Twenty Dollars

Cromemco

SDI

Graphics Software

Instruction Manual

List of Housekeeping and Graphics Calls

The following is an alphabetical listing of all calls available using the SDI graphics package. The graphics calls are indicated by an 'i' for the implicit calls and an 'e' for the explicit calls. Since the BASIC form of the text calls require additional programming steps, they are denoted by an '*' in front of the BASIC format to call your attention to this fact.

	FORTRAN Assembly SBASIC+SDILIB	BASIC SBASIC w/o SDILIB
	AFILL	X=USR (316,66)
	AINIT(p)	X=USR (316,72)
	ANIM	none
	ANIMAT	X=USR (316,73)
i	AREA(x,y)	X=USR (316,45)
	CLIP	X=USR (316,51)
	COLR(c)	X=USR (316,32,c)
	CURSOR(x,y)	X=USR (316,33,x,y)
	DEFCLR(c,R,G,B)	X=USR (316,84,c,R,G,B)
	DISP(p)	X=USR (316,70,p)
i	DOT	X=USR (316,37)
	ERABOX(xl,yl,x2,y2)	X=USR (316,82,xl,yl,x2,y2)
	ESTRNG("p")	none
	FILINT	X=USR (316,65)
i	FSEG(x,y)	X=USR (316,31,x,y)
	GRAFIX	none
i	HAREA(x,y)	X=USR (316,46,x,y)
	HCRSOR(x,y)	X=USR (316,34,x,y)
i	HDOT	X=USR (316,38)
i	HLINE(x,y)	X=USR (316,36,x,y)
i	HRLINE(x,y)	X=USR (316,20,x,y)
	HSCALE(xl,x2,yl,y2)	X=USR (316,54,xl,x2,yl,y2)
i	HTEXT("text ^")	* X=USR (316,48,T1)
e	HXAREA(xl,yl,x2,y2)	X=USR (316,5,xl,yl,x2,y2)
e	HXCIRC(x,y,r)	X=USR (316,7,x,y,r)
e	HXDOT(x,y)	X=USR (316,3,x,y)
e	HXFCIR(x,y,r)	X=USR (316,9,x,y,r)
e	HXLINE(xl,yl,x2,y2)	X=USR (316,1,xl,yl,x2,y2)
e	HXPOLY(a)	X=USR (316,17,a)
e	HXREAD(x,y,V)	none
e	HXTEXT(x,y,"text ^")	* X=USR (316,13,T1)
	INIT	X=USR (316,68)
	INIT1	none
i	LINE(x,y)	X=USR (316,35,x,y)
	PAGE	X=USR (316,71)
i	READ(x,y,V)	none
	RES(r)	X=USR (316,80,r)
	RESBOX(xl,yl,x2,y2)	X=USR (316,81,xl,yl,x2,y2)
i	RLINE(x,y)	X=USR (316,19,x,y)
	SCALE(xl,x2,yl,y2)	X=USR (316,49,xl,x2,yl,y2)
	SCROFF	X=USR (316,64)
	SCRON	X=USR (316,63)
i	TEXT("text ^")	* X=USR (316,47,T1)
	UNCLIP	X=USR (316,52)
	UNSCAL	X=USR (316,50)
	WAITHG	X=USR (316,86)
	WAITOD	X=USR (316,88)
	WAITVG	X=USR (316,85)
	WAITVS	X=USR (316,87)
	WCLOSE(xl,yl,x2,y2)	X=USR (316,76,xl,yl,x2,y2)
	WDISAB(p)	X=USR (316,77,p)
	WENAB	X=USR (316,78)
	WEXIT(p)	X=USR (316,79,p)
	WINIT(p)	X=USR (316,74,p)
	WOPEN(xl,yl,x2,y2)	X=USR (316,75,xl,yl,x2,y2)
	WORKON(p)	X=USR (316,75,p)
e	XAREA(xl,yl,x2,y2,c)	X=USR (316,4,xl,yl,x2,y2,c)
e	XCIRC(x,y,r,c)	X=USR (316,6,x,y,r,c)
e	XDOT(x,y,c)	X=USR (316,3,x,y,c)
e	XFCIR(x,y,r,c)	X=USR (316,8,x,y,r,c)
	XFILL	X=USR (316,67)
e	XFPOLY(c,a)	none
e	XLINE(xl,yl,x2,y2,c)	X=USR (316,0,xl,yl,x2,y2,c)
e	XPOLY(c,a)	X=USR (316,16,c,a)
e	XREAD(x,y,V)	none
e	XTEXT(x,y,c,"text ^")	* X=USR (316,12,x,y,c,t1)

Cromemco™

HIGH RESOLUTION GRAPHICS SOFTWARE

INSTRUCTION MANUAL

CROMEMCO, Inc.
280 Bernardo Avenue
Mountain View, CA 94043

Part no. 023-4015

August 1980

This manual was produced on a Cromemco Z2-H computer using the Cromemco Screen Editor. The edited text was formatted using the Cromemco Word Processing System Formatter. Final camera-ready copy was printed on a Cromemco 3355A printer.

Table of Contents

ABSTRACT

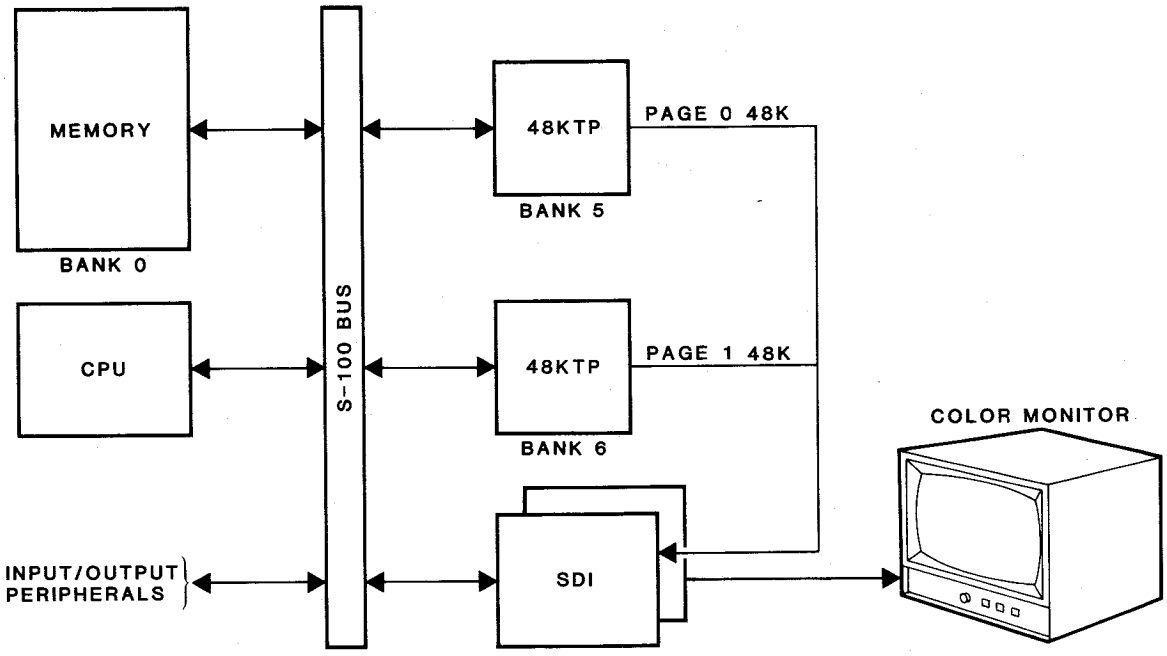
This manual describes the Cromemco SDI High Resolution Graphics software package. This package is available from Cromemco on either a 5" double-sided diskette (model SGS-S) or on an 8" diskette (model SGS-L). An SDI interface and two pages of 48KTP memory are required to **fully** utilize this software package.

Chapter 1

INTRODUCTION

The Cromemco SDI High Resolution Graphics software package makes full utilization of the SDI graphics interface possible from a high level language. The package is designed to work with Cromemco's 48KTP and 16KTP (Two Port) memory boards and requires two pages of Two Port memory with each page containing 48K bytes of RAM for complete utilization of all available calls.

The system configuration recommended for use with the SDI Graphics software is shown in the block diagram below.



An SDI System with Two 48KTP Boards

Notice that there are two banks of Two Port memory as well as the computer's **host** memory. These two banks of Two Port memory, bank 5 and 6, are used to

store picture data. A complete picture can be stored in each bank. In the terminology of Cromemco's high resolution graphics software package, these are referred to as page 0 and page 1 respectively.

This manual will explain how to create images on either page, using a variety of resolutions and colors, and create special effects by opening windows from one page onto the other.

The programs themselves, whether they are written in RATFOR, FORTRAN, BASIC or Z80 Assembly Language, are stored in bank 0, and all bank selecting is handled by the graphics software. The relationship between banks 0, 5 and 6 is shown on the following page.

1.1 Graphics Files

The six files which make up the graphics package are:

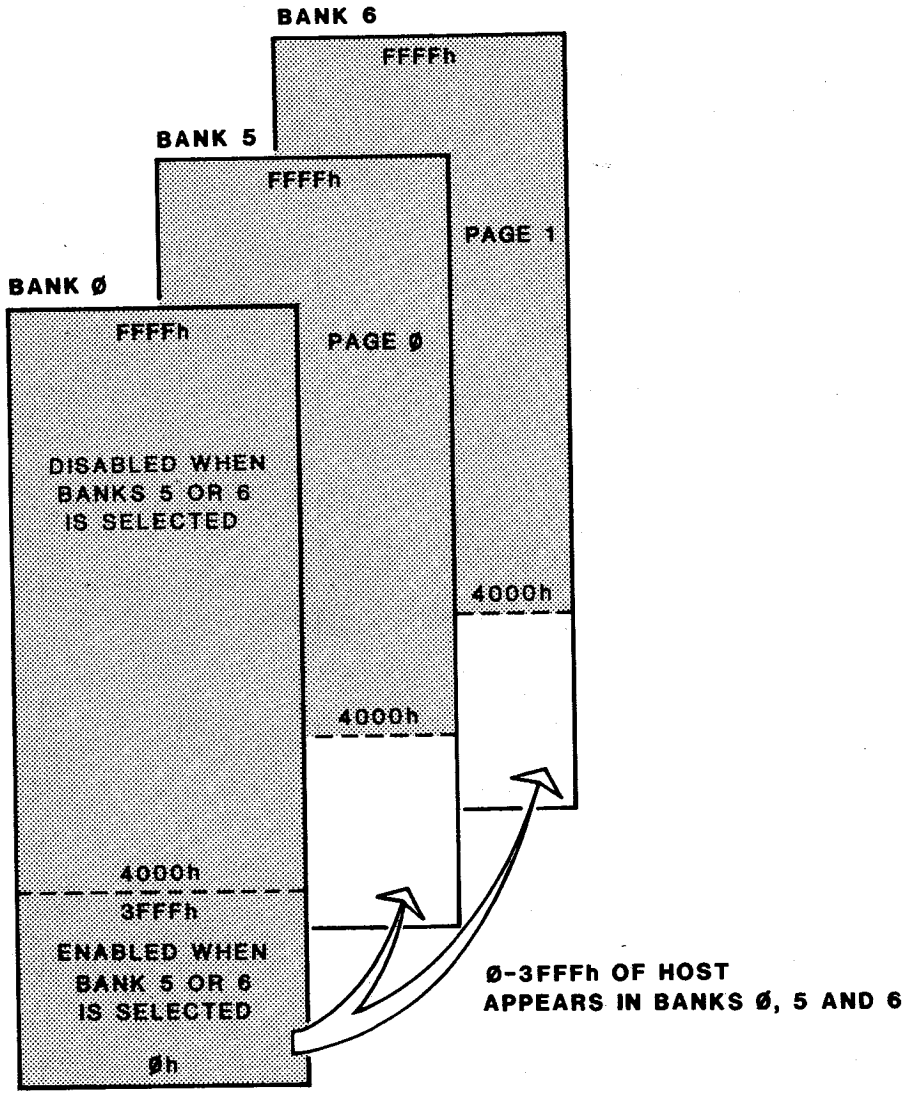
SDIFOR.REL - This is a collection of relocatable machine language subroutines which must be linked with either an Assembly language, a FORTRAN, or a RATFOR program.

SDIBAS.COM - This command file automatically loads both the SDI graphics package and BASIC. The active diskette must contain either Cromemco 16K Extended BASIC or Cromemco 32K Structured BASIC.

SDILIB - The SDI library module contains a set of procedures which can be incorporated into a 32K Structured BASIC program. These procedures do not increase the number of graphics calls available from a BASIC program. They do, however, allow the user to take advantage of the same formats used with FORTRAN and assembly language programs.

CMAPGEN.COM - This program is used for interactive color map generation. The output from this program is a relocatable file which can be linked with the user's FORTRAN or assembly file.

PIXSAVE.COM - This command file is called from CDOS and is used to store on disk a picture which is contained in either page 0 (bank 5) or page 1 (bank 6) of Two Port memory.



System with Two 48KTPs

PIXLOAD.COM - This command file is called from CDOS and is used to load a picture from disk to either page of Two Port memory.

1.2 Use of the Graphics Files

The graphics instructions available in the BASIC file (SDIBAS.COM) are a subset of the instructions contained in the FORTRAN/Assembly file (SDIFOR.REL). The differences between these two files and the procedures for incorporating them

into a user program are discussed later in this section.

Regardless of the programming language used the normal default state of the graphics display is the full color, medium resolution mode with the pixel in the lower left corner designated as (1,1) and the pixel in the upper right corner designated as (378,241). This designation is referred to as unscaled. These coordinate designations can be changed to any desired values by use of the SCALE call.

1.2.1 FORTRAN Programs

After writing a FORTRAN or Assembly language program which includes SDI graphics calls the user links his program with the file SDIFOR.REL using the LINK.COM file which is included with Cromemco's Macro Assembler, Cromemco's FORTRAN, or Cromemco's RATFOR packages. The order of linking is important since the graphics package must be in the low 16K of memory and hence should be first in the list of program modules to be linked. A typical linkage would have the following format:

```
LINK SDIFOR,BARGRAPH/E
```

When SDIFOR is linked to a program, the graphics calls which are available to the programmer have names which suggest the graphics operation to be performed. As an example, consider the case where we wish to draw a line from the point x=5, y=8 to the point x=98, y=4 using the color designated by color code 10. The respective FORTRAN or Assembly language call would be:

```
CALL XLINE(5,8,98,4,10)
```

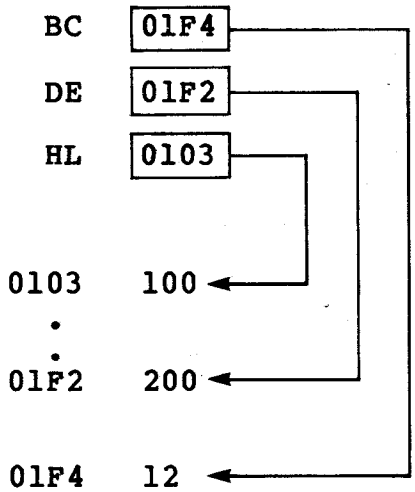
A format page for each of the available calls can be found in Chapters 2 and 3 of this manual.

1.2.2 Assembly Language Programs

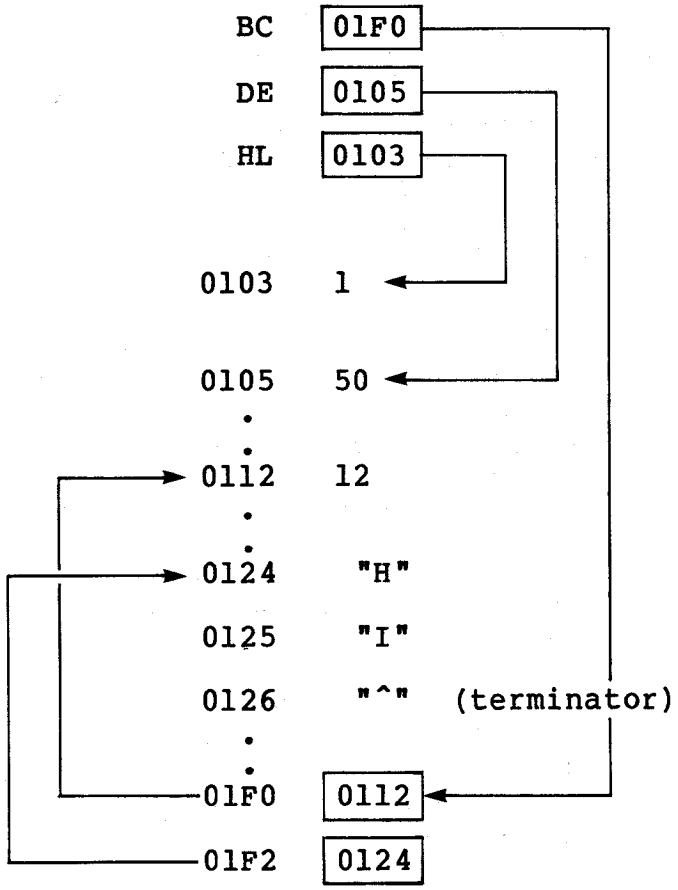
When linking SDIFOR with a Z80 program, the manner in which parameters are passed to the called subroutine is the same as for any FORTRAN subroutine. That is, if three or fewer parameters are passed, HL, DE, and BC point to the first, second and third parameters respectively. If four or more parameters are passed, HL and DE point to

the first two parameters, as before, while BC points to a block of 2 byte pointers. The first word (2 bytes) in the block points to the third parameter, the second word (2 bytes) in the block points to the fourth parameter, and so forth. This procedure is illustrated below.

