ø4Ø IF I>9 THEN C=1 ELSE C=2
1Ø5Ø PRINT @(2Ø-I*2,C),STR$(I);"-"
1Ø6Ø NEXT I
1Ø7Ø PRINT @(Ø,22),TMSG$
1Ø8Ø PRINT @(1,26),"(Total income is";TTINC;" million)"
1Ø9Ø CLOSE
11ØØ GOTO 11ØØ  'Break to end program
```

Printing Graphics Displays

There are many ways to use the stand-alone utilities (described in Graphic Utilities). The following discussion demonstrates one way to use the utilities with graphic displays generated under BASICG.

To print graphics, follow these steps:

1. When TRSDOS READY appears, set FORMS to FORMS P=66 L=6Ø W=Ø C=Ø. Then type: FORMS X <ENTER>. (See your Model II Owner's Manual).

2. Set the printer into Graphic Mode and set the printer's other parameters (elongation, non-elongated, etc.), if applicable, according to instructions in your printer owner's manual.

3. Write, run and save your program as a BASICG program file.

4. Transfer the contents of the video display to graphics memory using VDOGRPH.

5. Save the graphics memory to diskette using GSAVE.

6. Load the file into memory using GLOAD.

7. Enter the print command GPRINT.

Example #1:

1. Set FORMS and your printer's printing parameters.

2. Load BASICG and type in this program:

```
 1Ø DEFDBL Y
 2Ø CLS2
 3Ø LINE (Ø,12Ø)-(64Ø,12Ø)
 4Ø LINE (32Ø,Ø)-(32Ø,24Ø)
 5Ø FOR X=Ø TO 64Ø
 6Ø PI=3.141259
 7Ø X1=X/64Ø*2*PI-PI
 8Ø Y=SIN(X1)*1ØØ
 9Ø IF Y>1ØØ THEN X=X+7
1ØØ PSET (X,-Y+12Ø)
```

```
    11Ø NEXT X
    12Ø PRINT "THIS IS A SINE WAVE."
    13Ø SYSTEM"VDOGRPH"
```

3.RUN the program.

The program draws a sine wave on the Graphics Screen (graphics memory) and prints the statement in line 12Ø ("THIS IS A SINE WAVE.") on the Text Screen (video memory).

4.At the end of program execution, video memory is converted to graphics memory, as specified in program line 13Ø. The Text Screen is converted to graphics and then erased.

5.SINE (for sine wave) is the name we are giving this TRSDOS file. To save the contents of the graphics memory (which now includes the converted video memory) to diskette, type: SYSTEM "GSAVE SINE" <ENTER>

6.The graphics memory is saved as a TRSDOS file on your diskette.

7.Type: CLS2 <ENTER>

All video and graphics memory is now cleared.

8.To load the file back into memory, type:
SYSTEM "GLOAD SINE" <ENTER>

The display is now on the Graphics Screen.

1Ø. To print, type: SYSTEM "GPRINT" <ENTER>

Assembly Language Sample

The following is an assembler linker routine.

```
ØØ1ØØ              TITLE    HIGH RESOLUTION GRAPHICS TEST
ØØ2ØØ              SUBTTL   LINKAGE INFORMATION
ØØ3ØØ     ;
ØØ4ØØ              NAME     ('GTEST')
ØØ5ØØ              ENTRY    GTEST
ØØ6ØØ     ;
ØØ7ØØ              EXT      $INIT          ; FORTRAN INIT
ØØ8ØØ              EXT      CIRCLE         ; DRAW A CIRCLE
ØØ9ØØ              EXT      CLS            ; CLEAR SCREEN
Ø1ØØØ              EXT      GET            ; READ PIXELS INTO MEMORY
Ø11ØØ              EXT      GRPINI         ; GRAPHICS INIT
Ø12ØØ              EXT      LINE           ; DRAW A LINE
Ø13ØØ              EXT      LINEB          ; DRAW A BOX
Ø14ØØ              EXT      LINEBF         ; DRAW A FILLED BOX
Ø15ØØ              EXT      PAINT          ; PAINT SCREEN
Ø16ØØ              EXT      PAINTT         ; PAINT WITH A PATTERN
Ø17ØØ              EXT      PSET           ; SET/RESET PIXEL
Ø18ØØ              EXT      PRESET         ; SET/RESET PIXEL
Ø19ØØ              EXT      PUT            ; PUT MEMORY INTO PIXELS
Ø2ØØØ              EXT      SCREEN         ; SET SCREEN MODE
Ø21ØØ              EXT      SETXY          ; SET COORDINATES
Ø22ØØ              EXT      SETXYR         ; SET RELATIVE COORDINATES
Ø23ØØ              EXT      VIEW           ; DESIGNATE GRAPHICS AREAS
Ø24ØØ              EXT      POINT          ; RETURN PIXEL VALUE
Ø25ØØ              EXT      FVIEW          ; RETURN VIEWPORT PARAMETER
Ø26ØØ              EXT      $CA            ; CONVERT TO FLOATING POINT
Ø27ØØ              EXT      $AC            ; DATA RETURNED BY $CA
Ø28ØØ     ;
Ø29ØØ              SUBTTL   INITIALIZATION SECTION
Ø3ØØØ              PAGE
Ø31ØØ     ;
Ø32ØØ     ;        INITIALIZE FORTRAN UTILITIES
Ø33ØØ     ;
Ø34ØØ     GTEST:
Ø35ØØ              LD       BC,L1
Ø36ØØ              JP       $INIT
Ø37ØØ     ;
Ø38ØØ     ;        INITIALIZE GRAPHICS AND CLEAR GRAPHICS DISPLAY
Ø39ØØ     ;
Ø4ØØØ     L1:
Ø41ØØ              LD       HL,LOGØ
Ø42ØØ              CALL     GRPINI
```

```
Ø43ØØ    ;
Ø44ØØ    ;       SET BREAK KEY PROCESSING
Ø45ØØ    ;
Ø46ØØ            LD      HL,BREAK
Ø47ØØ            LD      A,3
Ø48ØØ            RST     8
Ø49ØØ    ;
Ø5ØØØ    ;       INITIALIZE I/O DRIVERS
Ø51ØØ    ;
Ø52ØØ            LD      A,Ø
Ø53ØØ            RST     8
Ø54ØØ    ;
Ø55ØØ    ;       INITIALIZE VIDEO
Ø56ØØ    ;
Ø57ØØ            LD      B,1
Ø58ØØ            LD      C,1
Ø59ØØ            LD      A,7
Ø6ØØØ            RST     8
Ø61ØØ    ;
Ø62ØØ            SUBTTL  CIRCLE, SETXY, AND PAINT TESTS
Ø63ØØ            PAGE
Ø64ØØ    ;
Ø65ØØ    ;       DISPLAY TEST MESSAGE
Ø66ØØ    ;
Ø67ØØ            LD      HL,MSG1
Ø68ØØ            LD      B,MSG2-MSG1
Ø69ØØ            LD      C,ØDH
Ø7ØØØ            LD      A,9
Ø71ØØ            RST     8
Ø72ØØ    ;
Ø73ØØ    ;       SET CENTER OF CIRCLE TO (3ØØ,1ØØ)
Ø74ØØ    ;
Ø75ØØ            LD      HL,D3ØØ
Ø76ØØ            LD      DE,D1ØØ
Ø77ØØ            CALL    SETXY
Ø78ØØ    ;
Ø79ØØ    ;       DRAW A CIRCLE OF RADIUS 1ØØ
Ø8ØØØ    ;
Ø81ØØ            LD      HL,FØ
Ø82ØØ            LD      (P3LIST),HL
Ø83ØØ            LD      (P3LIST+2),HL
Ø84ØØ            LD      (P3LIST+4),HL
Ø85ØØ            LD      HL,D1ØØ
Ø86ØØ            LD      DE,LOG1
Ø87ØØ            LD      BC,P3LIST
Ø88ØØ            CALL    CIRCLE
Ø89ØØ    ;
Ø9ØØØ    ;       PAINT THE CIRCLE
Ø91ØØ    ;
```

```
Ø92ØØ          LD      HL,LOG1
Ø93ØØ          LD      DE,LOG1
Ø94ØØ          CALL    PAINT
Ø95ØØ    ;
Ø96ØØ    ;     WAIT 5 SECONDS
Ø97ØØ    ;
Ø98ØØ          CALL    WAIT
Ø99ØØ    ;
1ØØØØ          SUBTTL  CIRCLE, CLS, GET, AND PUT TESTS
1Ø1ØØ          PAGE
1Ø2ØØ    ;
1Ø3ØØ    ;     CLEAR TEXT AND GRAPHICS
1Ø4ØØ    ;
1Ø5ØØ          LD      HL,LOG2
1Ø6ØØ          CALL    CLS
1Ø7ØØ    ;
1Ø8ØØ    ;     DISPLAY TEST MESSAGE
1Ø9ØØ    ;
11ØØØ          LD      HL,MSG2
111ØØ          LD      B,MSG3-MSG2
112ØØ          LD      C,ØDH
113ØØ          LD      A,9
114ØØ          RST     8
115ØØ    ;
116ØØ    ;     CONVERT TWO (2) TO FLOATING POINT
117ØØ    ;
118ØØ          LD      HL,2
119ØØ          CALL    $CA
12ØØØ          LD      HL,$AC
121ØØ          LD      BC,4
122ØØ          LD      DE,F2
123ØØ          LDIR
124ØØ    ;
125ØØ    ;     SET COORDINATES OF ELLIPSE
126ØØ    ;
127ØØ          LD      HL,D3ØØ
128ØØ          LD      DE,D1ØØ
129ØØ          CALL    SETXY
13ØØØ    ;
131ØØ    ;     DRAW ELLIPSE
132ØØ    ;
133ØØ          LD      HL,FØ
134ØØ          LD      BC,F2
135ØØ          LD      (P3LIST),HL
136ØØ          LD      (P3LIST+2),HL
137ØØ          LD      (P3LIST+4),BC
138ØØ          LD      HL,D2Ø
139ØØ          LD      DE,LOG1
14ØØØ          LD      BC,P3LIST
```