3 or Fewer Parameters
e.g., CALL XDOT (100,200,12)



4 or More Parameters
e.g., CALL XTEXT (1,50,12,"Hl^")



1.2.3 BASIC Programs

In order to include graphics calls in a BASIC program both the graphics package and BASIC must be loaded. A description of this procedure follows.

From Cromemco's Disk Operating System the user types SDIBAS followed by a carriage return in order to load both the graphics package and BASIC. It is assumed that either Cromemco's 16K Extended BASIC or Cromemco's 32K Structured BASIC is on the disk located in the current drive or on a diskette in drive A. Graphics calls can then be executed by including user defined function calls within the program. Consider the same example used in the previous section where we wish to draw a line from the point x=5, y=8 to the point x=98, y=4 using the color designated by color code 10. The respective BASIC language call would be:

```
>>1000 X=USR(316,10,5,8,98,4,10)
```

where the 316 in the BASIC command designates the user defined graphics subroutine and the 10 designates the XLINE call within the subroutine. A complete list of graphics routines and their corresponding code numbers appears in Appendix A and on the inside front cover.

With Structured BASIC programs the user has the option of incorporating the procedure calls contained in the file SDILIB into a program. In order to make room in memory for SDILIB the user may need to generate a smaller version of SBASIC by using BASICGEN.COM. In such a case the shortened version of SBASIC should be named SBASIC.COM. For details on how to use BASICGEN refer to the addendum to the Cromemco 32K Structured BASIC manual (part number 023-0094). Although SDILIB does not increase the number of calls available when programming in SBASIC, it does allow the programmer to use calls similar to the FORTRAN/Assembler calls. For the previously used example the programmer would use the format:

```
>>1000 .XLINE (5,8,98,4,10)
```

Note in the above example that XLINE is set off with a "." in front and a space following it.

Chapters 2 and 3 of this manual contain format pages for each of the calls available in the

graphics package. Where equivalent BASIC calls are available the format of the BASIC call is included. Those using this graphics software package will find that the available calls are sufficient to fully utilize all the capabilities of Cromemco's SDI high resolution graphics interface board.

1.3 Color Map

The 16 colors stored in the color map have been preset for the values shown in Table 1. A comparison of the color map values and the corresponding colors in the table should serve as a good starting point for creating your own color map. The contents of any color in this color map can be modified using the call Define Color (DEFCLR) as outlined in Chapter 2. In addition, when programming in FORTRAN or Assembly languages the programmer has the option of creating color maps using CMAPGEN. A discussion of CMAPGEN and its use is presented in Chapter 4.

Memory Map Color Code	Preset Color	Color Composition Red	Green	Blue
0	Black	0	0	0
1	Burnt Orange	13	4	0
2	Dark Blue	0	0	10
3	Bright Blue	0	0	15
4	Dark Green	0	10	0
5	Bright Green	0	15	0
6	Dark Slate Blue	0	5	5
7	Bright Slate Blue	0	8	8
8	Dark Red	10	0	0
9	Bright Red	15	0	0
10	Dark Purple	5	0	5
11	Bright Purple	8	0	8
12	Orange	15	6	0
13	Khaki	8	8	0
14	Yellow	12	9	0
15	White	15	15	15

Table 1
Preset Values of the Memory Map Color Codes

The calls available from the SDI graphics software package fall into two categories. The first category can be referred to as housekeeping calls. These are outlined in the next chapter. The second

category is the graphics calls which generate the actual display. These are covered in Chapter 3.

Chapter 2

SDI HOUSEKEEPING CALLS

While the housekeeping calls do not actually generate a display, they do determine the active work area (i.e., is a picture being generated on page 0 or 1?), what is displayed, and how it is displayed. Since these calls have a major influence on how the graphics calls are interpreted it is important that the user have a clear understanding of these calls and how they affect the display. All variables in the housekeeping and graphics calls must be integers.

For ease of understanding these calls have been subdivided into four logical groups and are discussed in the following four sections. Each of the housekeeping calls is discussed in some detail on subsequent pages of this chapter with each call starting on a new page. For ease of reference an alphabetical listing of all calls (i.e., both the housekeeping and the graphics calls) appears on the inside front cover of this manual. In addition, at the beginning of each section is a list of the calls included in that section. The n/a notation in the listings indicates that the call is not available from either 32K Structured BASIC or 16K Extended BASIC.

2.1 Fundamental Housekeeping Calls

In learning to use the graphics package the programmer will find it necessary to master the following set of housekeeping calls first. The calls included in this section are listed in Table 2.1.

BASIC Call #	FORTRAN or Assembly Call	Arguments(s)
(51)	CLIP	(none)
(84)	DEFCLR	(c,R,G,B)
(70)	DISP	(p)
(82)	ERABOX	(x1,y1,x2,y2)
(n/a)	GRAFIX	(none)
(54)	HSCALE	(x1,x2,y1,y2)
(68)	INIT	(none)
(n/a)	INIT1	(none)
(71)	PAGE	(none)
(80)	RES	(r)
(81)	RESBOX	(x1,y1,x2,y2)
(49)	SCALE	(x1,x2,y1,y2)
(64)	SCROFF	(none)
(63)	SCRON	(none)
(52)	UNCLIP	(none)
(50)	UNSCAL	(none)
(69)	WORKON	(p)

Table 2.1
Alphabetical List of the
Fundamental Housekeeping Calls

The BASIC call # applies to the 16K Extended version and the Structured version when the SDILIB procedure file has not been included in the program.

Function: **GRAFIX**

format:

FORTRAN:	CALL GRAFIX
Assembly:	n/a
BASIC:	n/a
SBASIC:	n/a

This must be the first executable instruction in a FORTRAN program. The purpose of this call is to move the FORTRAN stack from upper core into the lower 16K of RAM. This move is necessary because the lower 16K of RAM is the only part of system memory that resides in banks 5 and 6 with the Two Port memory boards which contain the image. This call is NOT used in BASIC programs.

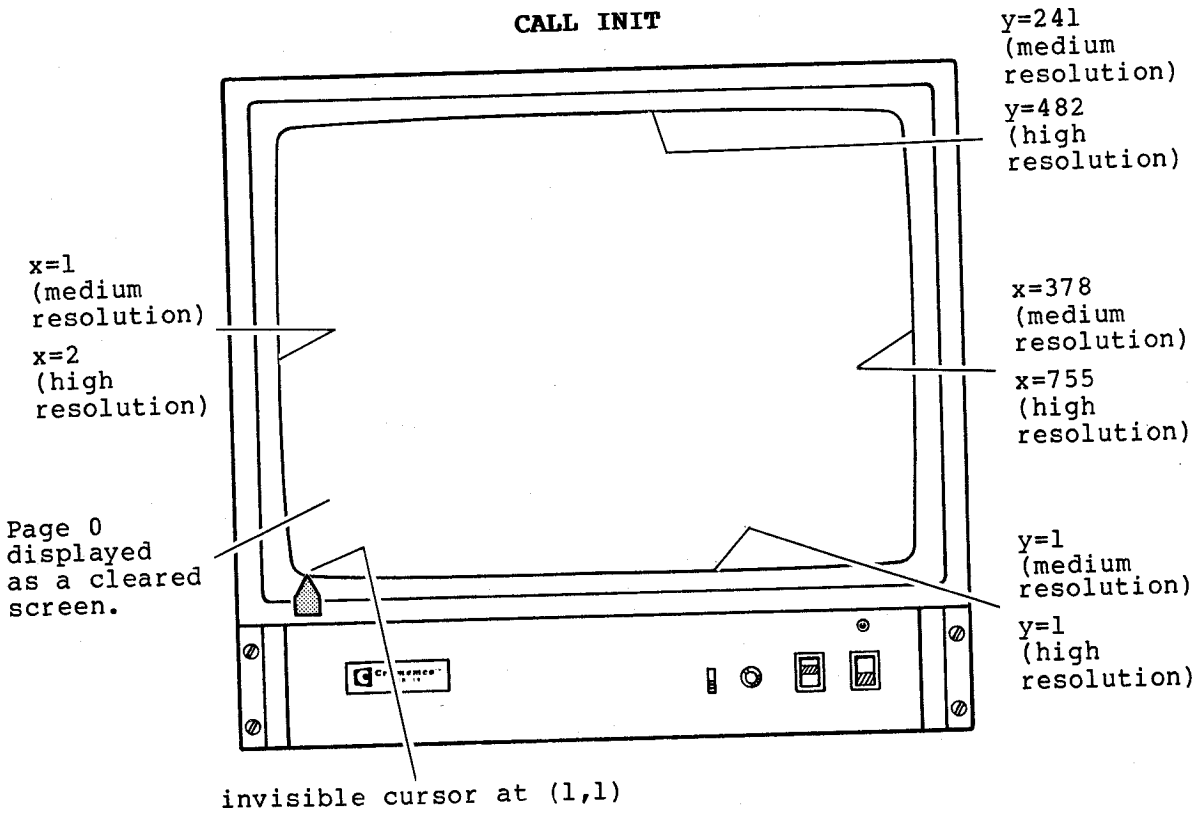
Function: **INIT**alize

format:

FORTRAN:	CALL INIT
Assembly:	CALL INIT
BASIC:	USR(316,68)
SBASIC:	.INIT

This call can be executed at any point within a program. It would normally follow GRAFIX in a FORTRAN program or be the first executable call in a BASIC program. This call performs the following housekeeping operations:

- Clears page 0 of the two port memory
- Turns on the SDI interface and displays page 0
- Calls the housekeeping function WORKON (0)
- Outputs the set of standard colors to the memory map on the SDI video board
- Moves the cursor to pixel (1,1)
- Calls the housekeeping function CLIP
- Sets scaling to the default values
- Selects the 16 color, medium resolution mode



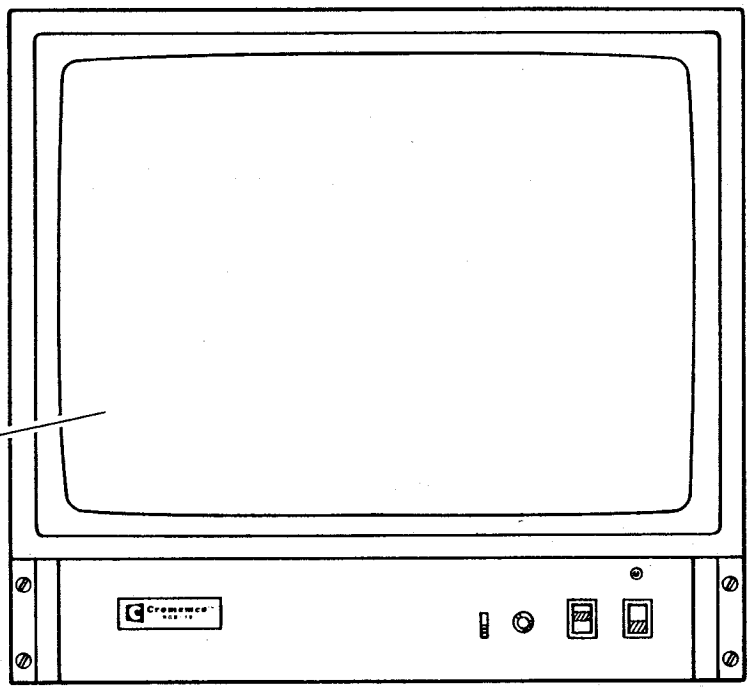
Function: **INIT**alize **1**

format:

FORTRAN:	CALL INIT1
Assembly:	CALL INIT1
BASIC	n/a
SBASIC	n/a

This call sets page 1 as the current work and display page and clears it. All other housekeeping parameters remain as previously set.

CALL INIT1



Page 1 displayed as a cleared screen.

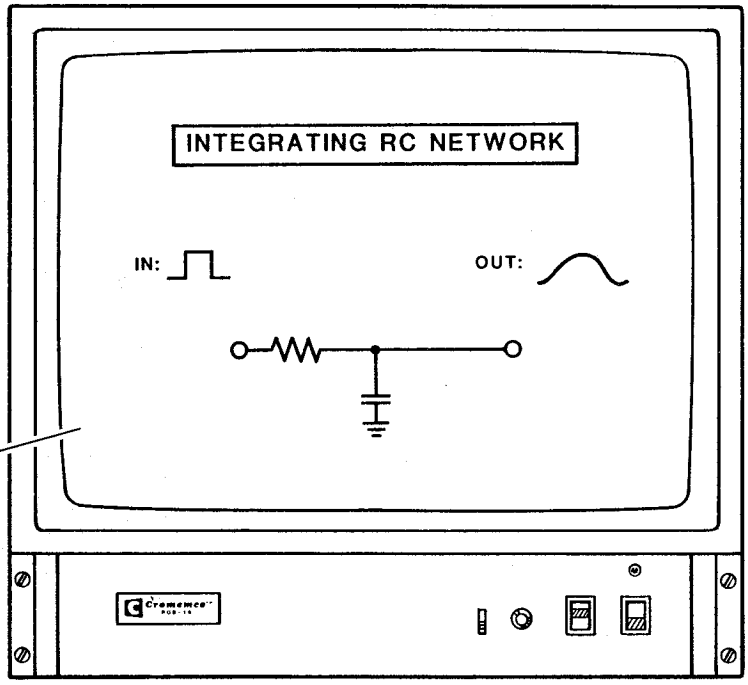
Function: **SCR**een **ON**

format:

FORTRAN:	CALL SCRON
Assembly:	CALL SCRON
BASIC:	USR(316,63)
SBASIC:	.SCRON

This call turns on the SDI display so that the page selected with the DISP call is displayed. When used in conjunction with SCROFF they constitute an on/off switch.

CALL SCRON



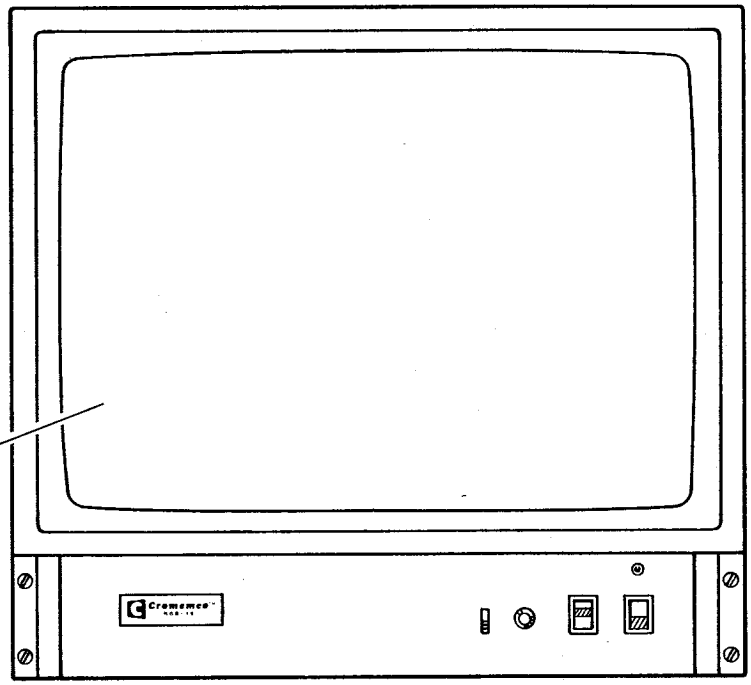
Function: **SCR**een **OFF**

format:

FORTRAN:	CALL SCROFF
Assembly:	CALL SCROFF
BASIC:	USR(316,64)
SBASIC:	.SCROFF

This call turns the SDI display off. The currently selected display page and work page remain in force. When used in conjunction with the SCRON call they constitute an on/off switch.

CALL SCROFF



Function: **WORKON**

format:

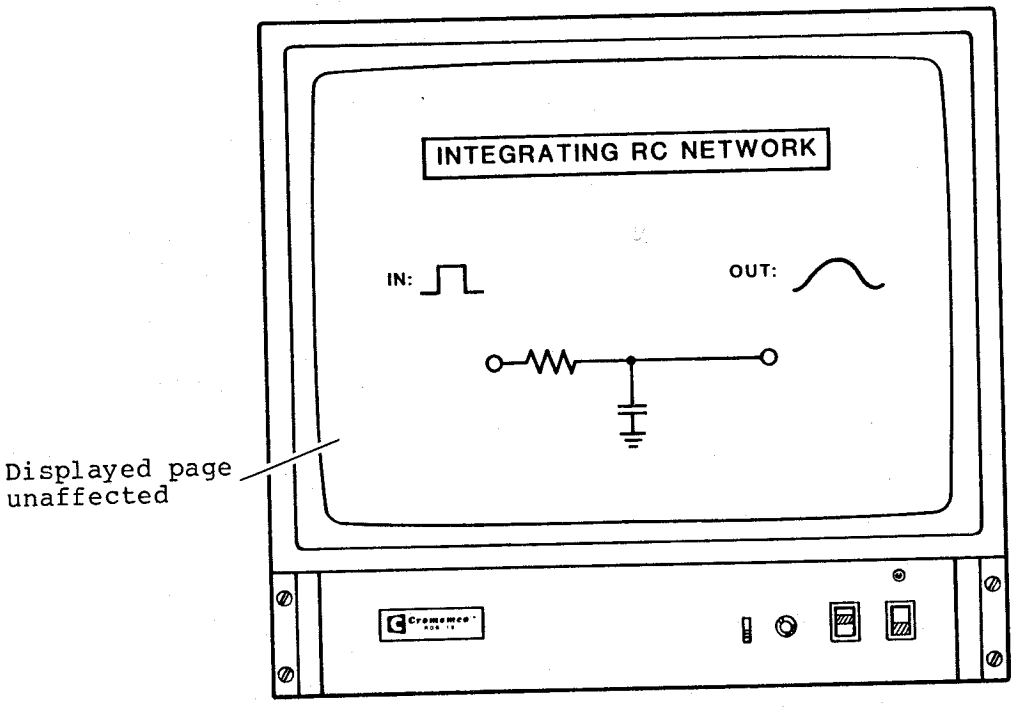
FORTRAN:	CALL WORKON(p)
Assembly:	CALL WORKON(p)
BASIC:	USR(316,69,p)
SBASIC:	.WORKON (p)

where:

p = 0 or 1 and denotes page number

This call selects the page which the program will write to, erase, set the resolution of, scale, read, etc. It does not necessarily have to be the page which is being displayed.

CALL WORKON(p)



Function: **DISP**lay

format:

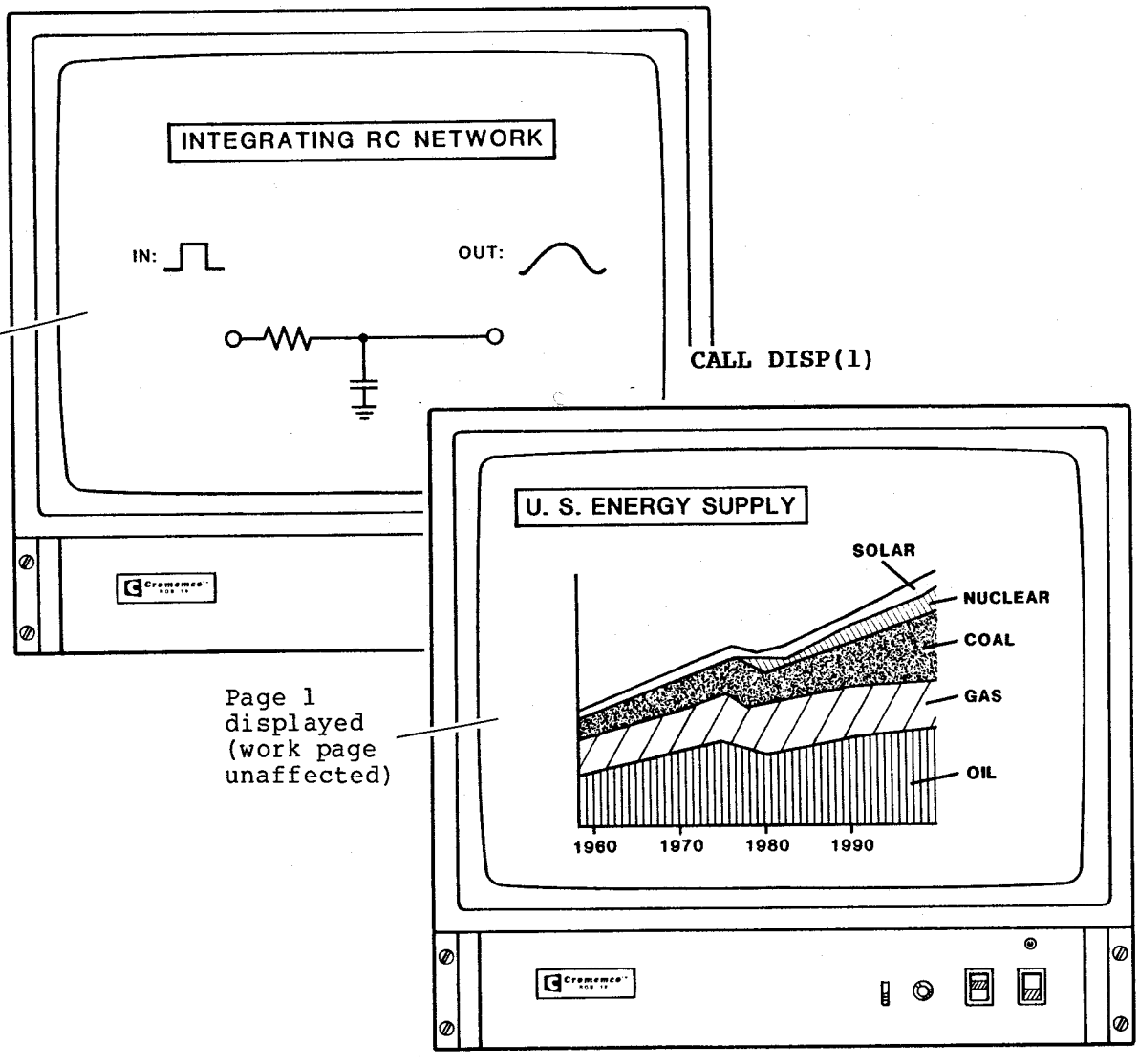
FORTRAN:	CALL DISP(p)
Assembly:	CALL DISP(p)
BASIC:	USR(316,70,p)
SBASIC:	.DISP (p)

where:

p = 0 or 1 and denotes page number

This call selects the page to be displayed on the video screen.

CALL DISP(0)



Function: **RES**olution

format:

FORTRAN:	CALL RES(r)
Assembly:	CALL RES(r)
BASIC:	USR(316,80,r)
SBASIC:	.RES (r)

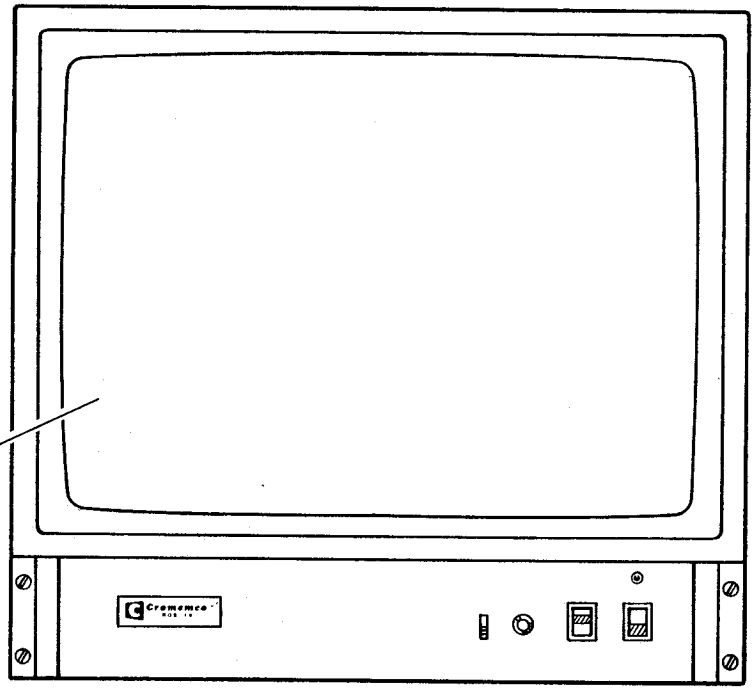
where:

r = 0 for 16 color medium resolution

1 for 2 color high resolution

This call selects the background resolution of the work page. All subsequent calls to RESBOX and ERABOX refer to this background resolution. Thus, RESBOX will allocate areas complementary to this background while ERABOX will allocate areas the same as this background.

CALL RES(r)



Background resolution of entire workpage set. Display page unaffected unless it is equal to work page.

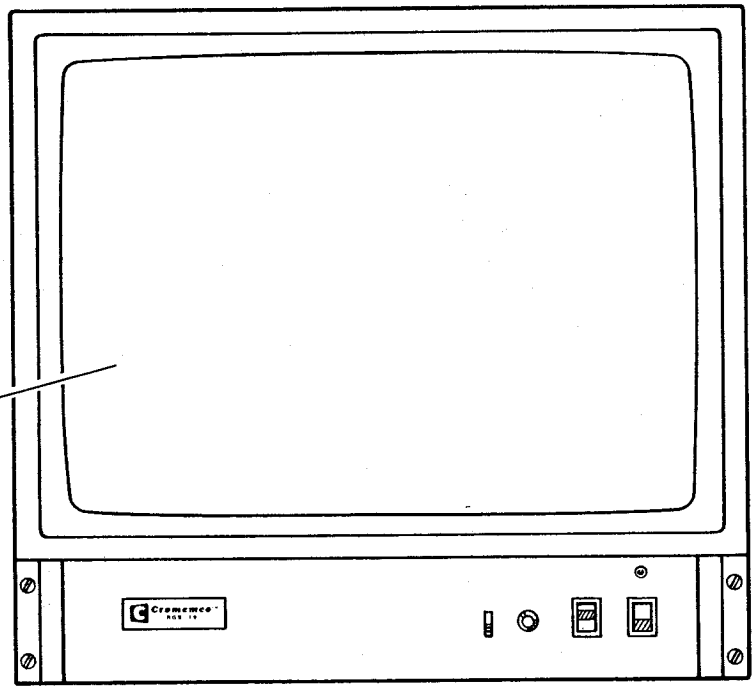
Function: clear **PAGE**

format:

FORTRAN:	CALL PAGE
Assembly:	CALL PAGE
BASIC:	USR(316,71)
SBASIC:	.PAGE

This call fills the image area of the work page with 0's. With the standard colors this is equivalent to erasing the screen.

CALL PAGE



Function: **DEFine CoLoR**

format:

FORTRAN:	CALL DEFCLR(c,R,G,B)
Assembly:	CALL DEFCLR(c,R,G,B)
BASIC:	USR(316,84,c,R,G,B)
SBASIC:	.DEFCLR (c,R,G,B)

where: all variables are integers between 0 and 15

c = the color code being specified

R,G,B = intensity of the red, green and blue color components.

0 corresponds to the color component having its minimum intensity and

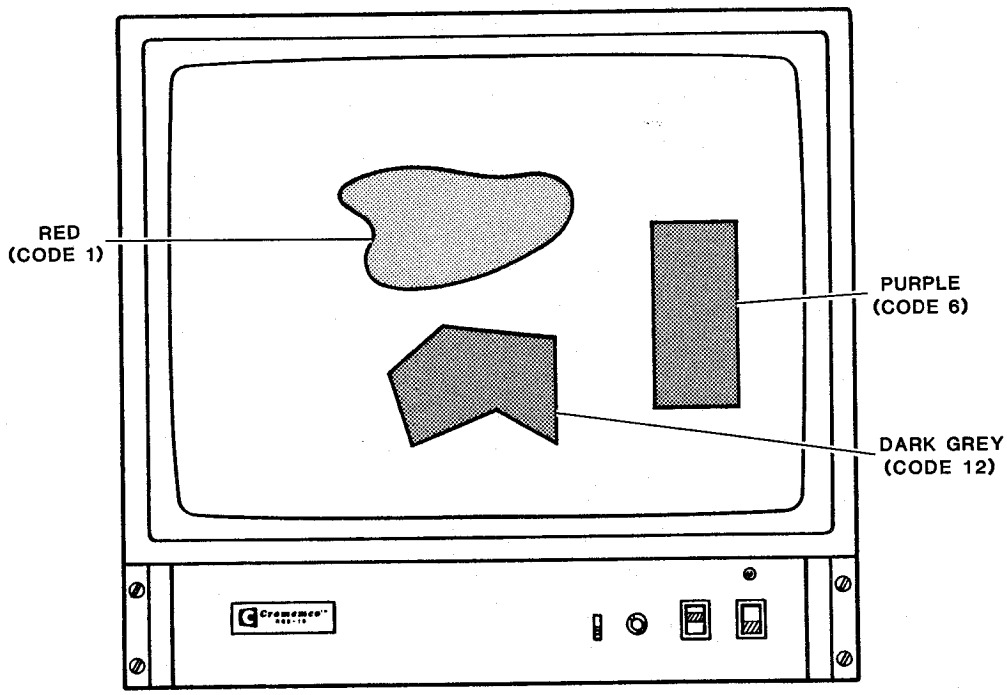
15 corresponds to the color component having its maximum intensity.

This call allows the user to change one or more of the colors stored in the color map. The color change is timed so that it takes place during a vertical blanking interval following the call.

(Please see diagram on the next page.)

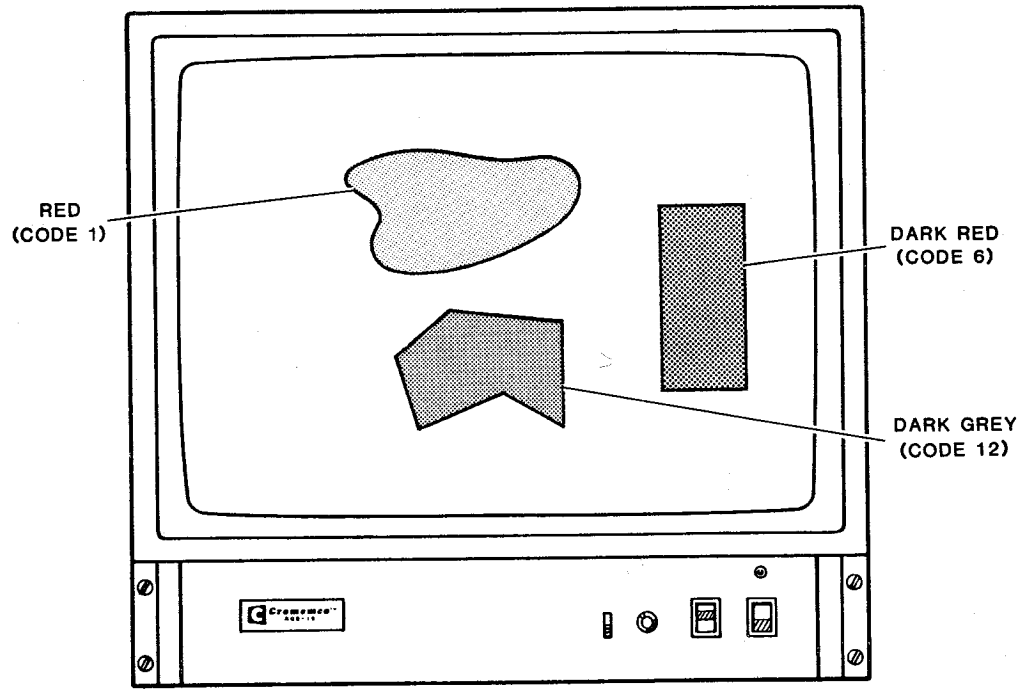
CALL DEFCLR(6,4,0,0)

BEFORE



CALL DEFCLR(6,4,0,0)

AFTER



Function: **SCALE**

format:

FORTRAN:	CALL SCALE(x1,x2,y1,y2)
Assembly:	CALL SCALE(x1,x2,y1,y2)
BASIC:	USR(316,49,x1,x2,y1,y2)
SBASIC:	.SCALE (x1,x2,y1,y2)

where:

x1 = the number associated with the left edge of the display screen

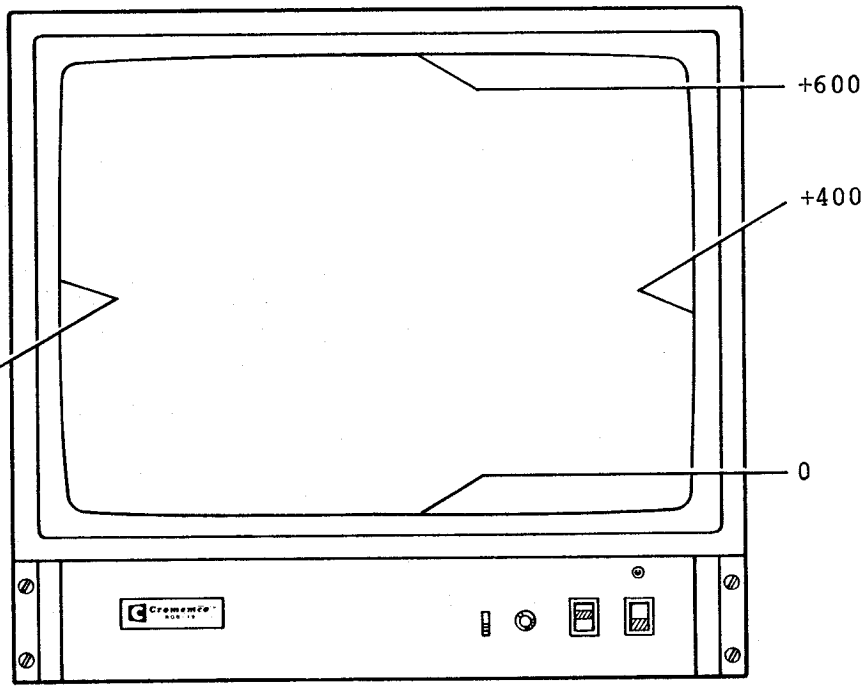
x2 = the number associated with the right edge of the display screen with x2 > x1

y1 = the number associated with the bottom of the display screen

y2 = the number associated with the top of the display screen with y2 > y1

This call is used with either the 16 color medium resolution mode or the two color high resolution mode and permits you to scale the display area of the work page. The normal unscaled values of x1, x2, y1, and y2 in the medium resolution are 1, 378, 1, and 241 respectively and in the high resolution mode the respective values are 2,755,1,482.