```
141ØØ              CALL    CIRCLE
142ØØ      ;
143ØØ      ;       SET COORDINATES FOR GET
144ØØ      ;
145ØØ              LD      HL,D26Ø
146ØØ              LD      DE,D6Ø
147ØØ              CALL    SETXY
148ØØ              LD      HL,D34Ø
149ØØ              LD      DE,D14Ø
15ØØØ              CALL    SETXY
151ØØ      ;
152ØØ      ;       STORE THE GRAPHICS
153ØØ      ;
154ØØ              LD      HL,STORE
155ØØ              LD      DE,D16ØØ
156ØØ              CALL    GET
157ØØ      ;
158ØØ      ;       WAIT 5 SECONDS AND CLEAR THE GRAPHICS
159ØØ      ;
16ØØØ              CALL    WAIT
161ØØ              LD      HL,LOG1
162ØØ              CALL    CLS
163ØØ      ;
164ØØ      ;       SET COORDINATES FOR PUT
165ØØ      ;
166ØØ              LD      HL,D1ØØ
167ØØ              LD      DE,D1ØØ
168ØØ              CALL    SETXY
169ØØ      ;
17ØØØ      ;       RESTORE ELLIPSE
171ØØ      ;
172ØØ              LD      HL,STORE
173ØØ              LD      DE,LOG1
174ØØ              CALL    PUT
175ØØ      ;
176ØØ      ;       CLEAR TEXT AND WAIT 5 SECONDS
177ØØ      ;
178ØØ              LD      HL,LOGØ
179ØØ              CALL    CLS
18ØØØ              CALL    WAIT
181ØØ      ;
182ØØ              SUBTTL  LINE, LINEB, LINEBF, AND SETXYR TESTS
183ØØ              PAGE
184ØØ      ;
185ØØ      ;       CLEAR SCREEN AND DISPLAY TEST MESSAGE
186ØØ      ;
187ØØ              LD      HL,LOG2
188ØØ              CALL    CLS
189ØØ              LD      HL,MSG3
```

```
19ØØØ              LD      B,MSG3A-MSG3
191ØØ              LD      C,ØDH
192ØØ              LD      A,9
193ØØ              RST     8
194ØØ              LD      HL,MSG3A
195ØØ              LD      B,MSG4-MSG3A
196ØØ              LD      C,ØDH
197ØØ              LD      A,9
198ØØ              RST     8
199ØØ      ;
2ØØØØ      ;       DRAW LINE
2Ø1ØØ      ;
2Ø2ØØ              LD      HL,D1
2Ø3ØØ              LD      DE,D1
2Ø4ØØ              CALL    SETXY
2Ø5ØØ              LD      HL,D21Ø
2Ø6ØØ              LD      DE,D8Ø
2Ø7ØØ              CALL    SETXY
2Ø8ØØ              LD      HL,LOG1
2Ø9ØØ              LD      DE,DM1
21ØØØ              CALL    LINE
211ØØ      ;
212ØØ      ;       DRAW BOX
213ØØ      ;
214ØØ              LD      HL,D21Ø
215ØØ              LD      DE,D8Ø
216ØØ              CALL    SETXYR
217ØØ              LD      HL,LOG1
218ØØ              LD      DE,DM1
219ØØ              CALL    LINEB
22ØØØ      ;
221ØØ      ;       DRAW FILLED IN BOX
222ØØ      ;
223ØØ              LD      HL,D639
224ØØ              LD      DE,D239
225ØØ              CALL    SETXY
226ØØ              LD      HL,LOG1
227ØØ              CALL    LINEBF
228ØØ      ;
229ØØ      ;       WAIT 5 SECONDS AND CLEAR THE SCREEN
23ØØØ      ;
231ØØ              CALL    WAIT
232ØØ              LD      HL,LOG2
233ØØ              CALL    CLS
234ØØ      ;
235ØØ              SUBTTL  PAINTT TEST
236ØØ              PAGE
237ØØ      ;
238ØØ      ;       DISPLAY TEST MESSAGE
```

```
239ØØ    ;
24ØØØ            LD       HL,MSG4
241ØØ            LD       B,MSG5-MSG4
242ØØ            LD       C,ØDH
243ØØ            LD       A,9
244ØØ            RST      8
245ØØ    ;
246ØØ    ;       DRAW AND PAINT CIRCLE
247ØØ    ;
248ØØ            LD       HL,D3ØØ
249ØØ            LD       DE,D1ØØ
25ØØØ            CALL     SETXY
251ØØ            LD       HL,FØ
252ØØ            LD       (P3LIST),HL
253ØØ            LD       (P3LIST+2),HL
254ØØ            LD       (P3LIST+4),HL
255ØØ            LD       HL,D15Ø
256ØØ            LD       DE,LOG1
257ØØ            LD       BC,P3LIST
258ØØ            CALL     CIRCLE
259ØØ            LD       HL,AARRAY
26ØØØ            LD       DE,LOG1
261ØØ            LD       BC,BARRAY
262ØØ            CALL     PAINTT
263ØØ    ;
264ØØ    ;       WAIT 5 SECONDS AND CLEAR SCREEN
265ØØ    ;
266ØØ            CALL     WAIT
267ØØ            LD       HL,LOG2
268ØØ            CALL     CLS
269ØØ    ;
27ØØØ            SUBTTL   PSET, PRESET, AND POINT TEST
271ØØ            PAGE
272ØØ    ;
273ØØ    ;       DISPLAY TEST MESSAGE
274ØØ    ;
275ØØ            LD       HL,MSG5
276ØØ            LD       B,MSG6-MSG5
277ØØ            LD       C,ØDH
278ØØ            LD       A,9
279ØØ            RST      8
28ØØØ    ;
281ØØ    ;       TURN PIXEL ON
282ØØ    ;
283ØØ            LD       HL,D3ØØ
284ØØ            LD       DE,D1ØØ
285ØØ            CALL     SETXY
286ØØ            LD       HL,LOG1
287ØØ            CALL     PSET
```

```
288ØØ                   CALL    POINT
289ØØ                   LD      C,A
29ØØØ                   LD      A,1
291ØØ                   CP      C
292ØØ                   JR      NZ,L2
293ØØ   ;
294ØØ   ;               TURN PIXEL OFF
295ØØ   ;
296ØØ                   LD      HL,LOGØ
297ØØ                   CALL    PRESET
298ØØ                   CALL    POINT
299ØØ                   LD      C,A
3ØØØØ                   XOR     A
3Ø1ØØ                   CP      C
3Ø2ØØ                   JR      NZ,L2
3Ø3ØØ   ;
3Ø4ØØ   ;               DISPLAY 'TEST PASSED'
3Ø5ØØ   ;
3Ø6ØØ                   LD      HL,MSG6
3Ø7ØØ                   LD      B,MSG7-MSG6
3Ø8ØØ                   LD      C,ØDH
3Ø9ØØ                   LD      A,9
31ØØØ                   RST     8
311ØØ                   JR      L3
312ØØ   ;
313ØØ   ;               DISPLAY 'TEST FAILED'
314ØØ   ;
315ØØ   L2:
316ØØ                   LD      HL,MSG7
317ØØ                   LD      B,MSG8-MSG7
318ØØ                   LD      C,ØDH
319ØØ                   LD      A,9
32ØØØ                   RST     8
321ØØ   ;
322ØØ   ;               WAIT 5 SECONDS AND CLEAR THE SCREEN
323ØØ   ;
324ØØ   L3:
325ØØ                   CALL    WAIT
326ØØ                   LD      HL,LOG2
327ØØ                   CALL    CLS
328ØØ   ;
329ØØ                   SUBTTL  SCREEN TEST
33ØØØ                   PAGE
331ØØ   ;
332ØØ   ;               DISPLAY TEST MESSAGE
333ØØ   ;
334ØØ                   LD      HL,MSG8
335ØØ                   LD      B,MSG9-MSG8
336ØØ                   LD      C,ØDH
```

```
337ØØ               LD      A,9
338ØØ               RST     8
339ØØ     ;
34ØØØ     ;         TURN OFF GRAPHICS AND DRAW A CIRCLE
341ØØ     ;
342ØØ               LD      HL,LOG1
343ØØ               CALL    SCREEN
344ØØ               LD      HL,D3ØØ
345ØØ               LD      DE,D1ØØ
346ØØ               CALL    SETXY
347ØØ               LD      HL,FØ
348ØØ               LD      (P3LIST),HL
349ØØ               LD      (P3LIST+2),HL
35ØØØ               LD      (P3LIST+4),HL
351ØØ               LD      HL,D1ØØ
352ØØ               LD      DE,LOG1
353ØØ               LD      BC,P3LIST
354ØØ               CALL    CIRCLE
355ØØ               LD      HL,LOG1
356ØØ               LD      DE,LOG1
357ØØ               CALL    PAINT
358ØØ     ;
359ØØ     ;         WAIT 5 SECONDS AND TURN GRAPHICS ON
36ØØØ     ;
361ØØ               CALL    WAIT
362ØØ               LD      HL,LOG2
363ØØ               CALL    SCREEN
364ØØ     ;
365ØØ     ;         WAIT 5 SECONDS,CLEAR SCREEN, AND TURN OFF FLASHING MODE
366ØØ     ;
367ØØ               CALL    WAIT
368ØØ               LD      HL,LOG2
369ØØ               CALL    CLS
37ØØØ               LD      HL,LOGØ
371ØØ               CALL    SCREEN
372ØØ     ;
373ØØ               SUBTTL  VIEW AND FVIEW TESTS
374ØØ               PAGE
375ØØ     ;
376ØØ     ;         DISPLAY TEST MESSAGE
377ØØ     ;
378ØØ               LD      HL,MSG9
379ØØ               LD      B,MSG1Ø-MSG9
38ØØØ               LD      C,ØDH
381ØØ               LD      A,9
382ØØ               RST     8
383ØØ     ;
384ØØ     ;         SET UP VIEW PORT
385ØØ     ;
```