CALL SCALE(-400,+400,0,600)



These coordinates used for medium resolution - 16 color mode only.

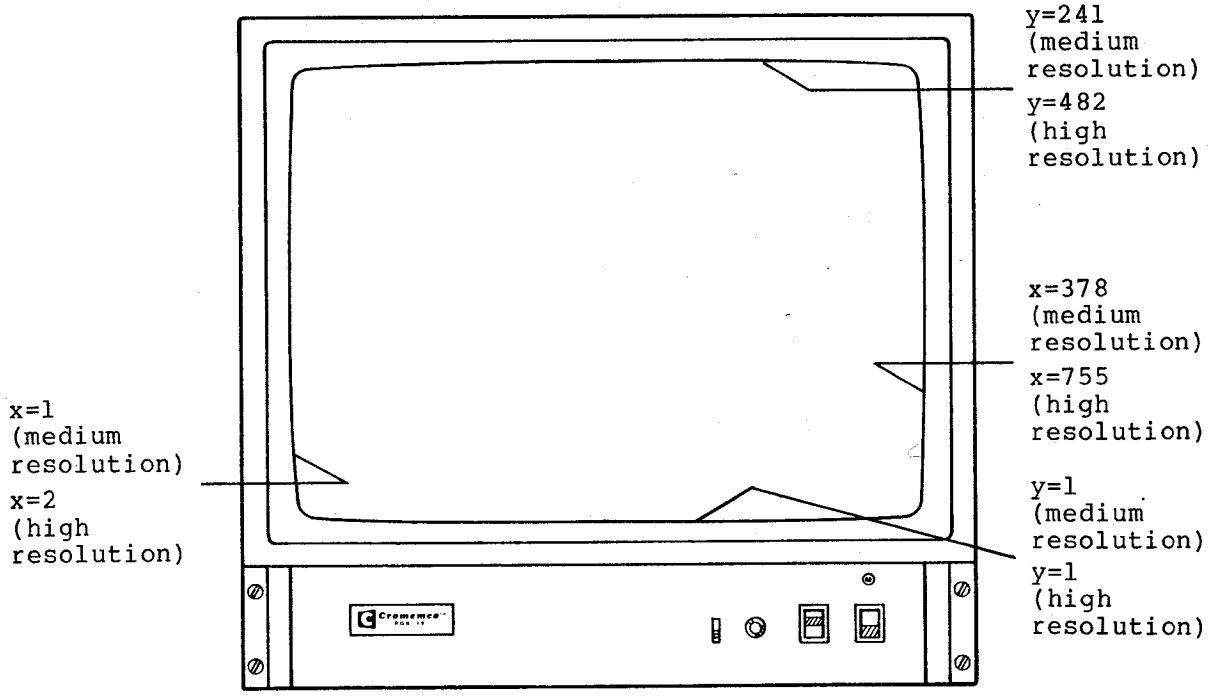
Function: **UNSCALe**

format:

	FORTRAN:	CALL UNSCAL
	Assembly:	CALL UNSCAL
	BASIC:	USR(316,50)
	SBASIC:	.UNSCAL

This call returns the scaling of the work page to the default values. In the high resolution mode horizontal points can have values between 2 and 755 while vertical points can have values between 1 and 482. For the medium resolution mode the range is 1 to 378 for the horizontal and 1 to 241 for the vertical. The user should be aware that the plotting speed is considerably faster when the default scaling values are used.

CALL UNSCAL



Only the current workpage is affected.

Function: **CLIP**

format:

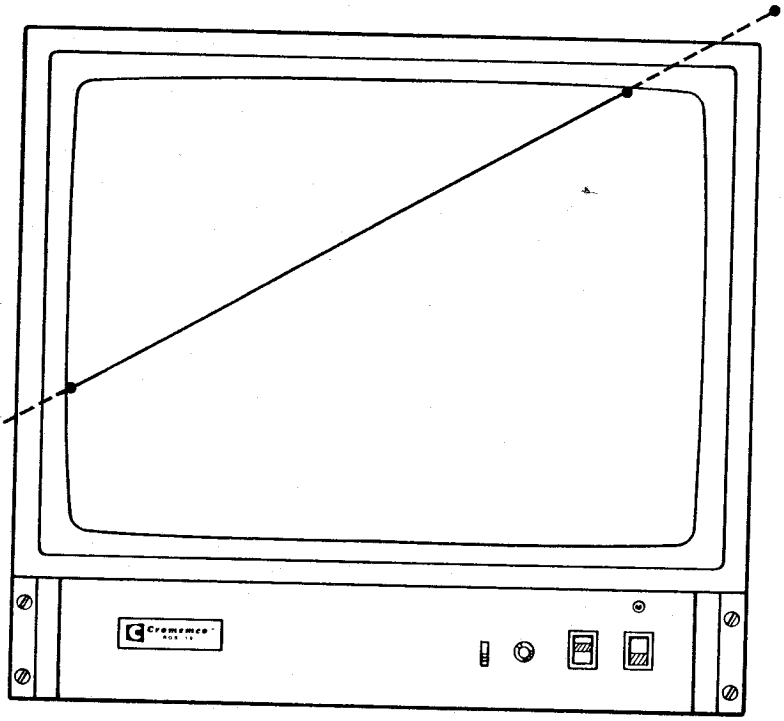
FORTRAN:	CALL CLIP
Assembly:	CALL CLIP
BASIC:	USR(316,51)
SBASIC:	.CLIP

This call eliminates any problems which might arise from trying to plot outside the screen area. Those values generated outside the scaled range of the screen area are ignored.

For example, if part of a circle which is being plotted runs off the screen, then that part of the circle will be ignored. The part of the circle which is to appear on the screen will be displayed.

Although the plot routine is slower after this call has been made, it should be used during program development since any attempt to plot outside the display area will probably overwrite the program or the Operating System. After the program has been debugged and you are sure that all plotted points are within the display area you can go back and insert an UNCLIP call to speed up the program. Remember that CLIP is called by the INIT call. When executed, the CLIP call is in effect for both pages of the display.

```
CALL CLIP
CALL XLINE(-50,50,400,260,1)
```



Clip allows the visible portion of off: screen segment to be plotted.

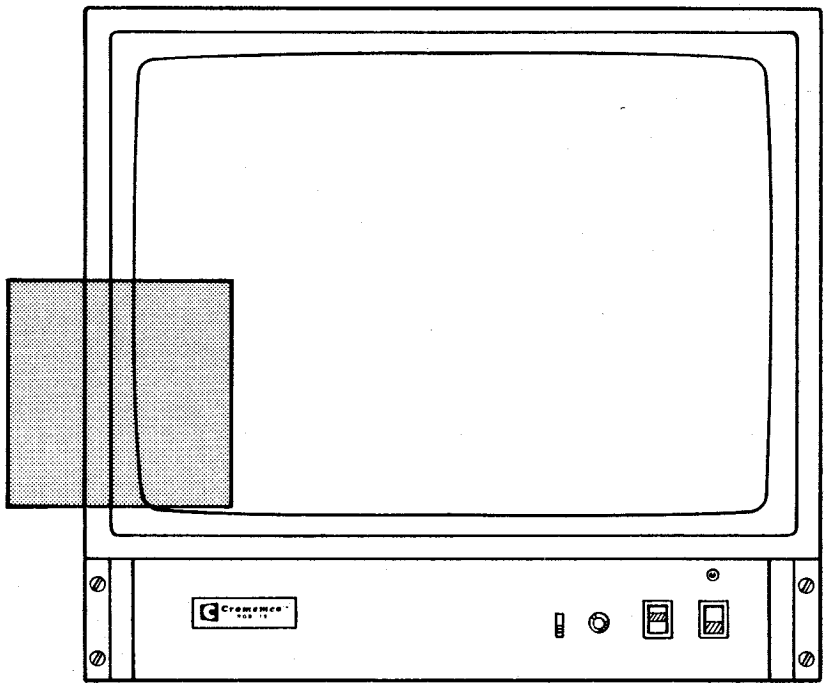
Function: **UNCLIP**

format:

FORTRAN:	CALL UNCLIP
Assembly:	CALL UNCLIP
BASIC:	USR(316,52)
SBASIC:	.UNCLIP

After this call has been executed no check is made to determine if plot parameters are within the screen area. If values are obtained which are outside the screen area it will probably result in either the program or the Operating System being overwritten. This call increases the plotting speed and therefore is useful after the program has been debugged and you know all plots lie within the scaled area of the screen. When this call is executed it is in effect for both pages of memory. During program development it is good practice to include the CLIP call in your program until you are certain that all plots are within the display area.

```
CALL UNCLIP
CALL XAREA(-50,1,50,100,7)
```



Writing off the screen will have unpredictable results.

Function: **RES**olution **BOX**

format:

FORTRAN:	CALL RESBOX(x1,y1,x2,y2)
Assembly:	CALL RESBOX(x1,y1,x2,y2)
BASIC:	USR(316,81,x1,y1,x2,y2)
SBASIC:	.RESBOX (x1,y1,x2,y2)

where:

x1,y1 = coordinates of the lower left corner of a box

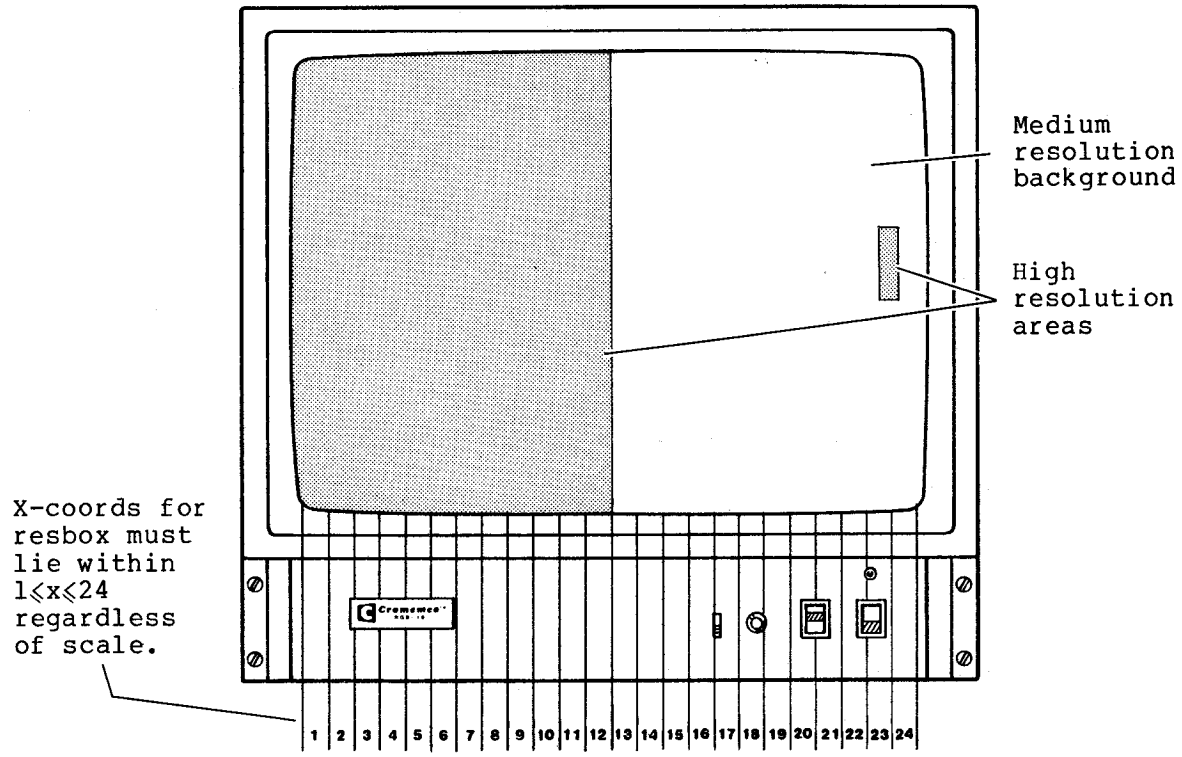
x2,y2 = coordinates of the top right corner of a box

x1 and x2 have values between 1 and 24

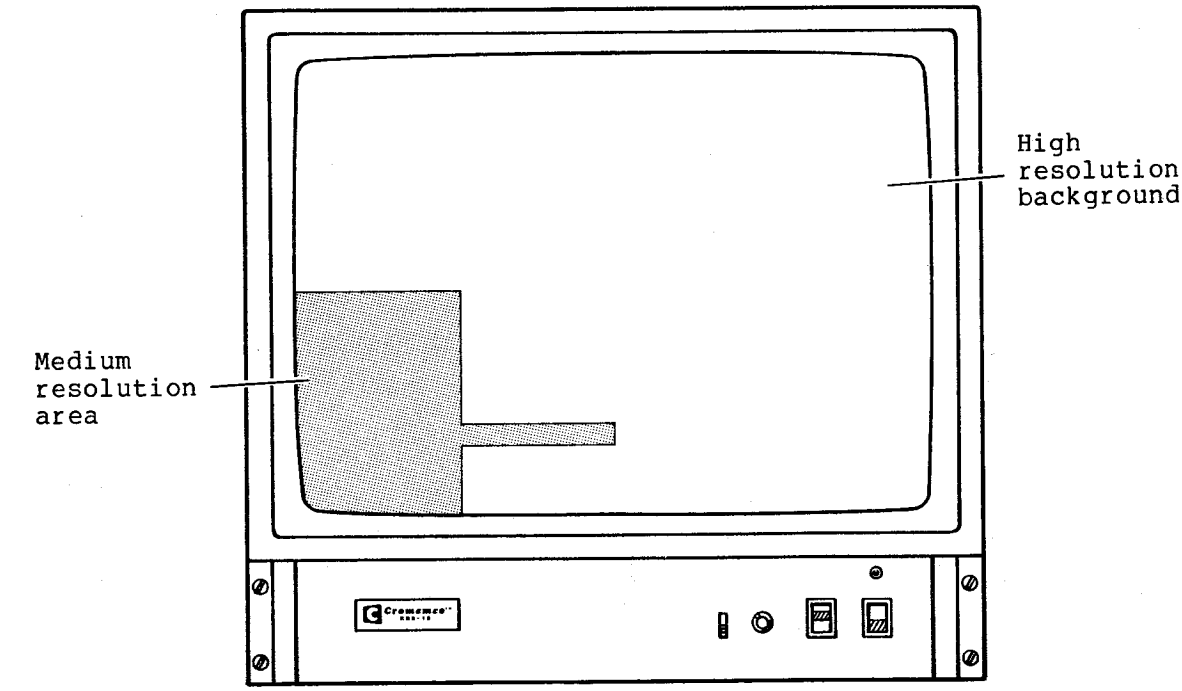
y1 and y2 are within the boundaries specified by SCALE or between 1 and 241 for the medium resolution unscaled mode and between 1 and 482 for the high resolution unscaled mode

This call allows you to change the resolution of a rectangular region of the work page. If the background resolution of the work page as determined by the most recent call to RES is medium resolution then the call will produce a region having 2 color high resolution capability. Similarly if the most recent call to RES caused the background resolution of the work page to be high resolution then the call will produce a region having medium resolution capability.

```
CALL RES(0)
CALL RESBOX(1,1,12,241)
CALL RESBOX(23,120,23,150)
```




```
CALL RES(1)
CALL RESBOX(1,1,6,120)
CALL RESBOX(3,50,12,60)
```



Function: **ERA**se **BOX**

format:

```
FORTRAN:   CALL ERABOX(x1,y1,x2,y2)
Assembly:  CALL ERABOX(x1,y1,x2,y2)
BASIC:     USR(316,82,x1,y1,x2,y2)
SBASIC:    .ERABOX (x1,y1,x2,y2)
```

where:

x1,y1 = coordinates of the lower left corner of a box

x2,y2 = coordinates of the top right corner of a box

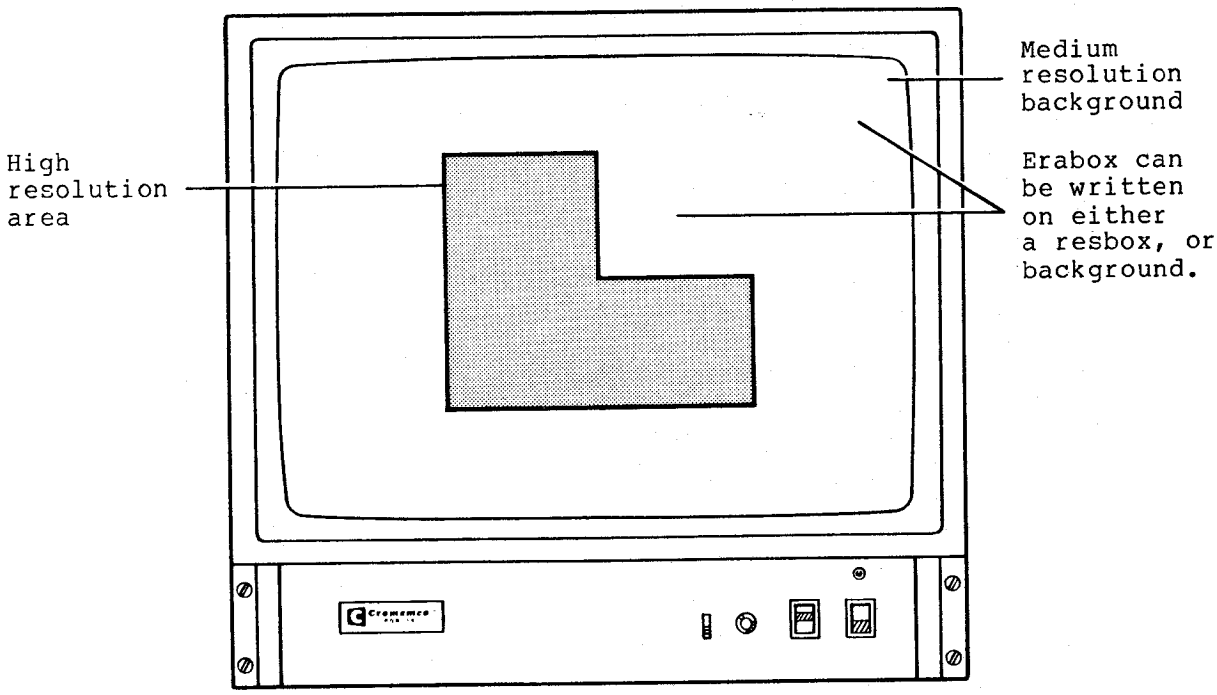
x1 and x2 have values between 1 and 24

y1 and y2 are within the boundaries specified by SCALE

This call allows the programmer to create a box having the original background resolution as determined by the most recent RES call. The area affected by ERABOX may be either totally within or overlapping part of a box created by a previous call to RESBOX. If ERABOX is called, and part or all the region defined by the call is already in the backgound resolution, then there is no change in that region.

(Please see diagram on next page.)

```
CALL RES(0)
CALL RESBOX(6,50,18,200)
CALL ERABOX(12,120,24,241)
```



2.2 Window Calls

Cromemco's SDI graphics interface gives the user the capability to open windows in the page of Two Port memory being displayed and displaying sections from the second page of Two Port memory in this window. The resolution selected for the image in the window does not have to be the same as the resolution selected for the original page. This capability permits some very interesting and beautiful effects. The calls available for utilizing the windowing capabilities of the SDI graphics interface are listed in Table 2.2.

BASIC Call #	**FORTRAN** or **Assembly Call**	**Argument(s)**
(76)	WCLOSE	(x1,y1,x2,y2)
(77)	WDISAB	(p)
(78)	WENAB	(none)
(79)	WEXIT	(p)
(74)	WINIT	(p)
(75)	WOPEN	(x1,y1,x2,y2)

Table 2.2
Alphabetical List of the
Window Housekeeping Calls

The BASIC call # applies to the 16K Extended version and the Structured version when SDILIB has not been included in the program.

Function: **Window INITalization**

format:

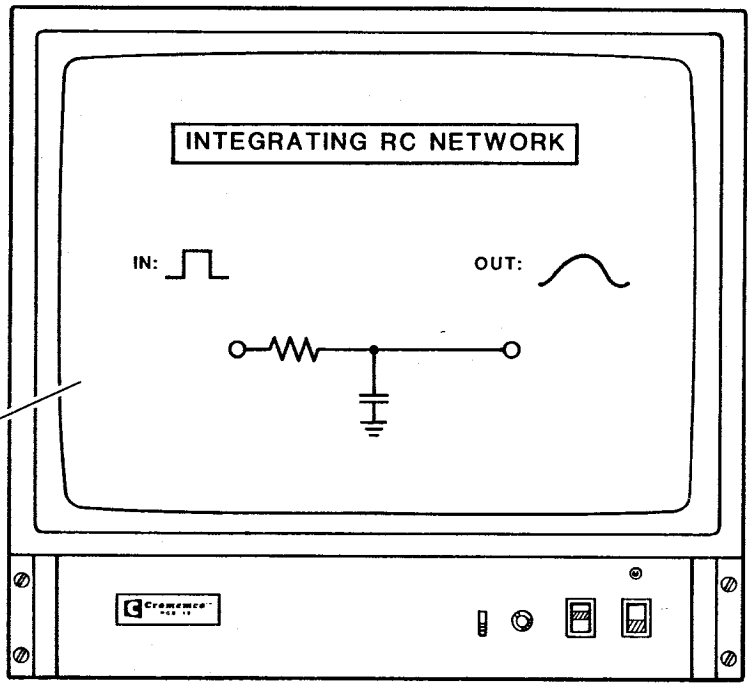
FORTRAN:	CALL WINIT(p)
Assembly:	CALL WINIT(p)
BASIC:	USR(316,74,p)
SBASIC:	.WINIT (p)

where:

p = 0 or 1 and designates the page

This call makes the specified page the front page and displays it in its entirety. Windows can now be opened in this page thus displaying the corresponding regions of the second page.

CALL WINIT(0)



Page 0 will be displayed if winit (0) is called. No windows open after initialization.

Function: **Window OPEN**

format:

FORTRAN:	CALL WOPEN(x1,y1,x2,y2)
Assembly:	CALL WOPEN(x1,y1,x2,y2)
BASIC:	USR(316,75,x1,y1,x2,y2)
SBASIC:	.WOPEN (x1,y1,x2,y2)

where:

x1,y1 = coordinates of the lower left corner of a box

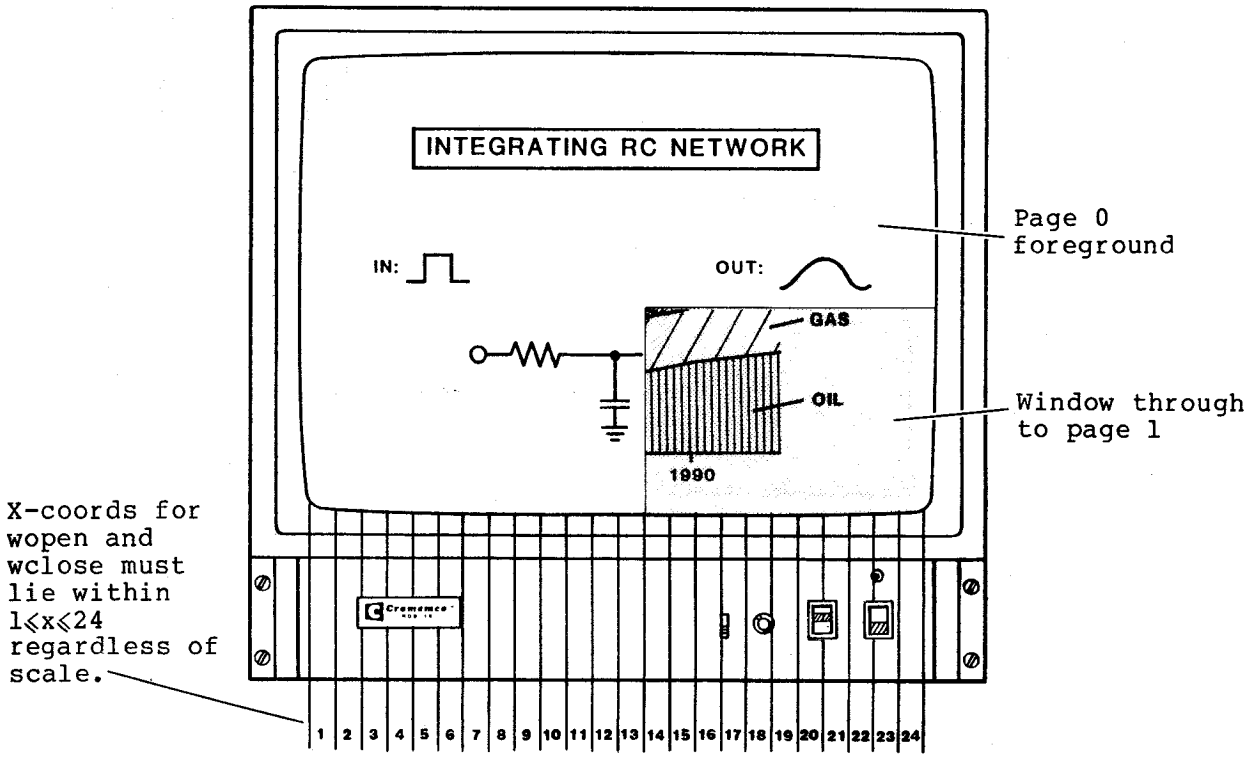
x2,y2 = coordinates of the top right corner of a box

x1 and x2 have values between 1 and 24

y1 and y2 are within the boundaries specified by SCALE

This call opens a window in the primary page having the dimensions and location specified by the variables in the call.

```
CALL WINIT(0)
CALL WOPEN(14,1,24,120)
```



Function: **Window CLOSE**

format:

FORTRAN:	CALL WCLOSE(x1,y1,x2,y2)
Assembly:	CALL WCLOSE(x1,y1,x2,y2)
BASIC:	USR(316,76,x1,y1,x2,y2)
SBASIC:	.WCLOSE (x1,y1,x2,y2)

where:

x1,y1 = coordinates of the lower left corner of a box

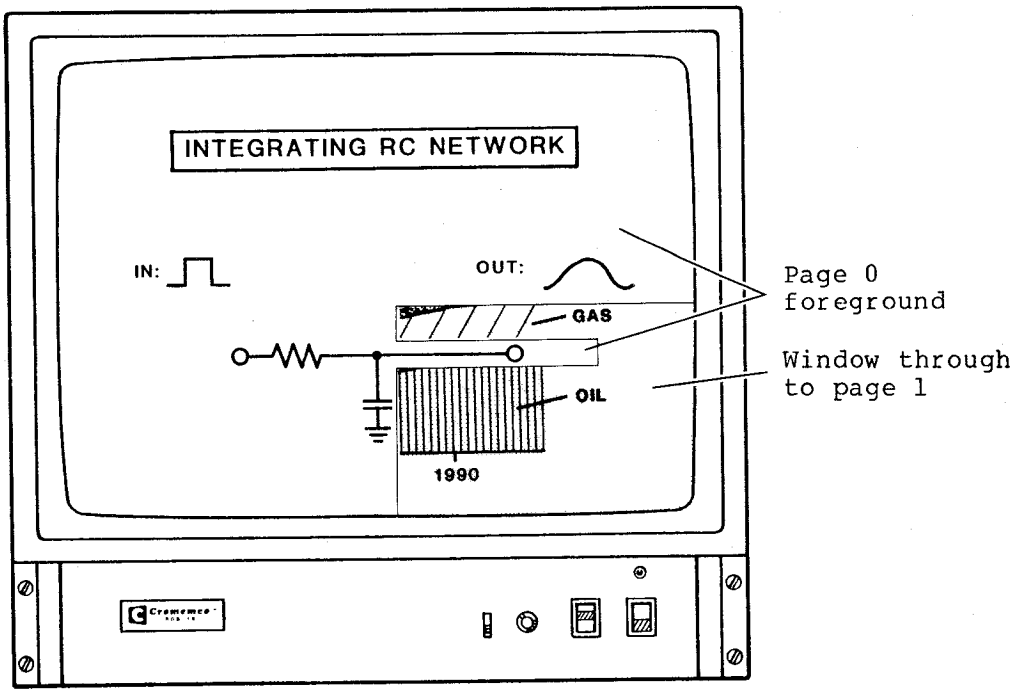
x2,y2 = coordinates of the top right corner of a box

x1 and x2 have values between 1 and 24

y1 and y2 are within the boundaries specified by SCALE

This call closes a window in the primary page having the dimensions and location specified by the call parameters. If part (or all) of the area specified does not presently have a window open, the call will not affect that part of the area.

```
CALL WINIT(0)
CALL WOPEN(14,1,24,120)
CALL WCLOSE(14,90,20,105)
```



Function: **Window DISABle**

format:

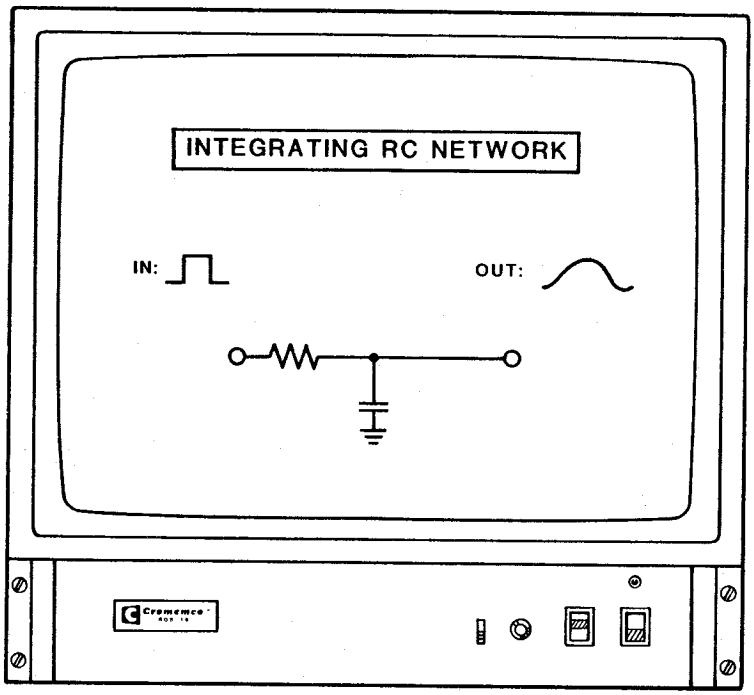
FORTRAN:	CALL WDISAB(p)
Assembly:	CALL WDISAB(p)
BASIC:	USR(316,77,p)
SBASIC:	.WDISAB (p)

where:

p = 0 or 1 and designates the page

This call displays the designated page in its entirety without windows. Previously designated windows in the page are not affected and can be reopened using the call WENAB. Either page may be specified without regard for the present work page or display page.

CALL WDISAB(0)



Page 0 displayed after wdisab(0). Windows are preserved but not used.

Function: **Window ENABle**

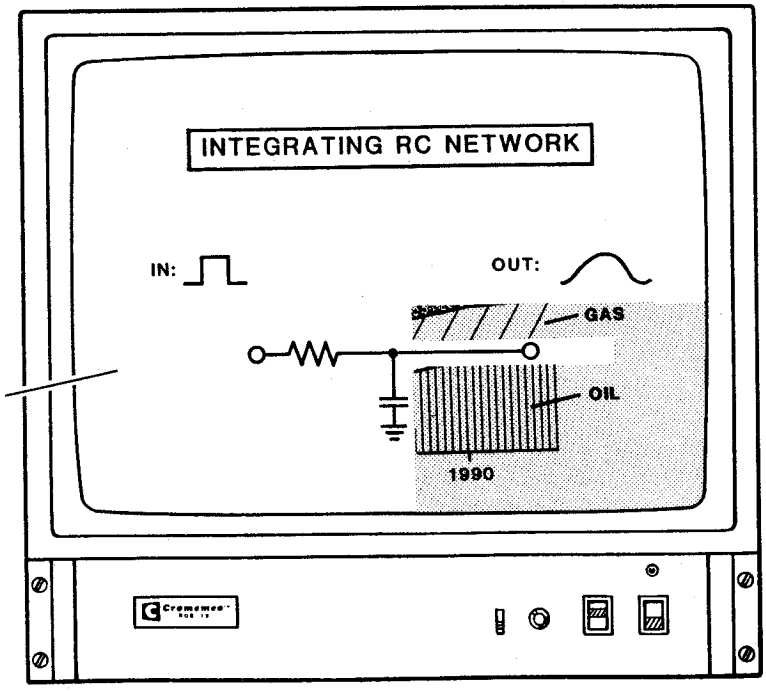
format:

FORTRAN:	CALL WENAB
Assembly:	CALL WENAB
BASIC:	USR(316,78)
SBASIC:	.WENAB

This call is the inverse of WDISAB and returns to the previously designated primary page with any designated windows open to the secondary page. The original window configuration before the call to WDISAB along with any new window opened or closed since will appear after the call to WENAB.

CALL WENAB

Windows reopened. New windows may have been opened since last wdisab call.