```
386ØØ              LD       HL,D42Ø
387ØØ              LD       (P3LIST),HL
388ØØ              LD       HL,D16Ø
389ØØ              LD       (P3LIST+2),HL
39ØØØ              LD       HL,LOGØ
391ØØ              LD       (P3LIST+4),HL
392ØØ              LD       HL,LOG1
393ØØ              LD       (P3LIST+6),HL
394ØØ              LD       HL,D21Ø
395ØØ              LD       DE,D8Ø
396ØØ              LD       BC,P3LIST
397ØØ              CALL     VIEW
398ØØ       ;
399ØØ       ;      DRAW MULTIPLE CIRCLES
4ØØØØ       ;
4Ø1ØØ              LD       HL,D1Ø5
4Ø2ØØ              LD       DE,D4Ø
4Ø3ØØ              CALL     SETXY
4Ø4ØØ              LD       HL,1Ø
4Ø5ØØ       L4:
4Ø6ØØ              LD       (TEMP),HL
4Ø7ØØ              LD       HL,FØ
4Ø8ØØ              LD       (P3LIST),HL
4Ø9ØØ              LD       (P3LIST+2),HL
41ØØØ              LD       (P3LIST+4),HL
411ØØ              LD       HL,TEMP
412ØØ              LD       DE,LOG1
413ØØ              LD       BC,P3LIST
414ØØ              CALL     CIRCLE
415ØØ              LD       HL,(TEMP)
416ØØ              LD       BC,(D1Ø)
417ØØ              ADD      HL,BC
418ØØ              LD       A,15Ø
419ØØ              CP       L
42ØØØ              JR       NZ,L4
421ØØ       ;
422ØØ       ;      CHECK  FVIEW  VALUES
423ØØ       ;
424ØØ              LD       HL,LOGØ
425ØØ              CALL     FVIEW
426ØØ              LD       A,21Ø
427ØØ              CP       L
428ØØ              JR       NZ,L6
429ØØ              LD       HL,LOG1
43ØØØ              CALL     FVIEW
431ØØ              LD       A,8Ø
432ØØ              CP       L
433ØØ              JR       NZ,L6
434ØØ              LD       HL,LOG2
```

```
435ØØ            CALL     FVIEW
436ØØ            LD       A,ØA4H
437ØØ            CP       L
438ØØ            JR       NZ,L6
439ØØ            LD       A,1
44ØØØ            CP       H
441ØØ            JR       NZ,L6
442ØØ            LD       HL,LOG3
443ØØ            CALL     FVIEW
444ØØ            LD       A,16Ø
445ØØ            CP       L
446ØØ            JR       NZ,L6
447ØØ     ;
448ØØ     ;      DISPLAY 'FVIEW PASSED'
449ØØ     ;
45ØØØ            LD       HL,MSG11
451ØØ            LD       B,MSG12-MSG11
452ØØ            LD       C,ØDH
453ØØ            LD       A,9
454ØØ            RST      8
455ØØ            JR       L7
456ØØ     ;
457ØØ     ;      DISPLAY 'FVIEW FAILED'
458ØØ     ;
459ØØ     L6:
46ØØØ            LD       HL,MSG1Ø
461ØØ            LD       B,MSG11-MSG1Ø
462ØØ            LD       C,ØDH
463ØØ            LD       A,9
464ØØ            RST      8
465ØØ     ;
466ØØ     ;      CHANGE VIEW PORTS AND DISPLAY DATA
467ØØ     ;
468ØØ     L7:
469ØØ            CALL     WAIT
47ØØØ            LD       HL,D41Ø
471ØØ            LD       (P3LIST),HL
472ØØ            LD       HL,D15Ø
473ØØ            LD       (P3LIST+2),HL
474ØØ            LD       HL,LOGØ
475ØØ            LD       (P3LIST+4),HL
476ØØ            LD       HL,LOG1
477ØØ            LD       (P3LIST+6),HL
478ØØ            LD       HL,D22Ø
479ØØ            LD       DE,D9Ø
48ØØØ            LD       BC,P3LIST
481ØØ            CALL     VIEW
482ØØ            LD       HL,D1
483ØØ            LD       DE,D1
```

```
484ØØ              CALL    SETXY
485ØØ              LD      HL,D1ØØ
486ØØ              LD      DE,D1ØØ
487ØØ              CALL    SETXY
488ØØ              LD      HL,LOG1
489ØØ              LD      DE,DM1
49ØØØ              CALL    LINE
491ØØ              CALL    WAIT
492ØØ              LD      HL,D4ØØ
493ØØ              LD      (P3LIST),HL
494ØØ              LD      HL,D14Ø
495ØØ              LD      (P3LIST+2),HL
496ØØ              LD      HL,LOGØ
497ØØ              LD      (P3LIST+4),HL
498ØØ              LD      HL,LOG1
499ØØ              LD      (P3LIST+6),HL
5ØØØØ              LD      HL,D23Ø
5Ø1ØØ              LD      DE,D1ØØ
5Ø2ØØ              LD      BC,P3LIST
5Ø3ØØ              CALL    VIEW
5Ø4ØØ              LD      HL,D8Ø
5Ø5ØØ              LD      DE,D2Ø
5Ø6ØØ              CALL    SETXY
5Ø7ØØ              LD      HL,FØ
5Ø8ØØ              LD      (P3LIST),HL
5Ø9ØØ              LD      (P3LIST+2),HL
51ØØØ              LD      (P3LIST+4),HL
511ØØ              LD      HL,D15
512ØØ              LD      DE,LOG1
513ØØ              LD      BC,P3LIST
514ØØ              CALL    CIRCLE
515ØØ              LD      HL,LOG1
516ØØ              LD      DE,LOG1
517ØØ              CALL    PAINT
518ØØ       ;
519ØØ       ;      SCROLL 12 LINES AND CLEAR SCREEN
52ØØØ       ;
521ØØ              CALL    WAIT
522ØØ              LD      HL,MSG12
523ØØ              LD      B,MSG13-MSG12
524ØØ              LD      C,ØDH
525ØØ              LD      A,9
526ØØ              RST     8
527ØØ              CALL    WAIT
528ØØ              LD      HL,D639
529ØØ              LD      (P3LIST),HL
53ØØØ              LD      HL,D239
531ØØ              LD      (P3LIST+2),HL
532ØØ              LD      HL,LOGØ
```

```
533ØØ               LD      (P3LIST+4),HL
534ØØ               LD      HL,LOG1
535ØØ               LD      (P3LIST+6),HL
536ØØ               LD      HL,DØ
537ØØ               LD      DE,DØ
538ØØ               LD      BC,P3LIST
539ØØ               CALL    VIEW
54ØØØ               LD      HL,LOG2
541ØØ               CALL    CLS
542ØØ     ;
543ØØ               SUBTTL PIE DRAWING TEST
544ØØ               PAGE
545ØØ     ;
546ØØ     ;         CONVERT 1, 3, 4, -1, -2, -3, -4 TO FLOATING POINT
547ØØ     ;
548ØØ               LD      HL,1
549ØØ               CALL    $CA
55ØØØ               LD      HL,$AC
551ØØ               LD      DE,F1
552ØØ               LD      BC,4
553ØØ               LDIR
554ØØ               LD      HL,3
555ØØ               CALL    $CA
556ØØ               LD      HL,$AC
557ØØ               LD      DE,F3
558ØØ               LD      BC,4
559ØØ               LDIR
56ØØØ               LD      HL,4
561ØØ               CALL    $CA
562ØØ               LD      HL,$AC
563ØØ               LD      DE,F4
564ØØ               LD      BC,4
565ØØ               LDIR
566ØØ               LD      HL,-1
567ØØ               CALL    $CA
568ØØ               LD      HL,$AC
569ØØ               LD      DE,FM1
57ØØØ               LD      BC,4
571ØØ               LDIR
572ØØ               LD      HL,-2
573ØØ               CALL    $CA
574ØØ               LD      HL,$AC
575ØØ               LD      DE,FM2
576ØØ               LD      BC,4
577ØØ               LDIR
578ØØ               LD      HL,-3
579ØØ               CALL    $CA
58ØØØ               LD      HL,$AC
581ØØ               LD      DE,FM3
```

```
582ØØ             LD       BC,4
583ØØ             LDIR
584ØØ             LD       HL,-4
585ØØ             CALL     $CA
586ØØ             LD       HL,$AC
587ØØ             LD       DE,FM4
588ØØ             LD       BC,4
589ØØ             LDIR
59ØØØ     ;
591ØØ     ;       DISPLAY  TEST MESSAGE
592ØØ     ;
593ØØ             LD       HL,MSG13
594ØØ             LD       B,MSG14-MSG13
595ØØ             LD       C,ØDH
596ØØ             LD       A,9
597ØØ             RST      8
598ØØ     ;
599ØØ     ;       DRAW PIE
6ØØØØ     ;
6Ø1ØØ             LD       HL,D3ØØ
6Ø2ØØ             LD       DE,D1ØØ
6Ø3ØØ             CALL     SETXY
6Ø4ØØ             LD       HL,FM1
6Ø5ØØ             LD       (P3LIST),HL
6Ø6ØØ             LD       HL,FM2
6Ø7ØØ             LD       (P3LIST+2),HL
6Ø8ØØ             LD       HL,FØ
6Ø9ØØ             LD       (P3LIST+4),HL
61ØØØ             LD       HL,D1ØØ
611ØØ             LD       DE,LOG1
612ØØ             LD       BC,P3LIST
613ØØ             CALL     CIRCLE
614ØØ             LD       HL,D3ØØ
615ØØ             LD       DE,D95
616ØØ             CALL     SETXY
617ØØ             LD       HL,LOG1
618ØØ             LD       DE,LOG1
619ØØ             CALL     PAINT
62ØØØ             LD       HL,D3ØØ
621ØØ             LD       DE,D1ØØ
622ØØ             CALL     SETXY
623ØØ             LD       HL,F2
624ØØ             LD       (P3LIST),HL
625ØØ             LD       HL,FM3
626ØØ             LD       (P3LIST+2),HL
627ØØ             LD       HL,FØ
628ØØ             LD       (P3LIST+4),HL
629ØØ             LD       HL,D1ØØ
63ØØØ             LD       DE,LOG1
```