Function: **Window EXIT**

format:

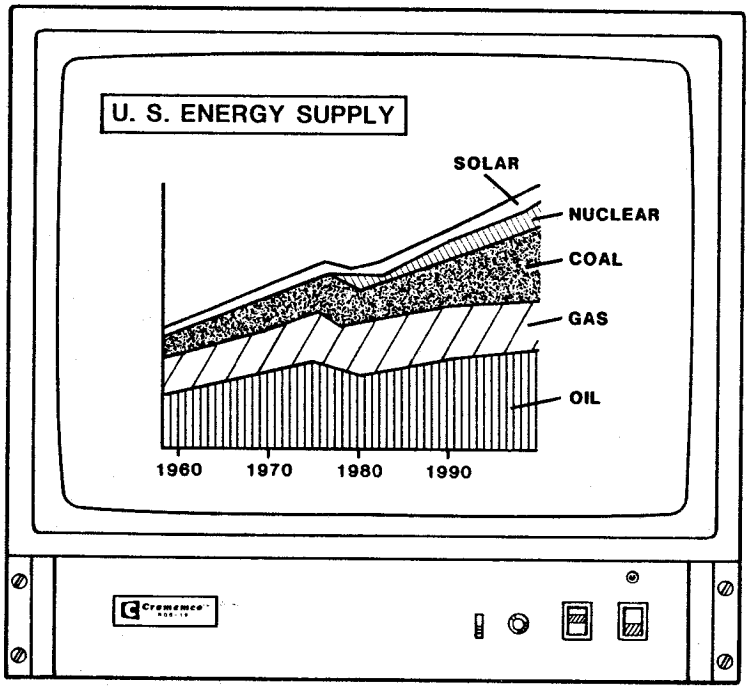
FORTRAN:	CALL WEXIT(p)
Assembly:	CALL WEXIT(p)
BASIC:	USR(316,79,p)
SBASIC:	.WEXIT (p)

where:

p = 0 or 1 to designate the page

This call exits the window mode and displays the designated page. All windows which have been opened are closed and can no longer be referenced. That is, there are no longer any windows designated. This call differs from WDISAB in that the WDISAB call closes the windows and they can be reopened with the WENAB call. The WEXIT deletes any record of the window ever having existed.

CALL WEXIT(1)



all windows closed,
window mode exited

Animation calls

These calls make it possible for the programmer to simulate motion by controlling which page of memory is displayed, by modifying one page of memory while displaying the second page, and by controlling which page of memory is affected by subsequent graphics calls. The calls associated with this mode of operation are contained in Table 2.3.

BASIC Call #	FORTRAN or Assembly Call	Argument(s)
(72)	AINIT	(p)
(n/a)	ANIM	(none)
(73)	ANIMAT	(none)

Table 2.3
Alphabetical List of the
Animation Housekeeping Calls

The BASIC call # applies to the 16K Extended version and the Structured version when SDILIB has not been included in the program.

Function: **A**nimation **INIT**ilization

format:

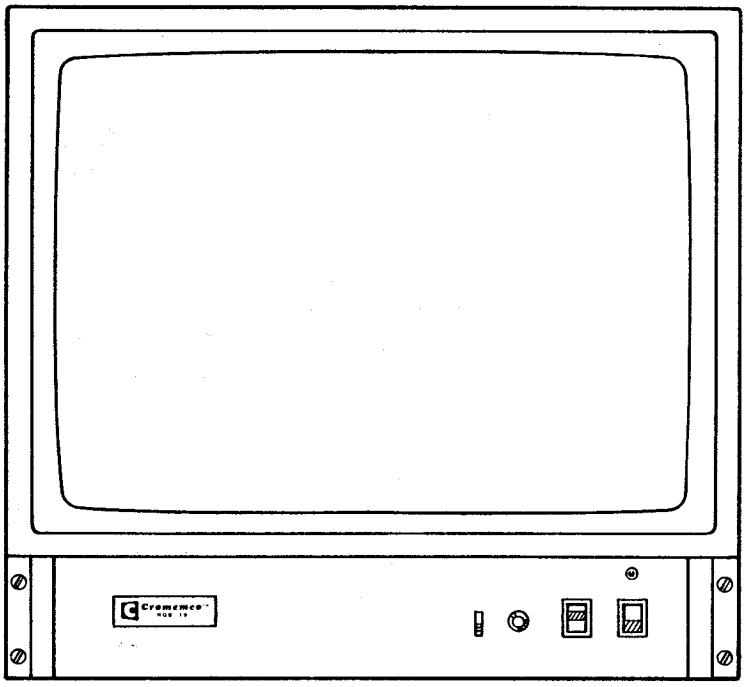
FORTRAN:	CALL AINIT(p)
Assembly:	CALL AINIT(p)
BASIC:	USR(316,72,p)
SBASIC:	.AINIT (p)

where:

p = 0 or 1 to designate the page

This call erases the designated page of Two Port memory, displays it, and makes it the work page. For example, assume that the programmer is working on page 1 while observing the effects of the calls. He can now clear, display, and draw on page 0 with the AINIT(0) call.

CALL AINIT(0)



Page 1 resolution and/or resboxes mode to match page 0

Page 0 cleared & displayed

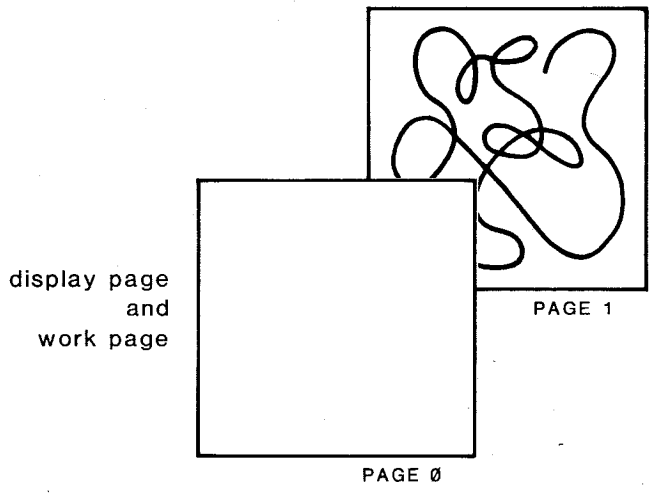
Function: **ANIMAT**ion

format:

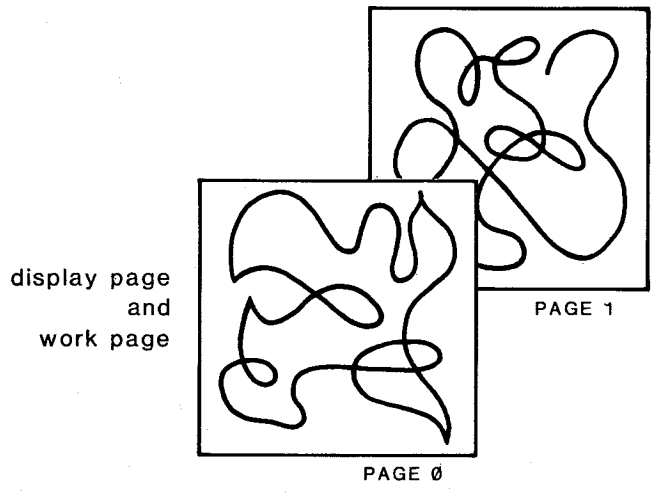
FORTRAN:	CALL ANIMAT
Assembly:	CALL ANIMAT
BASIC:	USR(316,73)
SBASIC:	.ANIMAT

This call toggles and clears the work page. If ANIMAT has been called previously it also toggles the display page.

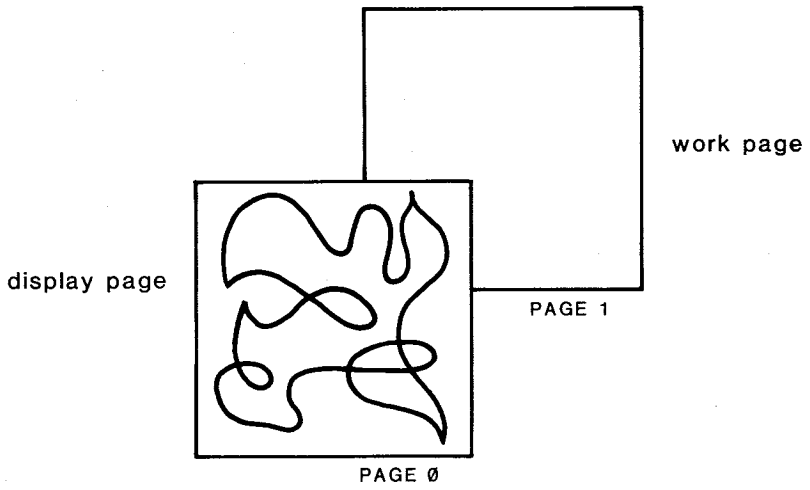
A. **CALL AINIT (0)**



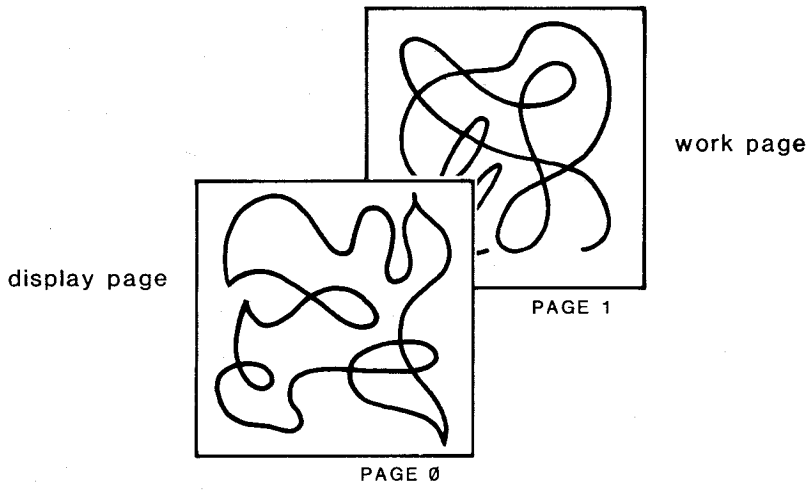
B. Draw



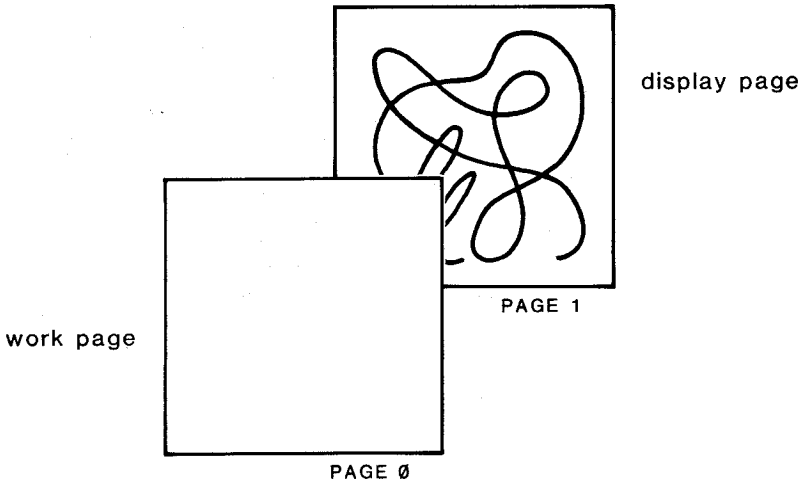
C. **CALL ANIMAT** (first time)



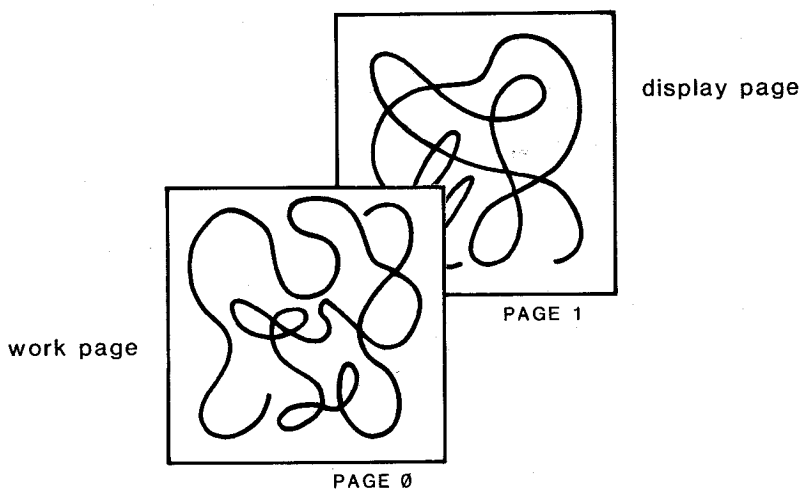
D. Draw



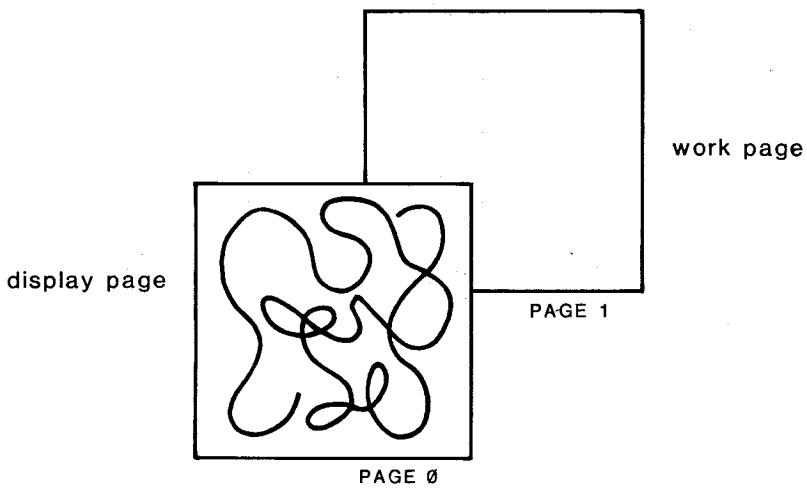
E. **CALL ANIMAT** (second time)



F. Draw



G. **CALL ANIMAT** (third time)



etc.

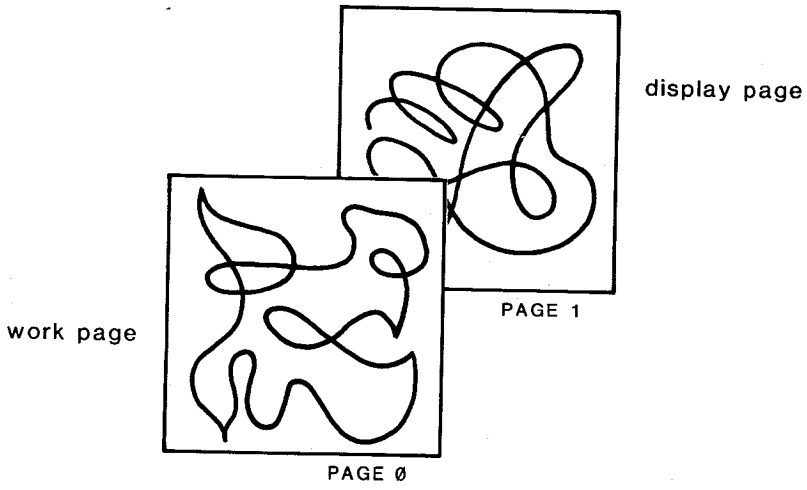
Function: small screen **ANIM**ation

format:

FORTRAN:	CALL ANIM
Assembly:	CALL ANIM
BASIC:	n/a
SBASIC:	n/a

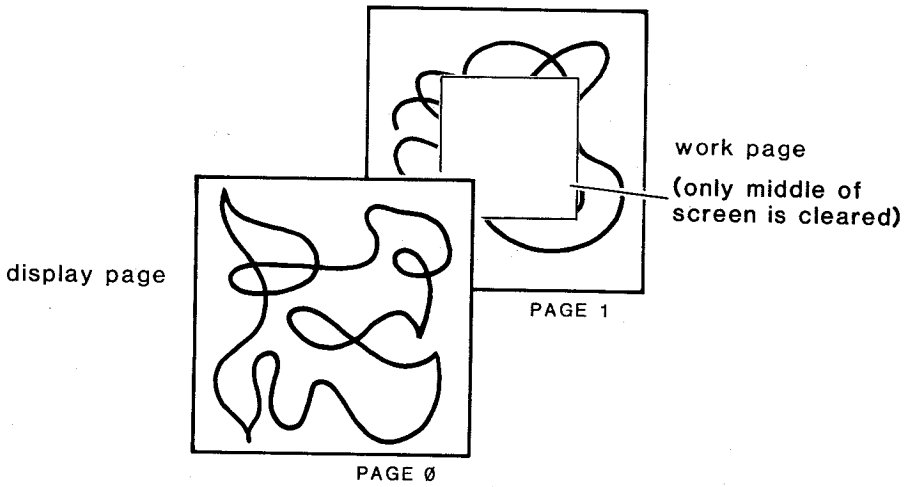
This call differs from ANIMAT in that only the center half of the screen is affected. There is a quarter width border on both pages which is not affected. The utility of ANIM is the speed advantage it has over ANIMAT.

Before Call



CALL ANIM

After Call



Wait Calls

These calls allow the programmer to control the display timing based on the timing characteristics of the video display signals. The calls available are listed in Table 2.4.

BASIC Call #	FORTRAN or Assembly Call	Argument(s)
(86)	WAITHG	(none)
(88)	WAITOD	(none)
(85)	WAITVG	(none)
(87)	WAITVS	(none)

Table 2.4
Alphabetical List of the
WAIT Housekeeping Calls

The BASIC call # applies to the 16K Extended version and the Structured version when SDILIB has not been included in the program.

Function: **WAIT** for **H**orizontal **G**ate pulse

format:

FORTRAN:	CALL WAITHG
Assembly:	CALL WAITHG
BASIC:	USR(316,86)
SBASIC:	.WAITHG

When called, WAITHG will go into a wait loop until the horizontal gate goes high. The horizontal gate waveform is available as bit 5 of input port 82H.

Function: **WAIT** for **OD**d display field

format:

FORTRAN:	CALL WAITOD
Assembly:	CALL WAITOD
BASIC:	USR(316,88)
SBASIC:	.WAITOD

When called, WAITOD will go into a wait loop until an odd field is detected. This corresponds to a bit 4 of input port 82H being high.