```
631ØØ             LD      BC,P3LIST
632ØØ             CALL    CIRCLE
633ØØ             LD      HL,F3
634ØØ             LD      (P3LIST),HL
635ØØ             LD      HL,F4
636ØØ             LD      (P3LIST+2),HL
637ØØ             LD      HL,FØ
638ØØ             LD      (P3LIST+4),HL
639ØØ             LD      HL,D1ØØ
64ØØØ             LD      DE,LOG1
641ØØ             LD      BC,P3LIST
642ØØ             CALL    CIRCLE
643ØØ             LD      HL,FM4
644ØØ             LD      (P3LIST),HL
645ØØ             LD      HL,FØ
646ØØ             LD      (P3LIST+2),HL
647ØØ             LD      (P3LIST+4),HL
648ØØ             LD      HL,D1ØØ
649ØØ             LD      DE,LOG1
65ØØØ             LD      BC,P3LIST
651ØØ             CALL    CIRCLE
652ØØ             LD      HL,FØ
653ØØ             LD      (P3LIST),HL
654ØØ             LD      (P3LIST+4),HL
655ØØ             LD      HL,F1
656ØØ             LD      (P3LIST+2),HL
657ØØ             LD      HL,D1ØØ
658ØØ             LD      DE,LOG1
659ØØ             LD      BC,P3LIST
66ØØØ             CALL    CIRCLE
661ØØ             LD      HL,D29Ø
662ØØ             LD      DE,D1ØØ
663ØØ             CALL    SETXY
664ØØ             LD      HL,LOG1
665ØØ             LD      DE,LOG1
666ØØ             CALL    PAINT
667ØØ             CALL    WAIT
668ØØ             LD      HL,LOG2
669ØØ             CALL    CLS
67ØØØ     ;
671ØØ             SUBTTL  RETURN TO TRSDOS
672ØØ             PAGE
673ØØ     BREAK:
674ØØ             LD      A,36
675ØØ             RST     8
676ØØ     ;
677ØØ             SUBTTL  WAIT FOR 5 SECONDS
678ØØ             PAGE
679ØØ     WAIT:
```

```
68ØØØ                LD       HL,Ø
681ØØ      L5:
682ØØ                LD       (TEMP),HL
683ØØ                LD       BC,Ø
684ØØ                LD       A,8
685ØØ                RST      8
686ØØ                LD       HL,(TEMP)
687ØØ                INC      HL
688ØØ                LD       A,(D1ØØ)
689ØØ                CP       H
69ØØØ                JR       NZ,L5
691ØØ                RET
692ØØ      ;
693ØØ                SUBTTL   LOCAL DATA
694ØØ                PAGE
695ØØ      MSG1:     DB       'DRAW A CIRCLE - SETXY, CIRCLE, PAINT TESTS'
696ØØ      MSG2:     DB       'DRAW, SAVE, AND RESTORE AN ELLIPSE - CLS, '
697ØØ                DB       'CIRCLE, GET, PUT TESTS'
698ØØ      MSG3:     DB       'DRAW A LINE CONNECTED TO A BOX CONNECTED TO'
699ØØ                DB       ' A FILLED BOX'
7ØØØØ      MSG3A:    DB       'LINE, LINEB, LINEBF, SETXYR TESTS'
7Ø1ØØ      MSG4:     DB       'PAINT A CIRCLE WITH TILES - PAINTT TEST'
7Ø2ØØ      MSG5:     DB       'PSET, PRESET, AND POINT TESTS'
7Ø3ØØ      MSG6:     DB       'TEST PASSED'
7Ø4ØØ      MSG7:     DB       'TEST FAILED'
7Ø5ØØ      MSG8:     DB       'TURN OFF GRAPHICS, DRAW A CIRCLE, THEN TURN '
7Ø6ØØ                DB       'ON GRAPHICS - SCREEN'
7Ø7ØØ      MSG9:     DB       'VIEW AND FVIEW TESTS'
7Ø8ØØ      MSG1Ø:    DB       'FVIEW FAILED'
7Ø9ØØ      MSG11:    DB       'FVIEW PASSED'
71ØØØ      MSG12:    DB       ØDH,ØDH,ØDH,ØDH,ØDH,ØDH,ØDH,ØDH,ØDH,ØDH,ØDH,ØDH
711ØØ      MSG13:    DB       'PIE DRAWING TEST'
712ØØ      MSG14     EQU      $
713ØØ      D1Ø5:     DW       1Ø5
714ØØ      D4Ø:      DW       4Ø
715ØØ      D1Ø:      DW       1Ø
716ØØ      D21Ø:     DW       21Ø
717ØØ      LOG3:     DB       3
718ØØ      TEMP:     DS       2
719ØØ      D1:       DW       1
72ØØØ      D34Ø:     DW       34Ø
721ØØ      D26Ø:     DW       26Ø
722ØØ      D14Ø:     DW       14Ø
723ØØ      D6Ø:      DW       6Ø
724ØØ      D2Ø:      DW       2Ø
725ØØ      P3LIST:   DS       8
726ØØ      D1ØØ:     DW       1ØØ
727ØØ      D3ØØ:     DW       3ØØ
728ØØ      FØ:       DW       Ø,Ø
```

```
72900     LOG1:    DB       1
73000     LOG2:    DB       2
73100     STORE:   DS       1600
73200     F2:      DS       4
73300     D1600:   DW       1600
73400     LOG0:    DB       0
73500     D80:     DW       80
73600     DM1:     DW       -1
73700     D420:    DW       420
73800     D160:    DW       160
73900     D639:    DW       639
74000     D239:    DW       239
74100     AARRAY:  DB       8,81H,42H,24H,18H,18H,24H,42H,81H
74200     BARRAY:  DB       1,0
74300     D150:    DW       150
74400     D0:      DW       0
74500     D410:    DW       410
74600     D400:    DW       400
74700     D220:    DW       220
74800     D230:    DW       230
74900     D90:     DW       90
75000     D15:     DW       15
75100     D290:    DW       290
75200     D95:     DW       95
75300     F1:      DS       4
75400     F3:      DS       4
75500     F4:      DS       4
75600     FM1:     DS       4
75700     FM2:     DS       4
75800     FM3:     DS       4
75900     FM4:     DS       4
76000     ;
76100              SUBTTL   MACROS AND SYMBOLS
76200              END      GTEST
```

COBOL Sample Program

```
000100 IDENTIFICATION DIVISION.
000110 PROGRAM-ID.
000120     GRAFIX.
000130
000140 ENVIRONMENT DIVISION.
000150 CONFIGURATION SECTION.
000160 SOURCE-COMPUTER.  TRS-80-MODEL-II.
000170 OBJECT-COMPUTER.  TRS-80-MODEL-II-64K-HIGH-RES-GRAPHICS.
000180
000190 DATA DIVISION.
000200 WORKING-STORAGE SECTION.
000210     COPY "CBLGRAPH/CPY".
000220 01  GET-BUFFER.
000230*   BUFFER SIZE = 96 X PIXELS / 8 BY 31 Y PIXELS + 4 BYTES
000240     02  FILLER  PIC XXXX.
000250     02  STORAGE PIC X(12) OCCURS 31 TIMES.
000260
000270 PROCEDURE DIVISION.
000280 DRAW-CAR.
000290     CALL GRAPH-SUB USING GRAPHICS-PARAMETERS.
000300     CALL GRAPH-SUB USING GRPINI-CMD.
000310     MOVE 2 TO CLEAR-KEY.
000320     CALL GRAPH-SUB USING CLS-CMD.
000330*
000340     MOVE 50 TO Y-COORD, X-COORD.
000350     CALL GRAPH-SUB USING SETXY-CMD.
000360     MOVE 10 TO RADIUS.
000370     MOVE 0 TO START-CIR, END-CIR, RATIO-CIR.
000380     MOVE 1 TO COLOR.
000390     CALL GRAPH-SUB USING CIRCLE-CMD.
000400*
000410     MOVE 0 TO Y-COORD.
000420     CALL GRAPH-SUB USING SETXYR-CMD.
000430     CALL GRAPH-SUB USING CIRCLE-CMD.
000440*
000450     MOVE -10 TO X-COORD.
000460     CALL GRAPH-SUB USING SETXYR-CMD.
000470     MOVE -30 TO X-COORD.
000480     CALL GRAPH-SUB USING SETXYR-CMD.
000490     MOVE -1 TO STYLE.
000500     CALL GRAPH-SUB USING LINE-CMD.
000510*
000520     CALL GRAPH-SUB USING SETXYR-CMD.
000530     MOVE 10 TO X-COORD.
```

```
ØØØ54Ø      CALL GRAPH-SUB USING SETXYR-CMD.
ØØØ55Ø      CALL GRAPH-SUB USING LINE-CMD.
ØØØ56Ø*
ØØØ57Ø      MOVE 7Ø TO X-COORD.
ØØØ58Ø      CALL GRAPH-SUB USING SETXYR-CMD.
ØØØ59Ø      MOVE 1Ø TO X-COORD.
ØØØ6ØØ      CALL GRAPH-SUB USING SETXYR-CMD.
ØØØ61Ø      CALL GRAPH-SUB USING LINE-CMD.
ØØØ62Ø*
ØØØ63Ø      MOVE -45 TO X-COORD.
ØØØ64Ø      CALL GRAPH-SUB USING SETXYR-CMD.
ØØØ65Ø      MOVE 45 TO RADIUS.
ØØØ66Ø      MOVE 3.142 TO END-CIR.
ØØØ67Ø      CALL GRAPH-SUB USING CIRCLE-CMD.
ØØØ68Ø*
ØØØ69Ø      MOVE Ø TO X-COORD.
ØØØ7ØØ      MOVE -8 TO Y-COORD.
ØØØ71Ø      CALL GRAPH-SUB USING SETXYR-CMD.
ØØØ72Ø      MOVE 25 TO RADIUS.
ØØØ73Ø      MOVE -Ø.ØØ1 TO START-CIR.
ØØØ74Ø      MOVE -3.14 TO END-CIR.
ØØØ75Ø      MOVE Ø.4 TO RATIO-CIR.
ØØØ76Ø      CALL GRAPH-SUB USING CIRCLE-CMD.
ØØØ77Ø*
ØØØ78Ø GET-CAR.
ØØØ79Ø      MOVE 376 TO GET-SIZE.
ØØØ8ØØ      CALL GRAPH-SUB USING GPBUF-CMD.
ØØØ81Ø      CALL GRAPH-SUB USING GET-BUFFER.
ØØØ82Ø*
ØØØ83Ø      MOVE 25 TO X-COORD, Y-COORD.
ØØØ84Ø      CALL GRAPH-SUB USING SETXY-CMD.
ØØØ85Ø      MOVE 95 TO X-COORD.
ØØØ86Ø      MOVE 3Ø TO Y-COORD.
ØØØ87Ø      CALL GRAPH-SUB USING SETXYR-CMD.
ØØØ88Ø      CALL GRAPH-SUB USING GET-CMD.
ØØØ89Ø*
ØØØ9ØØ MOVE-CAR.
ØØØ91Ø      MOVE 25 TO X-COORD, Y-COORD.
ØØØ92Ø      CALL GRAPH-SUB USING SETXY-CMD.
ØØØ93Ø      MOVE 1 TO X-COORD.
ØØØ94Ø      MOVE Ø TO Y-COORD.
ØØØ95Ø      MOVE 4 TO ACTION.
ØØØ96Ø      PERFORM PUT-CAR 5ØØ TIMES.
ØØØ97Ø      GO TO ALL-DONE.
ØØØ98Ø PUT-CAR.
ØØØ99Ø      CALL GRAPH-SUB USING SETXYR-CMD.
ØØ1ØØØ      CALL GRAPH-SUB USING PUT-CMD.
ØØ1Ø1Ø ALL-DONE.
ØØ1Ø2Ø      EXIT PROGRAM.
ØØ1Ø3Ø END PROGRAM.
```

FORTRAN Sample Programs

```
ØØ1ØØ      C       HIGH RESOLUTION GRAPHICS TEST - MAIN PROGRAM
ØØ2ØØ      C
ØØ3ØØ              CALL GRPINI(Ø)
ØØ4ØØ      C
ØØ5ØØ      C       CIRCLE TEST
ØØ6ØØ      C
ØØ7ØØ              CALL CTEST
ØØ8ØØ      C
ØØ9ØØ      C       LINE TEST
Ø1ØØØ      C
Ø11ØØ              CALL LTEST
Ø12ØØ      C
Ø13ØØ      C       LINEB TEST
Ø14ØØ      C
Ø15ØØ              CALL LBTST
Ø16ØØ      C
Ø17ØØ      C       LINEBF TEST
Ø18ØØ      C
Ø19ØØ              CALL LBFTST
Ø2ØØØ      C
Ø21ØØ      C       PAINTT TEST
Ø22ØØ      C
Ø23ØØ              CALL PTTTST
Ø24ØØ      C
Ø25ØØ      C       GET AND PUT TEST
Ø26ØØ      C
Ø27ØØ              CALL GPTST
Ø28ØØ      C
Ø29ØØ      C       PSET/POINT TEST
Ø3ØØØ      C
Ø31ØØ              CALL PPTST
Ø32ØØ      C
Ø33ØØ      C       PRESET/POINT TEST
Ø34ØØ      C
Ø35ØØ              CALL PRETST
Ø36ØØ      C
Ø37ØØ      C       SCREEN TEST
Ø38ØØ      C
Ø39ØØ              CALL SCRTST
Ø4ØØØ      C
Ø41ØØ      C       VIEW/FVIEW TEST
Ø42ØØ      C
Ø43ØØ              CALL VTEST
Ø44ØØ              CALL CLS(2)
Ø45ØØ              END
```

```
                SUBROUTINE CTEST
        C
        C       THIS SUBROUTINE TESTS CIRCLE, SETXY, AND PAINT
        C
                CALL CLS(2)
                WRITE (3,1ØØ)
        1ØØ     FORMAT('2TEST CIRCLE, SETXY, AND PAINT')
                CALL WAIT
                DO 1Ø I=1,1ØØ
                IX=IRAND(639)
                IY=IRAND(239)
                IR=IRAND(15Ø)
                START=IRAND(12)
                START=START-6.Ø
                END=IRAND(12)
                END=END-6.Ø
                IF (START.LT.END) GOTO 1
                T=START
                START=END
                END=T
        1       CONTINUE
                RATIO=IRAND(1ØØØ)
                IF (RATIO.GT.Ø) RATIO=RATIO/4Ø.
                CALL SETXY(IX,IY)
                CALL CIRCLE(IR,1,START,END,RATIO)
        1Ø      CONTINUE
        C
        C       RANDOMLY FILL IN THE AREAS
        C
                DO 11 I=1,5Ø
                IX=IRAND(639)
                IY=IRAND(239)
                CALL SETXY(IX,IY)
                CALL PAINT(1,1)
        11      CONTINUE
                CALL WAIT
                RETURN
                END
```

```
ØØ1ØØ           SUBROUTINE LTEST
ØØ2ØØ     C
ØØ3ØØ     C     THIS ROUTINE EXERCISES LINE
ØØ4ØØ     C
ØØ5ØØ           CALL CLS(2)
ØØ6ØØ           WRITE(3,1ØØ)
ØØ7ØØ     1ØØ   FORMAT('2LINE AND PAINT TEST')
ØØ8ØØ           CALL WAIT
ØØ9ØØ           J=1ØØ
Ø1ØØØ           DO 1Ø I=1,639,2
Ø11ØØ           CALL SETXY(I,15)
Ø12ØØ           CALL SETXY(I,239)
Ø13ØØ           CALL LINE(1,J)
Ø14ØØ           J=J-1
Ø15ØØ     1Ø    CONTINUE
Ø16ØØ           CALL WAIT
Ø17ØØ           CALL CLS(1)
Ø18ØØ     C
Ø19ØØ     C     DRAW WHITE LINES AND FILL IN RANDOMLY
Ø2ØØØ     C
Ø21ØØ           IX=IRAND(639)
Ø22ØØ           IY=IRAND(2Ø9)+3Ø
Ø23ØØ           CALL SETXY(IX,IY)
Ø24ØØ           DO 11 I=1,1ØØ
Ø25ØØ           IX=IRAND(639)
Ø26ØØ           IY=IRAND(2Ø9)+3Ø
Ø27ØØ           CALL SETXY(IX,IY)
Ø28ØØ           CALL LINE(1,-1)
Ø29ØØ     11    CONTINUE
Ø3ØØØ           DO 12 I=1,5Ø
Ø31ØØ           IX=IRAND(639)
Ø32ØØ           IY=IRAND(2Ø9)+3Ø
Ø33ØØ           CALL SETXY(IX,IY)
Ø34ØØ           CALL PAINT(1,Ø)
Ø35ØØ     12    CONTINUE
Ø36ØØ           CALL WAIT
Ø37ØØ           CALL CLS(1)
Ø38ØØ     C
Ø39ØØ     C     WHITE OUT SCREEN,DRAW BLACK LINES, PAINT
Ø391Ø     C     BLACK RANDOMLY
Ø4ØØØ     C
Ø41ØØ           CALL SETXY(Ø,3Ø)
Ø42ØØ           CALL SETXY(639,3Ø)
Ø43ØØ           CALL LINE(1,-1)
Ø44ØØ           CALL SETXY(1ØØ,1ØØ)
Ø45ØØ           CALL PAINT(1,1)
Ø46ØØ           DO 15 I=1,1ØØ
Ø47ØØ           IX=IRAND(639)
Ø48ØØ           IY=IRAND(2Ø9)+3Ø
```

```
Ø49ØØ               CALL SETXY(IX,IY)
Ø5ØØØ               CALL LINE(Ø,-1)
Ø51ØØ       15      CONTINUE
Ø52ØØ               DO 16 I=1,5Ø
Ø53ØØ               IX=IRAND(639)
Ø54ØØ               IY=IRAND(2Ø9)+3Ø
Ø55ØØ               CALL SETXY(IX,IY)
Ø56ØØ               CALL PAINT(Ø,Ø)
Ø57ØØ       16      CONTINUE
Ø58ØØ               CALL WAIT
Ø59ØØ               RETURN
Ø6ØØØ               END
ØØ1ØØ               SUBROUTINE LBTST
ØØ2ØØ       C
ØØ3ØØ       C       LINEB TEST
ØØ4ØØ       C
ØØ5ØØ               CALL CLS(2)
ØØ6ØØ               WRITE (3,1ØØ)
ØØ7ØØ       1ØØ     FORMAT('2LINEB TEST')
ØØ8ØØ               CALL WAIT
ØØ9ØØ               ISTYL=2Ø
Ø1ØØØ               IXP=639
Ø11ØØ               DO 1Ø IX=Ø,1ØØ,3
Ø12ØØ               CALL SETXY(IX,IX+3Ø)
Ø13ØØ               CALL SETXY(IXP,IXP-4ØØ)
Ø14ØØ               CALL LINEB(1,ISTYL)
Ø15ØØ               ISTYL=ISTYL-1
Ø16ØØ               IXP=IXP-3
Ø17ØØ       1Ø      CONTINUE
Ø18ØØ               CALL CLS(Ø)
Ø19ØØ               CALL WAIT
Ø2ØØØ       C
Ø21ØØ       C       WHITE OUT SCREEN AND DRAW BLACK BOXES
Ø22ØØ       C
Ø23ØØ               CALL CLS(2)
Ø24ØØ               CALL PAINT(1,1)
Ø25ØØ               ISTYL=2Ø
Ø26ØØ               IXP=639
Ø27ØØ               DO 11 IX=Ø,11Ø,3
Ø28ØØ               CALL SETXY(IX,IX)
Ø29ØØ               CALL SETXY(IXP,IXP-4ØØ)
Ø3ØØØ               CALL LINEB(Ø,ISTYL)
Ø31ØØ               ISTYL=ISTYL-1
Ø32ØØ               IXP=IXP-3
Ø33ØØ       11      CONTINUE
Ø34ØØ               CALL WAIT
Ø35ØØ               RETURN
Ø36ØØ               END
```