Function: **WAIT** for **V**ertical **G**ate pulse

format:

FORTRAN:	CALL WAITVG
Assembly:	CALL WAITVG
BASIC:	USR(316,85)
SBASIC:	.WAITVG

WAITVG causes the user's program to wait until the vertical gate signal goes high. This signal is available as bit 7 of input port 82H.

Function: **WAIT** for **V**ertical blanking **S**tart

format:

FORTRAN:	CALL WAITVS
Assembly:	CALL WAITVS
BASIC:	USR(316,87)
SBASIC:	.WAITVS

WAITVS causes the users program to remain in a wait condition until the start of the vertical blanking. More precisely, the call will return after the 3rd or 4th horizontal gate following the start of vertical blanking. This signal is available as bit 6 of input port 82H.

Chapter 3

SDI GRAPHICS CALLS

The reader should be aware that plots can be made in either the 2 color high resolution (755 x 482 points) mode or the 16 color medium resolution (378 x 241 points) mode. The programmer also has the choice of plotting explicitly (i.e., specifying within a call all needed location and color information) or implicitly (i.e., specifying needed location information with regard to an implied cursor). These options are discussed in more detail in the next two sections while the programmer's option to create solid filled figures is discussed in Section 3.3.

1 Explicit calls

In each of the explicit calls the programmer specifies the exact location on the screen at which the display is to occur in terms of the default scale (In the high resolution mode x varies between 2 and 755 while y varies between 1 and 482 with 1,1 designating the lower left corner of the screen.) or a scale designated using the call SCALE.
If the programmer is working in the 16 color medium resolution mode the color of the display must be specified in each call. An explicit call of particular interest is the read screen (XREAD). This call returns the color map code associated with a designated point on the screen. The explit plot calls are listed in Table 3.1.

BASIC Call #	FORTRAN or Assembly Call	Argument(s)
(n/a)	ESTRNG	("p")
(4)	XAREA	(x1,y1,x2,y2,c)
(5)	HXAREA	(x1,y1,x2,y2)
(6)	XCIRC	(x,y,r,c)
(7)	HXCIRC	(x,y,r)
(8)	XFCIR	(x,y,r,c)
(9)	HXFCIR	(x,y,r)
(3)	XDOT	(x,y,c)
(3)	HXDOT	(x,y)
(0)	XLINE	(x1,y1,x2,y2,c)
(1)	HXLINE	(x1,y1,x2,y2)
(16)	XPOLY	(c,a)
(14)	XREAD	(x,y,V)
(15)	HXREAD	(x,y,V)
(12)	XTEXT	(x,y,c,"text ^")
(13)	HXTEXT	(x,y,"text ^")

Table 3.1
List of the Explicit Graphics Calls

Descriptions of the high resolution calls have been omitted from the subsequent format pages since they are identical to the corresponding medium resolution calls except for the omission of the color parameter (c).

Function: eXplicit **DOT**

format:

FORTRAN:	CALL XDOT(x,y,c)
Assembly:	CALL XDOT(x,y,c)
BASIC:	USR(316,2,x,y,c)
SBASIC:	.XDOT (x,y,c)

where:

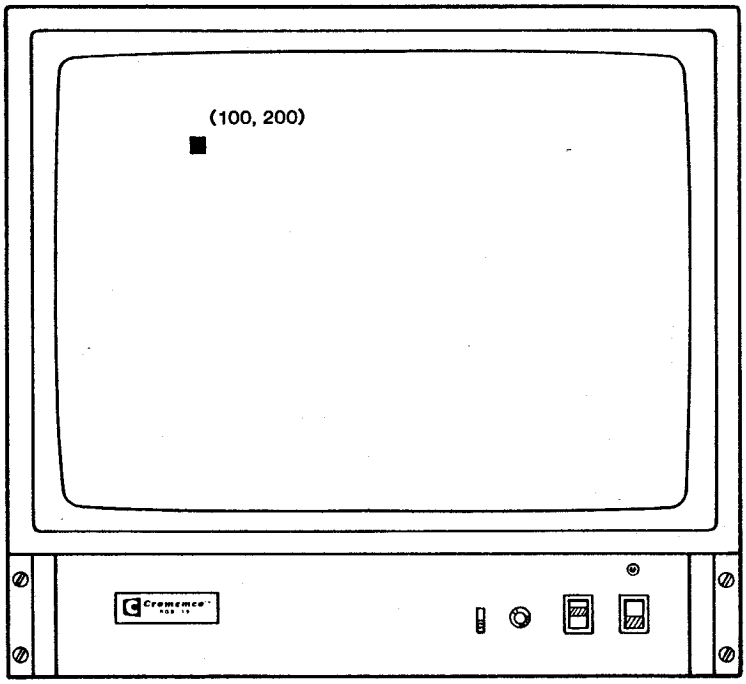
x = horizontal location on the screen
(1 ≤ x ≤ 378 for default scaling)

y = vertical location on the screen
(1 ≤ y ≤ 241 for default scaling)

c = color designation
(0 ≤ c ≤ 15)

This call results in a dot being written to the work page in the location and color specified.

CALL XDOT(100,200,4)



color represented by code 4
appears at x=100, y=200

Function: eXplicit **LINE**

format:

FORTRAN:	CALL XLINE(x1,y1,x2,y2,c)
Assembly:	CALL XLINE(x1,y1,x2,y2,c)
BASIC:	USR(316,0,x1,y1,x2,y2,c)
SBASIC:	.XLINE (x1,y1,x2,y2,c)

where:

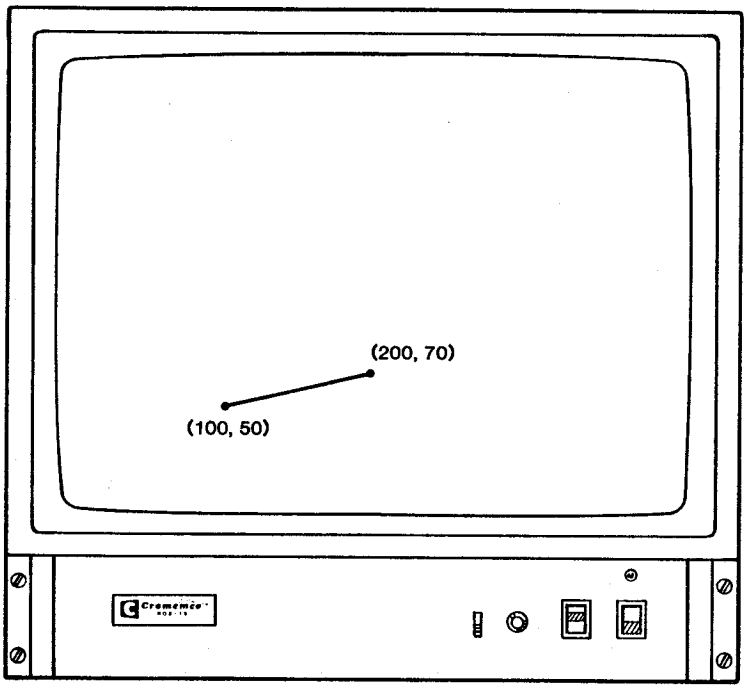
x1,y1 = starting point of the line

x2,y2 = ending point of the line

c = color designation

This call results in a line being drawn in the work page between the designated end points in the color indicated.

CALL XLINE(100,50,200,70,5)



line segment is color represented by code 5

Function: eXplicit **AREA**

format:

FORTRAN:	CALL XAREA(x1,y1,x2,y2,c)
Assembly:	CALL XAREA(x1,y1,x2,y2,c)
BASIC:	USR(316,4,x1,y1,x2,y2,c)
SBASIC:	.XAREA (x1,y1,x2,y2,c)

where:

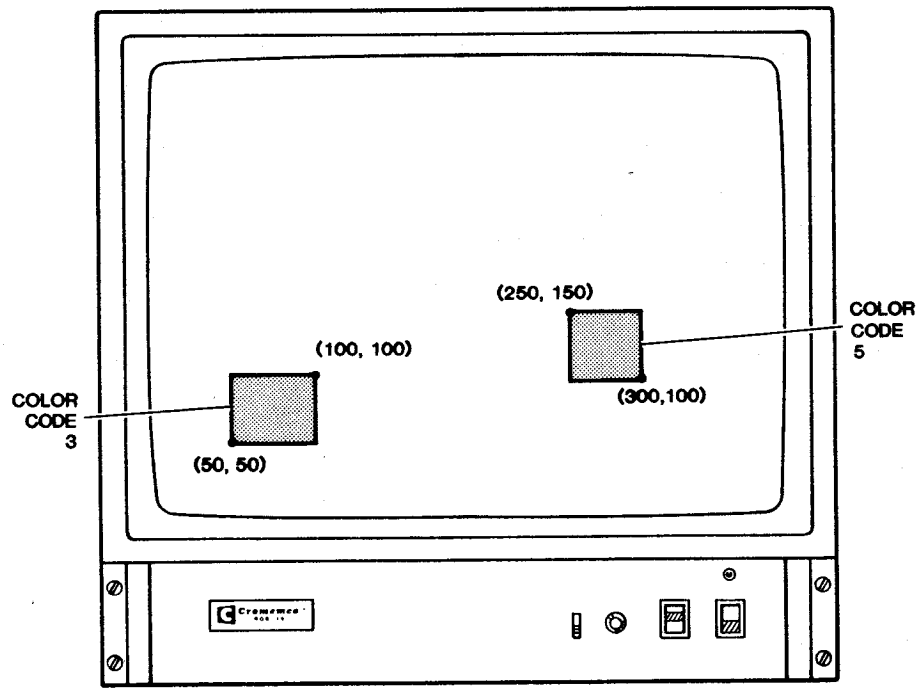
x1,y1 = coordinates of one corner of a rectangular area

x2,y2 = coordinates of the diagonally opposite corner of the rectangular area

c = color

This call results in a solid rectangle being written to the work page in the area and color designated.

CALL XAREA(50,50,100,100,3)
CALL XAREA(300,100,250,150,5)



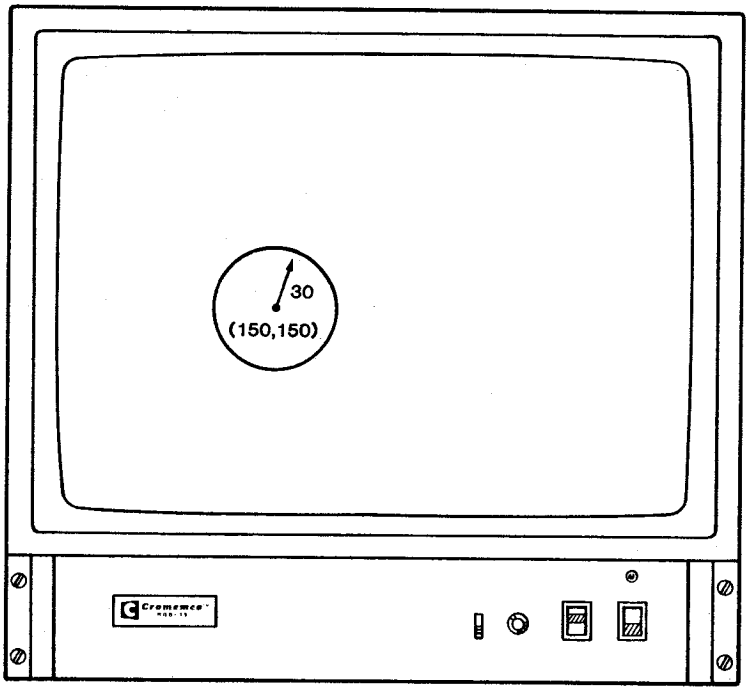
Function: e**X**plicit **CIRC**le

format:

FORTRAN:	CALL XCIRC(x,y,r,c)
Assembly:	CALL XCIRC(x,y,r,c)
BASIC:	USR(316,6,x,y,r,c)
SBASIC:	.XCIRC (x,y,r,c)

This call results in a circle of radius r whose center is located at horizontal position x and vertical position y being drawn to the work page. The color of the circle is designated by the parameter c.

CALL XCIRC(150,150,30,8)



circle is drawn in color
represented by code 8

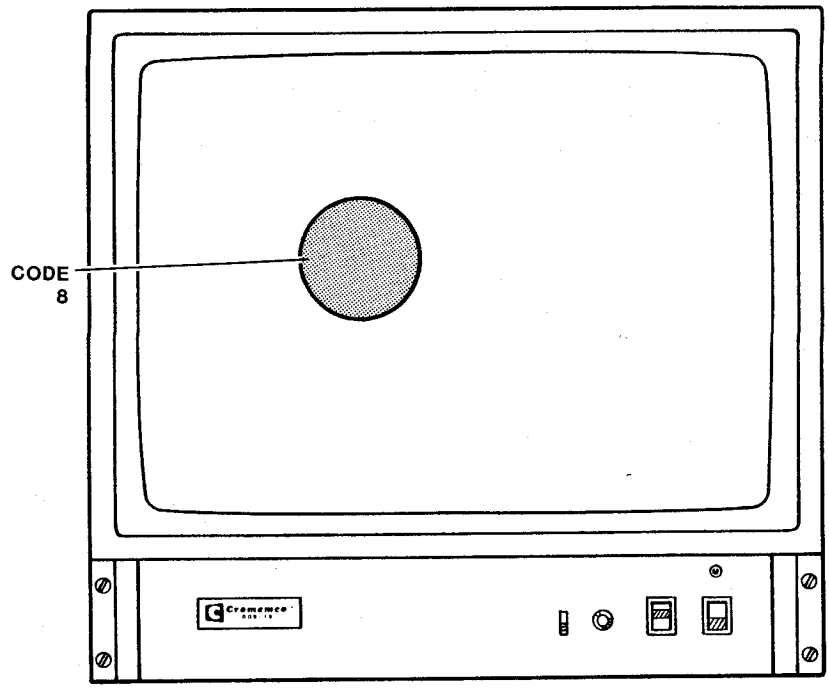
Function: e**X**plicit **F**illed **CIR**cle

format:

FORTRAN:	CALL XFCIR(x,y,r,c)
Assembly:	CALL XFCIR (x,y,r,c)
BASIC:	USR(316,8,x,y,r,c)
SBASIC:	.XFCIR (x,y,r,c)

This call produces the same results as XCIRC with the exception that the total area of the circle is in the designated color. Note that this is not an auto-fill call.

CALL XFCIR(150,150,30,8)



center is (150,150) and radius is 30

Programming Example 1.
The following example illustrates the use of some of the graphics calls in both the high resolution and the medium resolution mode.

```
# Program : MODCMP
# Purpose : To compare hi-res and med-res mode.

        call    grafix
        call    init

# Redefine color map
        call    defclr(10,15,15,0)
        call    defclr(11,15,10,0)
        call    defclr(12,10,15,0)
        call    defclr(13,0,4,15)
        call    defclr(15,0,15,4)

# Med-res mode
        call    xcirc(40,210,20,9)
        call    xline(50,150,150,230,10)
        call    xfcir(200,210,25,11)
        call    xarea(240,190,280,220,12)
        call    xtext(100 ,150,13,'MED-RES mode^')

# Hi-res mode
        call    resbox(1,1,24,120)      #y1,y2 follow "scale"
        call    hxcirc(40,90,20)
        call    hxline(50,30,150,110)
        call    hxfcir(200,90,25)
        call    hxarea(240,70,280,100)
        call    hxtext(100 ,30 ,'HI-RES mode^')
END
```

Function: e**X**plicit **POLY**gon

format:

FORTRAN:	CALL XPOLY(c,a)
Assembly:	CALL XPOLY(c,a)
BASIC:	USR(316,16,c,a)
SBASIC:	.XPOLY (c,a)

where:

c = color map code (0-15)

a = an array containing as its first element the number of line segments making up the polygon. Subsequent elements of the array containing the x and y components of the polygon's corners.

XPOLY can be used to draw a polygon by supplying its vertices in an array 'A' and issuing a single call to XPOLY. The format of the array 'A' is:

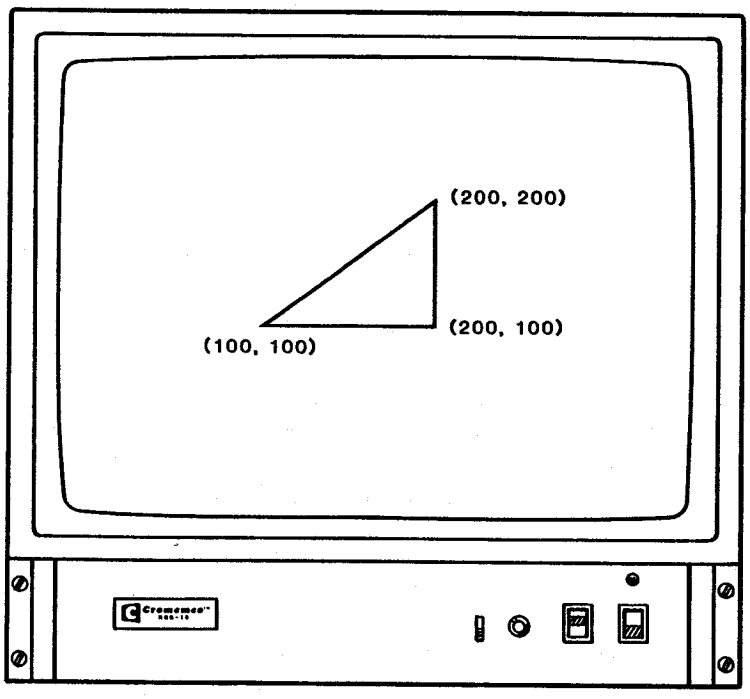
Array	Comment
A(1) = n	number of vertices
A(2) = x(1)} A(3) = y(1)}	vertex #1
A(4) = x(2)} A(5) = y(2)}	vertex #2
.	
.	
.	
A(2n) = x(n) } A(2n+1) = y(n)}	vertex #n

EXAMPLE:

To draw a high resolution triangle having corners at x=100, y=100; x=200, y=200; and x=200, y=100 we would use the following sequence:

```
INTEGER A(7)
DATA A/3,100,100,200,200,200,100/
CALL XPOLY(3,A)
```

(Please see diagram on next page.)