```
ØØ1ØØ             SUBROUTINE LBFTST
ØØ2ØØ     C
ØØ3ØØ     C       LINEBF TEST
ØØ4ØØ     C
ØØ5ØØ             CALL CLS(2)
ØØ6ØØ             WRITE (3,1ØØ)
ØØ7ØØ     1ØØ     FORMAT('2LINEBF TEST')
ØØ8ØØ             CALL WAIT
ØØ9ØØ             IXP=639
Ø1ØØØ             ICLR=1
Ø11ØØ             DO 1Ø IX=Ø,12Ø
Ø12ØØ             CALL SETXY(IX,IX+3Ø)
Ø13ØØ             CALL SETXY(IXP,IXP-4ØØ)
Ø14ØØ             CALL LINEBF(ICLR)
Ø15ØØ             IXP=IXP-3
Ø16ØØ             ICLR=ICLR-1
Ø17ØØ             IF (ICLR.LT.Ø) ICLR=1
Ø18ØØ     1Ø      CONTINUE
Ø19ØØ             CALL WAIT
Ø2ØØØ             RETURN
Ø21ØØ             END
```

```
ØØ1ØØ                SUBROUTINE PTTTST
ØØ2ØØ      C
ØØ3ØØ      C         PAINT WITH TILES TEST
ØØ4ØØ      C
ØØ5ØØ                LOGICAL A(65),B(4),IS(16)
ØØ6ØØ                DATA A(1)/8/
ØØ7ØØ      C         X
ØØ8ØØ                DATA A(2),A(3),A(4),A(5)/X'41',X'22',X'14',X'Ø8'/
ØØ9ØØ                DATA A(6),A(7),A(8),A(9)/X'14',X'22',X'41',X'ØØ'/
Ø1ØØØ      C         FINE HORIZONTAL LINES
Ø11ØØ                DATA A(1Ø),A(11),A(12)/2,X'FF',X'ØØ'/
Ø12ØØ      C         MEDIUM HORIZONTAL LINES
Ø13ØØ                DATA A(13)/4/
Ø14ØØ                DATA A(14),A(15),A(16),A(17)/X'FF',X'FF',X'ØØ',X'ØØ'/
Ø15ØØ      C         DIAGONAL LINES
Ø16ØØ                DATA A(18)/4/
Ø17ØØ                DATA A(19),A(2Ø),A(21),A(22)/X'Ø3',X'ØC',X'3Ø',X'CØ'/
Ø18ØØ      C         LEFT TO RIGHT DIAGONALS
Ø19ØØ                DATA A(23)/4/
Ø2ØØØ                DATA A(24),A(25),A(26),A(27)/X'CØ',X'3Ø',X'ØC',X'Ø3'/
Ø21ØØ      C         FINE VERTICAL LINES
Ø22ØØ                DATA A(28),A(29)/1,X'AA'/
Ø23ØØ      C         MEDIUM VERTICAL LINES
Ø24ØØ                DATA A(3Ø),A(31)/1,X'CC'/
Ø25ØØ      C         COARSE VERTICAL LINES
Ø26ØØ                DATA A(32),A(33)/1,X'FØ'/
Ø27ØØ      C         ONE PIXEL DOTS
Ø28ØØ                DATA A(34),A(35),A(36)/2,X'22',X'ØØ'/
Ø29ØØ      C         TWO PIXEL DOTS
Ø3ØØØ                DATA A(37),A(38),A(39)/2,X'99',X'66'/
Ø31ØØ      C         PLUSES
Ø32ØØ                DATA A(4Ø),A(41),A(42),A(43)/3,X'3C',X'3C',X'FF'/
Ø33ØØ      C         SOLID
Ø34ØØ                DATA A(44),A(45)/1,X'FF'/
Ø35ØØ      C         BROAD CROSS HATCH
Ø36ØØ                DATA A(46),A(47),A(48),A(49)/3,X'92',X'92',X'FF'/
Ø37ØØ      C         THICK CROSS HATCH
Ø38ØØ                DATA A(5Ø)/4/
Ø39ØØ                DATA A(51),A(52),A(53),A(54)/X'FF',X'FF',X'DB',X'DB'/
Ø4ØØØ      C         FINE CROSS HATCH
Ø41ØØ                DATA A(54),A(55),A(56)/2,X'92',X'FF'/
Ø42ØØ      C         ALTERNATING PIXELS
Ø43ØØ                DATA A(57),A(58),A(59)/2,X'55',X'AA'/
Ø44ØØ                DATA B(1),B(2),B(3),B(4)/1,Ø,1,X'FF'/
Ø45ØØ                DATA IS(1),IS(2),IS(3),IS(4),IS(5),IS(6)/1,1Ø,13,18,
Ø455Ø               123,28/
Ø46ØØ                DATA IS(7),IS(8),IS(9),IS(1Ø),IS(11)/3Ø,32,34,37,4Ø/
Ø47ØØ                DATA IS(12),IS(13),IS(14),IS(15),IS(16)/44,46,5Ø,54,57/
Ø48ØØ                CALL CLS(2)
```

```
Ø49ØØ           WRITE(3,1ØØ)
Ø5ØØØ   1ØØ     FORMAT('2PAINTT AND SETXYR TESTS')
Ø51ØØ           CALL WAIT
Ø52ØØ   C
Ø53ØØ   C       PAINT ON A BLACK BACKGROUND
Ø54ØØ   C
Ø55ØØ           DO 1Ø I=1,16
Ø56ØØ           CALL SETXY(Ø,4Ø)
Ø57ØØ           CALL SETXYR(639,199)
Ø58ØØ           CALL LINEB(1,-1)
Ø59ØØ           CALL SETXYR(-3ØØ,-1ØØ)
Ø6ØØØ           ITMP=IS(I)
Ø61ØØ           CALL PAINTT(A(ITMP),1,B)
Ø62ØØ           CALL WAIT
Ø63ØØ           CALL CLS(1)
Ø64ØØ   1Ø      CONTINUE
Ø65ØØ   C
Ø66ØØ   C       PAINT ON A WHITE BACKGROUND
Ø67ØØ   C
Ø68ØØ           DO 11 I=1,16
Ø69ØØ           IF(I.EQ.12) GOTO 11
Ø7ØØØ           CALL CLS(1)
Ø71ØØ           CALL SETXY(Ø,4Ø)
Ø72ØØ           CALL SETXYR(639,199)
Ø73ØØ           CALL LINEBF(1)
Ø74ØØ           CALL SETXYR(-3ØØ,-1ØØ)
Ø75ØØ           ITMP=IS(I)
Ø76ØØ           CALL PAINTT(A(ITMP),Ø,B(3))
Ø77ØØ           CALL WAIT
Ø78ØØ   11      CONTINUE
Ø79ØØ           RETURN
Ø8ØØØ           END
```

```
ØØ1ØØ             SUBROUTINE GPTST
ØØ2ØØ      C
ØØ3ØØ      C      GET AND PUT TEST
ØØ4ØØ      C
ØØ5ØØ             LOGICAL A(1ØØØ)
ØØ6ØØ             CALL CLS(Ø)
ØØ7ØØ             WRITE (3,1ØØ)
ØØ8ØØ      1ØØ    FORMAT('2GET AND PUT TEST')
ØØ9ØØ             CALL SETXY(1ØØ,1ØØ)
Ø1ØØØ             CALL SETXYR(3Ø,3Ø)
Ø11ØØ             CALL LINEBF(1)
Ø12ØØ             CALL GET(A,1ØØØ)
Ø13ØØ             CALL CLS(1)
Ø14ØØ             CALL WAIT
Ø15ØØ             CALL SETXY(1ØØ,1ØØ)
Ø16ØØ             CALL PUT(A,1)
Ø17ØØ             CALL WAIT
Ø18ØØ             RETURN
Ø19ØØ             END
```

```
        SUBROUTINE PPTST
C
C       PSET AND POINT TEST
C
        CALL CLS(2)
        WRITE(3,1ØØ)
1ØØ     FORMAT('2PSET AND POINT TEST')
        CALL WAIT
        CALL CLS(2)
C
C       SET AND CHECK ALL PIXELS
C
        DO 1Ø I=Ø,639
        DO 11 J=Ø,239
        CALL SETXY(I,J)
        CALL PSET(1)
        K=POINT(L)
        IF(K.EQ.Ø) GOTO 999
11      CONTINUE
1Ø      CONTINUE
C
C       RESET AND CHECK ALL PIXELS
C
        DO 12 I=Ø,639
        DO 13 J=Ø,239
        CALL SETXY(I,J)
        CALL PSET(Ø)
        K=POINT(L)
        IF (K.EQ.1) GOTO 999
13      CONTINUE
12      CONTINUE
        CALL CLS(2)
        WRITE(3,1Ø1)
1Ø1     FORMAT('2PSET AND POINT PASSED')
        GOTO 1ØØØ
999     CALL CLS(2)
        WRITE(3,1Ø2)
1Ø2     FORMAT('2PSET AND POINT FAILED')
1ØØØ    CALL WAIT
        RETURN
        END
        SUBROUTINE PRETST
C
C       PRESET AND POINT TEST
C
        CALL CLS(2)
        WRITE(3,1ØØ)
1ØØ     FORMAT('2PRESET AND POINT TEST')
        CALL WAIT
        CALL CLS(2)
C
C       SET AND CHECK ALL PIXELS
C
        DO 1Ø I=Ø,639
        DO 11 J=Ø,239
        CALL SETXY(I,J)
        CALL PRESET(1)
        K=POINT(L)
        IF(K.EQ.Ø) GOTO 999
11      CONTINUE
1Ø      CONTINUE
C
C       RESET AND CHECK ALL PIXELS
C
        DO 12 I=Ø,639
        DO 13 J=Ø,239
        CALL SETXY(I,J)
        CALL PRESET(Ø)
        K=POINT(L)
        IF (K.EQ.1) GOTO 999
13      CONTINUE
12      CONTINUE
        CALL CLS(2)
        WRITE(3,1Ø1)
1Ø1     FORMAT('2PRESET AND POINT PASSED')
        GOTO 1ØØØ
999     CALL CLS(2)
        WRITE(3,1Ø2)
1Ø2     FORMAT('2PRESET AND POINT FAILED')
1ØØØ    CALL WAIT
        RETURN
        END
```

```
ØØ1ØØ           SUBROUTINE SCRTST
ØØ2ØØ     C
ØØ3ØØ     C     SCREEN TEST
ØØ4ØØ     C
ØØ5ØØ           CALL CLS(2)
ØØ6ØØ           WRITE(3,1ØØ)
ØØ7ØØ     1ØØ   FORMAT('2SCREEN TEST')
ØØ8ØØ           CALL WAIT
ØØ9ØØ           CALL SETXY(3ØØ,12Ø)
Ø1ØØØ           CALL CIRCLE(1ØØ,1,Ø.Ø,6.28,Ø.5)
Ø11ØØ           CALL CIRCLE(1ØØ,1,Ø.Ø,6.28,Ø.25)
Ø12ØØ           CALL CIRCLE(5Ø,1,Ø.Ø,6.28,Ø.5)
Ø13ØØ           CALL PAINT(1,1)
Ø14ØØ     C
Ø15ØØ     C     GRAPHICS BUT NOT FLASHING
Ø16ØØ     C
Ø17ØØ           CALL SCREEN(Ø)
Ø18ØØ           CALL WAIT
Ø19ØØ           CALL WAIT
Ø2ØØØ           CALL WAIT
Ø21ØØ     C
Ø22ØØ     C     NEITHER GRAPHICS NOR FLASHING
Ø23ØØ     C
Ø24ØØ           CALL SCREEN(1)
Ø25ØØ           CALL WAIT
Ø26ØØ           CALL WAIT
Ø27ØØ           CALL WAIT
Ø28ØØ     C
Ø29ØØ     C     GRAPHICS AND FLASHING
Ø3ØØØ     C
Ø31ØØ           CALL SCREEN(2)
Ø32ØØ           CALL WAIT
Ø33ØØ           CALL WAIT
Ø34ØØ           CALL WAIT
Ø35ØØ     C
Ø36ØØ     C     FLASHING BUT NOT GRAPHICS
Ø37ØØ     C
Ø38ØØ           CALL SCREEN(3)
Ø39ØØ           CALL WAIT
Ø4ØØØ           CALL WAIT
Ø41ØØ           CALL WAIT
Ø42ØØ     C
Ø43ØØ     C     RETURN TO NORMAL SCREEN
Ø44ØØ     C
Ø45ØØ           CALL SCREEN(2)
Ø46ØØ           RETURN
Ø47ØØ           END
```

```
ØØ1ØØ            SUBROUTINE VTEST
ØØ2ØØ   C
ØØ3ØØ   C        VIEW AND FVIEW TEST
ØØ4ØØ   C
ØØ5ØØ            INTEGER FVIEW
ØØ6ØØ            CALL CLS(2)
ØØ7ØØ            WRITE(3,1ØØ)
ØØ8ØØ   1ØØ      FORMAT('2VIEW AND FVIEW TEST')
ØØ9ØØ            CALL WAIT
Ø1ØØØ   C
Ø11ØØ   C        TURN OFF FLASHING MODE
Ø12ØØ   C
Ø13ØØ            CALL SCREEN(Ø)
Ø14ØØ   C
Ø15ØØ   C        DRAW VIEWPORT AND CIRCLES
Ø16ØØ   C
Ø17ØØ            CALL VIEW(Ø,4Ø,639,239,Ø,1)
Ø18ØØ            CALL DCIRCL(1)
Ø19ØØ   C
Ø2ØØØ   C        DRAW VIEWPORT AND LINES
Ø21ØØ   C
Ø22ØØ            CALL VIEW(2Ø,5Ø,619,229,1,Ø)
Ø23ØØ            CALL DLINE(Ø)
Ø24ØØ   C
Ø25ØØ   C        DRAW VIEWPORT AND CIRCLES
Ø26ØØ   C
Ø27ØØ            CALL VIEW(4Ø,6Ø,599,2Ø9,Ø,Ø)
Ø28ØØ            CALL DCIRCL(1)
Ø29ØØ   C
Ø3ØØØ   C        DRAW VIEWPORT AND LINES
Ø31ØØ   C
Ø32ØØ            CALL VIEW(6Ø,7Ø,579,199,1,1)
Ø33ØØ            CALL DLINE(Ø)
Ø34ØØ   C
Ø35ØØ   C        CLEAR SCREEN
Ø36ØØ   C
Ø37ØØ            IX1=FVIEW(Ø)
Ø38ØØ            IY1=FVIEW(1)
Ø39ØØ            IX2=FVIEW(2)
Ø4ØØØ            IY2=FVIEW(3)
Ø41ØØ            CALL VIEW(6Ø-IX1,7Ø-IY1,6Ø+IX2,4Ø+IY2,Ø,1)
Ø42ØØ            CALL CLS(2)
Ø43ØØ            RETURN
Ø44ØØ            END
```

```
ØØ45ØØ           SUBROUTINE DCIRCL(ICLR)
Ø46ØØ            CALL SETXY(1ØØ,1ØØ)
Ø47ØØ            DO 1Ø I=5,3ØØ,5
Ø48ØØ            CALL CIRCLE(I,ICLR,Ø.Ø,6.28,Ø.5)
Ø49ØØ    1Ø      CONTINUE
Ø5ØØØ            CALL WAIT
Ø51ØØ            RETURN
Ø52ØØ            END
Ø53ØØ            SUBROUTINE DLINE(ICLR)
Ø54ØØ            DO 11 I=2,2ØØ,4
Ø55ØØ            CALL SETXY(-1Ø,-1Ø)
Ø56ØØ            CALL SETXY(I+2ØØ,I)
Ø57ØØ            CALL LINE(ICLR,-1)
Ø58ØØ    11      CONTINUE
Ø59ØØ            CALL WAIT
Ø6ØØØ            RETURN
Ø61ØØ            END
```

```
ØØ1ØØ          SUBROUTINE WAIT
ØØ2ØØ   C
ØØ3ØØ   C      THIS SUBROUTINE INTRODUCES A TIME DELAY
ØØ4ØØ   C
ØØ5ØØ          DO 11 J=1,2Ø
ØØ6ØØ          DO 1Ø I=1,1ØØØØ
ØØ7ØØ   1Ø     CONTINUE
ØØ8ØØ   11     CONTINUE
ØØ9ØØ          RETURN
Ø1ØØØ          END
```

```
ØØ1ØØ                   TITLE   INTEGER RANDOM NUMBER GENERATOR
ØØ2ØØ           ;
ØØ3ØØ                   NAME    ('IRAND')
ØØ4ØØ                   ENTRY   IRAND
ØØ5ØØ           ;
ØØ6ØØ           IRAND:
ØØ7ØØ                   PUSH    AF                ; SAVE REGISTERS
ØØ8ØØ                   PUSH    BC
ØØ9ØØ                   PUSH    IX
Ø1ØØØ                   PUSH    HL
Ø11ØØ                   POP     IX
Ø12ØØ                   LD      B,(HL)
Ø13ØØ                   INC     B
Ø14ØØ                   XOR     A
Ø15ØØ                   CP      B
Ø16ØØ                   JR      NZ,L1
Ø17ØØ                   LD      B,ØFFH
Ø18ØØ           L1:
Ø19ØØ                   LD      A,2Ø
Ø2ØØØ                   RST     8                 ; RANDOM NUM FOR LOW
Ø21ØØ                   LD      L,C               ;  ORDER BITS IN L
Ø22ØØ                   LD      B,(IX+1)
Ø23ØØ                   INC     B
Ø24ØØ                   LD      A,2Ø
Ø25ØØ                   RST     8                 ; RANDOM NUM FOR HIGH
Ø26ØØ                   LD      H,C               ;  ORDER BITS IN H
Ø27ØØ                   POP     IX
Ø28ØØ                   POP     BC
Ø29ØØ                   POP     AF
Ø3ØØØ                   RET
Ø31ØØ                   END
```

DEC.	HEX.	BINARY	DEC.	HEX.	BINARY
80	50	01010000	120	78	01111000
81	51	01010001	121	79	01111001
82	52	01010010	122	7A	01111010
83	53	01010011	123	7B	01111011
84	54	01010100	124	7C	01111100
85	55	01010101	125	7D	01111101
86	56	01010110	126	7E	01111110
87	57	01010111	127	7F	01111111
88	58	01011000	128	80	10000000
89	59	01011001	129	81	10000001
90	5A	01011010	130	82	10000010
91	5B	01011011	131	83	10000011
92	5C	01011100	132	84	10000100
93	5D	01011101	133	85	10000101
94	5E	01011110	134	86	10000110
95	5F	01011111	135	87	10000111
96	60	01100000	136	88	10001000
97	61	01100001	137	89	10001001
98	62	01100010	138	8A	10001010
99	63	01100011	139	8B	10001011
100	64	01100100	140	8C	10001100
101	65	01100101	141	8D	10001101
102	66	01100110	142	8E	10001110
103	67	01100111	143	8F	10001111
104	68	01101000	144	90	10010000
105	69	01101001	145	91	10010001
106	6A	01101010	146	92	10010010
107	6B	01101011	147	93	10010011
108	6C	01101100	148	94	10010100
109	6D	01101101	149	95	10010101
110	6E	01101110	150	96	10010110
111	6F	01101111	151	97	10010111
112	70	01110000	152	98	10011000
113	71	01110001	153	99	10011001
114	72	01110010	154	9A	10011010
115	73	01110011	155	9B	10011011
116	74	01110100	156	9C	10011100
117	75	01110101	157	9D	10011101
118	76	01110110	158	9E	10011110
119	77	01110111	159	9F	10011111

Appendix E/ Base Conversion Chart

DEC.	HEX.	BINARY	DEC.	HEX.	BINARY
0	00	00000000	40	28	00101000
1	01	00000001	41	29	00101001
2	02	00000010	42	2A	00101010
3	03	00000011	43	2B	00101011
4	04	00000100	44	2C	00101100
5	05	00000101	45	2D	00101101
6	06	00000110	46	2E	00101110
7	07	00000111	47	2F	00101111
8	08	00001000	48	30	00110000
9	09	00001001	49	31	00110001
10	0A	00001010	50	32	00110010
11	0B	00001011	51	33	00110011
12	0C	00001100	52	34	00110100
13	0D	00001101	53	35	00110101
14	0E	00001110	54	36	00110110
15	0F	00001111	55	37	00110111
16	10	00010000	56	38	00111000
17	11	00010001	57	39	00111001
18	12	00010010	58	3A	00111010
19	13	00010011	59	3B	00111011
20	14	00010100	60	3C	00111100
21	15	00010101	61	3D	00111101
22	16	00010110	62	3E	00111110
23	17	00010111	63	3F	00111111
24	18	00011000	64	40	01000000
25	19	00011001	65	41	01000001
26	1A	00011010	66	42	01000010
27	1B	00011011	67	43	01000011
28	1C	00011100	68	44	01000100
29	1D	00011101	69	45	01000101
30	1E	00011110	70	46	01000110
31	1F	00011111	71	47	01000111
32	20	00100000	72	48	01001000
33	21	00100001	73	49	01001001
34	22	00100010	74	4A	01001010
35	23	00100011	75	4B	01001011
36	24	00100100	76	4C	01001100
37	25	00100101	77	4D	01001101
38	26	00100110	78	4E	01001110
39	27	00100111	79	4F	01001111

DEC.	HEX.	BINARY	DEC.	HEX.	BINARY
160	A0	10100000	200	C8	11001000
161	A1	10100001	201	C9	11001001
162	A2	10100010	202	CA	11001010
163	A3	10100011	203	CB	11001011
164	A4	10100100	204	CC	11001100
165	A5	10100101	205	CD	11001101
166	A6	10100110	206	CE	11001110
167	A7	10100111	207	CF	11001111
168	A8	10101000	208	D0	11010000
169	A9	10101001	209	D1	11010001
170	AA	10101010	210	D2	11010010
171	AB	10101011	211	D3	11010011
172	AC	10101100	212	D4	11010100
173	AD	10101101	213	D5	11010101
174	AE	10101110	214	D6	11010110
175	AF	10101111	215	D7	11010111
176	B0	10110000	216	D8	11011000
177	B1	10110001	217	D9	11011001
178	B2	10110010	218	DA	11011010
179	B3	10110011	219	DB	11011011
180	B4	10110100	220	DC	11011100
181	B5	10110101	221	DD	11011101
182	B6	10110110	222	DE	11011110
183	B7	10110111	223	DF	11011111
184	B8	10111000	224	E0	11100000
185	B9	10111001	225	E1	11100001
186	BA	10111010	226	E2	11100010
187	BB	10111011	227	E3	11100011
188	BC	10111100	228	E4	11100100
189	BD	10111101	229	E5	11100101
190	BE	10111110	230	E6	11100110
191	BF	10111111	231	E7	11100111
192	C0	11000000	232	E8	11101000
193	C1	11000001	233	E9	11101001
194	C2	11000010	234	EA	11101010
195	C3	11000011	235	EB	11101011
196	C4	11000100	236	EC	11101100
197	C5	11000101	237	ED	11101101
198	C6	11000110	238	EE	11101110
199	C7	11000111	239	EF	11101111

DEC.	HEX.	BINARY
240	FØ	1111ØØØØ
241	F1	1111ØØØ1
242	F2	1111ØØ1Ø
243	F3	1111ØØ11
244	F4	1111Ø1ØØ
245	F5	1111Ø1Ø1
246	F6	1111Ø11Ø
247	F7	1111Ø111
248	F8	11111ØØØ
249	F9	11111ØØ1
25Ø	FA	11111Ø1Ø
251	FB	11111Ø11
252	FC	111111ØØ
253	FD	111111Ø1
254	FE	1111111Ø
255	FF	11111111

Appendix F/ Pixel Grid Reference

The following hexadecimal numbers include commonly used tiling designs.

Important Note: You cannot use more than two empty rows of tiles when tiling or you'll get an Illegal Function Call error.

Example (four rows of empty tiles):

```
CHR$(&HFF)+CHR$(&HFF)+CHR$(&HØØ)+CHR$(&HØØ)+CHR$(&HØØ)+CHR$(&HØØ)
```

gives you a Function Call error.

1. "X"

```
CHR$(&H41)+CHR$(&H22)+CHR$(&H14)+CHR$(&HØ8)+CHR$(&H14)
+CHR$(&H22)+CHR$(&H41)+CHR$(&HØØ)
```

								Hex	Decimal
Ø	1	Ø	Ø	Ø	Ø	Ø	1	41	65
Ø	Ø	1	Ø	Ø	Ø	1	Ø	22	34
Ø	Ø	Ø	1	Ø	1	Ø	Ø	14	2Ø
Ø	Ø	Ø	Ø	1	Ø	Ø	Ø	Ø8	8
Ø	Ø	Ø	1	Ø	1	Ø	Ø	14	2Ø
Ø	Ø	1	Ø	Ø	Ø	1	Ø	22	34
Ø	1	Ø	Ø	Ø	Ø	Ø	1	41	65
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	ØØ	Ø

2. "Fine" horizontal lines

```
CHR$(&HFF)+CHR$(&HØØ)
```

								Hex	Decimal
1	1	1	1	1	1	1	1	FF	255
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	ØØ	Ø

3. "Medium" horizontal lines

```
CHR$(&HFF)+CHR$(&HFF)+CHR$(&HØØ)+CHR$(&HØØ)
```

								Hex	Decimal
1	1	1	1	1	1	1	1	FF	255
1	1	1	1	1	1	1	1	FF	255
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	ØØ	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	ØØ	Ø

4. Diagonal lines

(Right to left):
CHR$(&HØ3)+CHR$(&HØC)+CHR$(&H3Ø)+CHR$(&HCØ)

								Hex	Decimal
Ø	Ø	Ø	Ø	Ø	Ø	1	1	Ø3	3
Ø	Ø	Ø	Ø	1	1	Ø	Ø	ØC	12
Ø	Ø	1	1	Ø	Ø	Ø	Ø	3Ø	48
1	1	Ø	Ø	Ø	Ø	Ø	Ø	CØ	192

(Left to right)
CHR$(&HCØ)+CHR$(&H3Ø)+CHR$(&HØC)+CHR$(&HØ3)

								Hex	Decimal
1	1	Ø	Ø	Ø	Ø	Ø	Ø	CØ	192
Ø	Ø	1	1	Ø	Ø	Ø	Ø	3Ø	48
Ø	Ø	Ø	Ø	1	1	Ø	Ø	ØC	12
Ø	Ø	Ø	Ø	Ø	Ø	1	1	Ø3	3

5. "Fine" vertical lines

CHR$(&HAA)

								Hex	Decimal
1	Ø	1	Ø	1	Ø	1	Ø	AA	17Ø

6. "Medium" vertical lines

CHR$(&HCC)

								Hex	Decimal
1	1	Ø	Ø	1	1	Ø	Ø	CC	2Ø4

7. "Coarse" vertical lines

CHR$(&HFØ)

								Hex	Decimal
1	1	1	1	Ø	Ø	Ø	Ø	FØ	24Ø

8. One-pixel dots

CHR$(&H22)+CHR$(&HØØ)

								Hex	Decimal
Ø	Ø	1	Ø	Ø	Ø	1	Ø	22	34
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	ØØ	Ø

9. Two-pixel dots

CHR$(&H99)+CHR$(&H66)

								Hex	Decimal
1	Ø	Ø	1	1	Ø	Ø	1	99	153
Ø	1	1	Ø	Ø	1	1	Ø	66	1Ø2

1Ø. Pluses ("+")

CHR$(&H3C)+CHR$(&H3C)+CHR$(&HFF)

								Hex	Decimal
Ø	Ø	1	1	1	1	Ø	Ø	3C	6Ø
Ø	Ø	1	1	1	1	Ø	Ø	3C	6Ø
1	1	1	1	1	1	1	1	FF	255

11. Solid (all pixels ON)

CHR$(&HFF)

								Hex	Decimal
1	1	1	1	1	1	1	1	FF	255

12. "Broad" cross-hatch

CHR$(&H92)+CHR$(&H92)+CHR$(&HFF)

								Hex	Decimal
1	Ø	Ø	1	Ø	Ø	1	Ø	92	146
1	Ø	Ø	1	Ø	Ø	1	Ø	92	146
1	1	1	1	1	1	1	1	FF	255

13. "Thick" cross-hatch

CHR$(&HFF)+CHR$(&HFF)+CHR$(&HDB)+CHR$(&HDB)

								Hex	Decimal
1	1	1	1	1	1	1	1	FF	255
1	1	1	1	1	1	1	1	FF	255
1	1	Ø	1	1	Ø	1	1	DB	219
1	1	Ø	1	1	Ø	1	1	DB	219

14. "Fine" cross-hatch

CHR$(&H92)+CHR$(&HFF)

								Hex	Decimal
1	Ø	Ø	1	Ø	Ø	1	Ø	92	146
1	1	1	1	1	1	1	1	FF	255

15. Alternating pixels

CHR$(&H55)+CHR$(&HAA)

								Hex	Decimal
Ø	1	Ø	1	Ø	1	Ø	1	55	85
1	Ø	1	Ø	1	Ø	1	Ø	AA	17Ø

Appendix G/ Line Style Reference

type	binary numbers	hex	decimal
long dash	ØØØØ ØØØØ 1111 1111	&HØØFF	255
short dash	ØØØØ 1111 ØØØØ 1111	&HFØFØ	-3856
"short-short" dash	1111 ØØØØ 1111 ØØØØ	&HCCCC	-131Ø8
solid line	1111 1111 1111 1111	&HFFFF	-1
OFF/ON	Ø1Ø1 Ø1Ø1 Ø1Ø1 Ø1Ø1	&H5555	21845
"wide" dots	ØØØØ 1ØØØ ØØØØ 1ØØØ	&HØ8Ø8	2Ø56
"medium" dots	1ØØØ 1ØØØ 1ØØØ 1ØØØ	&H8888	-3Ø584
"dot-dash"	1ØØØ 1111 1111 1ØØØ	&H8FF8	-2868Ø

Index

Index

Index

Index

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.
280-316 VICTORIA ROAD RYDALMERE, N.S.W. 2116	PARC INDUSTRIEL DE NANINNE 5140 NANINNE	BILSTON ROAD WEDNESBURY WEST MIDLANDS WS10 7JN

8749327 **PRINTED IN U.S.A.**