Programming Example 2.
The use of the call XPOLY and its high resolution counter part HXPOLY are demonstrated in the following program.

```
# Program : HPOLY
# Purpose : To use the routines: HXPOLY/XPOLY
#
# Remark  : HXPOLY is the hi-res version of XPOLY.
#           The hi-res mode is not compatible with the auto-fill mode,
#           i.e., it cannot automatically fill in the polygon in hi-res mode.

        integer a(20),b(10)
        data    a       /9,47,180,132,180,132,140,126,140,
                         136,120,136,210,164,210,164,120,132,30/

        data    b       /4,190,180,218,180,
                         164,30,136,30/


        call    grafix
        call    init
        call    initl                           # set up pagel
        call    defclr(8,15,10,0)               # redefine some colors
        call    defclr(15,0,8,15)

# Using XPOLY, draw  polygons
        call    xpoly(8,a)
        call    xpoly(8,b)

# Using HXPOLY, draw hi-res polygons
        call    scale(-400,250,-100,450)
        call    resbox(15,1,24,450)
        call    hscale(-400,250,-100,450)
        call    hxpoly(a)
        call    hxpoly(b)
        end
```

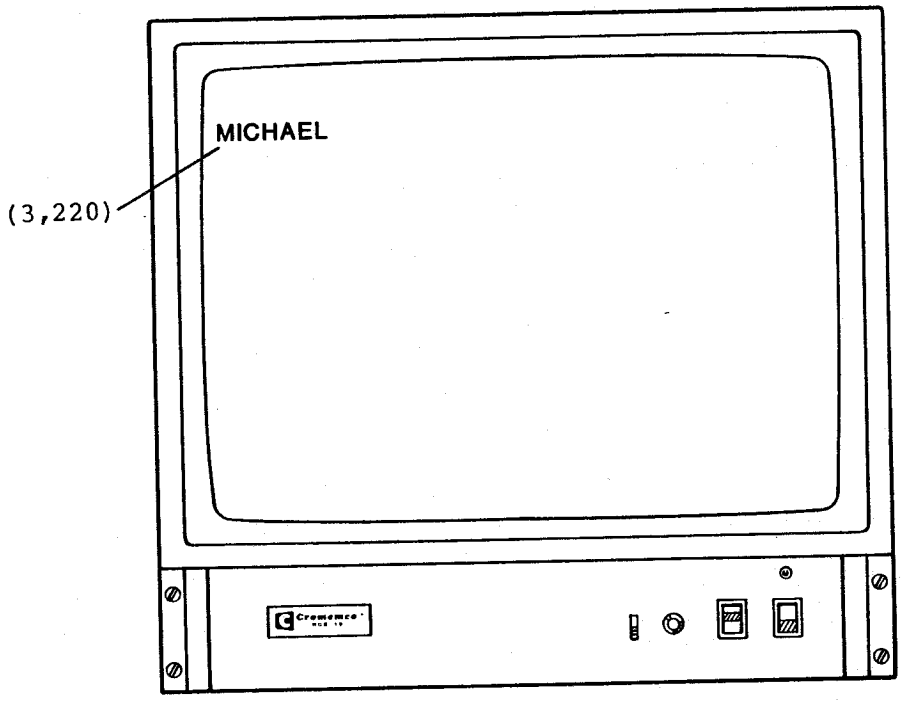
Function: eXplicit **TEXT**

format:

FORTRAN:	CALL XTEXT(x,y,c,"text.^")
Assembly:	CALL XTEXT(x,y,c,"text.^")
BASIC:	See next page
SBASIC:	See next page

This call results in the text contained within the quotation marks and terminated by the delimiter "^" to be printed to the work page with the lower left corner of the text beginning at the x (horizontal) and y (vertical) position and in the color specified by c. The delimiter symbol can be changed using the FORTRAN or Assembly call ESTRNG (" ")where the symbol inserted between the quotation marks specifies the new delimiter.

CALL XTEXT(3,220,15,"MICHAEL^")



color code is 15

Function: eXplicit **TEXT**

format:

FORTRAN:	See previous page
Assembly:	See previous page
BASIC:	USR(316,12,x,y,c,T1)
SBASIC:	.XTEXT (x,y,c,T$)

This call works in a slightly different manner than the other BASIC graphics calls. The user must first dimension a variable sufficient to hold the text to be printed. (I.e., DIM T$(16) where T$ is the phrase to be printed: "This is a test.^" and is 17 characters long). Then the actual plot command consists of three sequential instructions. This is illustrated in the following example.

EXAMPLE:

```
1000 DIM T$(47)
1010 T$="Cromemco's computers have many applications^"
  .
  .
  .
3040 T=LEN(T$)
3050 T1=SYS(2)
3060 X=USR(316,12,100,50,10,T1)
  .
  .
  .
5090 END
```

The above program as well as the two Structured BASIC examples below will print the phrase:

Cromemco's computers have many applications

starting at point x=100, y=50 in the color designated by location 10 in the color map. In 32K structured BASIC lines 3040 through 3060 would be replaced with the following lines:

```
3040 T=ADR(T$)
3050 X=USR(316,12,100,50,10,T)
```

If the SDI library package SDILIB has been incorporated into your Structured BASIC program than lines 3040 through 3060 can be replaced by the single line:

```
3040 .XTEXT (100,50,10,T$)
```

Programming Example 3.
The following example illustrates the use of graphics calls XLINE, XTEXT, and XAREA (and their high resolution counterparts) as well as the housekeeping calls ERABOX, RESBOX, SCALE, and HSCALE.

```
# Program : BOX
# Purpose : To illustrate the use of the subroutines:
#                       RESBOX (opposite to original res-mode) and
#                       ERABOX (back to original res-mode).


        call    grafix
        call    init                                    # entire screen in med-res
        call    defclr(15,0,15,15)

# Define resboxes
        call    scale(-400,400,-300,300)
        n1=-300
        n2=-261
        for (iarea=1;iarea<=5;iarea=iarea+1)
                {
                call    xarea(n1,-300,n2,300,iarea)
                n1=n1+40
                n2=n2+40
                }
        call    xtext(-280,-290,15,'MED^')

        call    resbox(13,-300,24,300)                  # half-screen in hi-res
        call    hscale(-400,400,-300,300)
        call    hxtext(80,-210,'HI-RESOLUTION^')
        call    hxtext(80,-230,'  characters^')

        call    scale(-400,400,-300,300)
        call    erabox(19,-100,23,280)                  # box in med-res
        call    xarea(270,-100,330,280,2)
        call    xtext(290,220,8,'M^')
        call    xtext(290,190,8,'E^')
        call    xtext(290,160,8,'D^')
        call    xtext(290,100,8,'E^')
        call    xtext(290,70 ,8,'R^')
        call    xtext(290,40,8,'A^')
        call    xtext(290,10,8,'B^')
        call    xtext(290,-20,8,'O^')
        call    xtext(290,-50,8,'X^')

        call    erabox(13,-155,18,155)                  # box in med-res
        ix1=150
        ix2=200
        ix3=-150
        ix4=-200
        repeat
                {
                call    xline (0,ix1,ix2,0,9)
                call    xline (ix2,0,0,ix3,9)
                call    xline (0,ix3,ix4,0,9)
                call    xline (ix4,0,0,ix1,9)
                ix1=ix1-10
                ix2=ix2-10
                ix3=ix3+10
                ix4=ix4+10
                } until (ix1<=0)

        call    resbox(9 ,179,19,211)                   # set box in hi-res
        call    hscale(-400,400,-300,300)
        call    hxarea(-138,179 ,201,211)

        end
```

Function: End of **STRiNG**

format:

FORTRAN:	CALL ESTRNG("p")
Assembly:	CALL ESTRNG("p")
BASIC:	n/a
SBASIC:	n/a

where: p is any delimiter character

This call specifies a new symbol to be used as a delimiter character (in place of ^) in the TEXT calls.

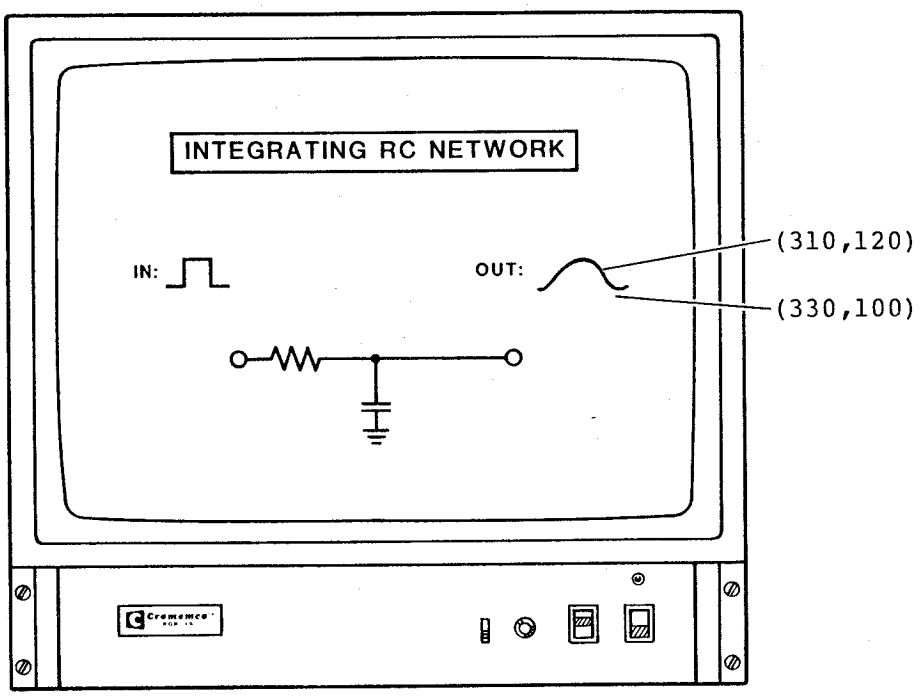
Function: e**X**plicit **READ**

format:

FORTRAN:	CALL XREAD(x,y,v)
Assembly:	CALL XREAD(x,y,v)
BASIC:	n/a
SBASIC:	n/a

This call will set the variable, v, to a value between 0 and 15 corresponding to the color read from the current work page at the location designated by x (horizontal) and y (vertical). The variable v must be declared as an integer.

```
CALL XREAD(310,120,N)
CALL XREAD(330,100,M)
```



Assuming a background of color code 0 and a plottig color code of 12, the first call to XREAD will assign the value 12 to variable N and the second call will assign the value 0 to variable M.

Programming Example 4.
The following subroutine, HTURN, can be used to write text rotated at any angle on page 0. The subsequent program makes use of this subroutine to interactively demonstrate the many choices of angles available for text rotation. The key to character rotation is first writing the text string on page 1 horizontally and then reading (XREAD), rotating, and writing the string pixel by pixel onto page 0.

```
# Purpose : to rotate a string at an arbitrary angle and write it
#           at a specified location on page0.
#
# Input   :      * (x,y) -desired starting location of string
#                * itheta  -angle of rotation in degrees
#                * a hi-res string starting at (1,4) on page1
#                    is assumed, prior to calling routine HTURN.
#
# Output  : a rotated & translated  hi-res string.


        Subroutine HTURN (x,y,itheta)
        Integer pix,ver,hor,newx,newy,x,y,itheta

# Define TRIG. functions
        theta =itheta
        cosa = cos (3.1415927/180.* theta)
        sina = sin (3.1415927/180.* theta)

#Character rotation
        for  (hor=1;hor<=754;hor=hor+1)
           {
           for (ver=1;ver<=18;ver=ver+1)
                {
                call    workon(1)
                call    hxread (hor,ver,pix)
                if (pix>0)
                   {
                   # Convert to real
                   xhor =hor
                   xver =ver

                   # Rotation
                   newx = xhor  * cosa - xver  * sina
                   newy = xhor  * sina + xver  * cosa

                   # Translation
                   newx = newx+x
                   newy = newy+y

                   # Output
                   call workon(0)
                   call unscal
                   call hxdot(newx,newy)
                   }
                }# end of vertical scan
           }# end of horizontal scan
        return
        end
```

Programming Example 5.

```
# Program to illustrate an interactive example of rotated text.
# User responds to prompt with : x,y,angle.

        call    grafix
        call    init
        call    res(1)
        call    workon(1)
        call    page
        call    res(1)

# Input HI-RES text
        call    hxtext(1,4,'A ROTATED string at any angle^')

# Input data
5       write (3,10)
10      format('0','ix=xxxb ','iy=xxxb ','itheta=xxxb'/)
        read (3,20)ix,iy,itheta
20      format (3(i3,1x))

# Rotate the string by calling routine HTURN
        call    hturn(ix,iy,itheta)
        call    workon(0)
        call    page                    # clear page0
        goto 5
        end
```

3.2 Implicit calls

The location and color of the displays produced using the implicit calls are related to an implied cursor whose position and color is determined by the CURSOR and COLR calls respectively. The default cursor position is at the screen center. These calls draw a line from the cursor to some other point, plot a point, line, or text at the cursor etc. Some commands move the cursor while others leave it at its pre-command position. The pages of this manual covering the individual commands indicate which commands perform which actions. One implicit call, because of its usefulness and uniqueness, deserves special note. It is the relative line call (RLINE) and is used to specify the x and y component of a line's length. Using this call a line is plotted with respect to the implied cursor.

BASIC Call #	**FORTRAN** or **Assembly Call**	**Argument(s)**
(45)	AREA	(x,y)
(46)	HAREA	(x,y)
(32)	COLR	(c)
(33)	CURSOR	(x,y)
(34)	HCRSOR	(x,y)
(37)	DOT	(none)
(38)	HDOT	(none)
(35)	LINE	(x,y)
(36)	HLINE	(x,y)
(19)	RLINE	(x,y)
(20)	HRLINE	(x,y)
(none)	READ	(x,y,V)
(47)	TEXT	("text ^")
(48)	HTEXT	("text ^")

Table 3.2
List of the Implicit Graphics Calls

The following pages contain descriptions of each of the implicit graphics calls. Descriptions of the corresponding high resolution calls have been omitted since these calls are identical to the medium resolution calls except for the omission of any color parameter (c).

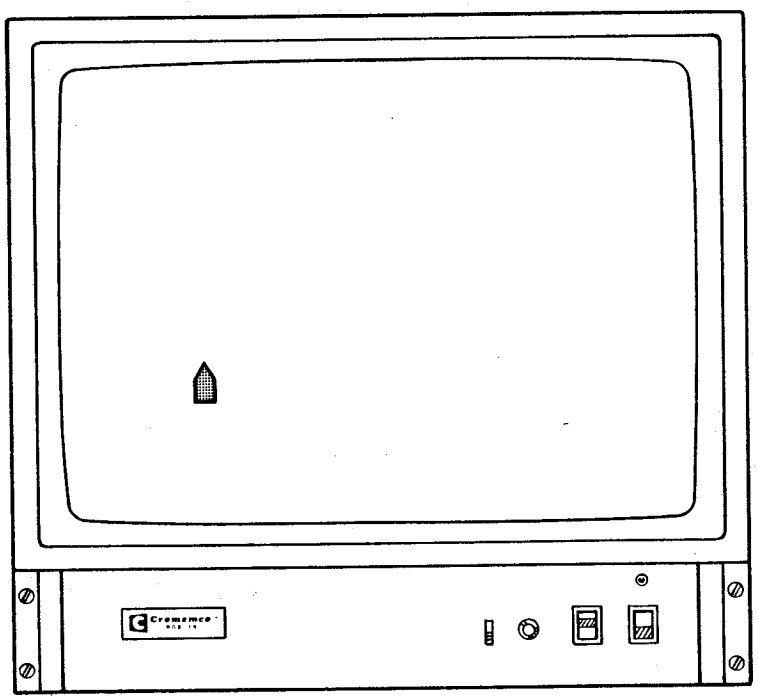
Function: set **CURSOR** location

format:

FORTRAN:	CALL CURSOR(x,y)
Assembly:	CALL CURSOR(x,y)
BASIC:	USR(316,33,x,y)
SBASIC:	.CURSOR (x,y)

This call is used with the implicit plot calls to position the implied cursor at the x (horizontal) and y (vertical) position designated on the work page. The location of the cursor on one page in no way affects the position of the cursor on the other page.

CALL CURSOR(100,100)



no change in screen

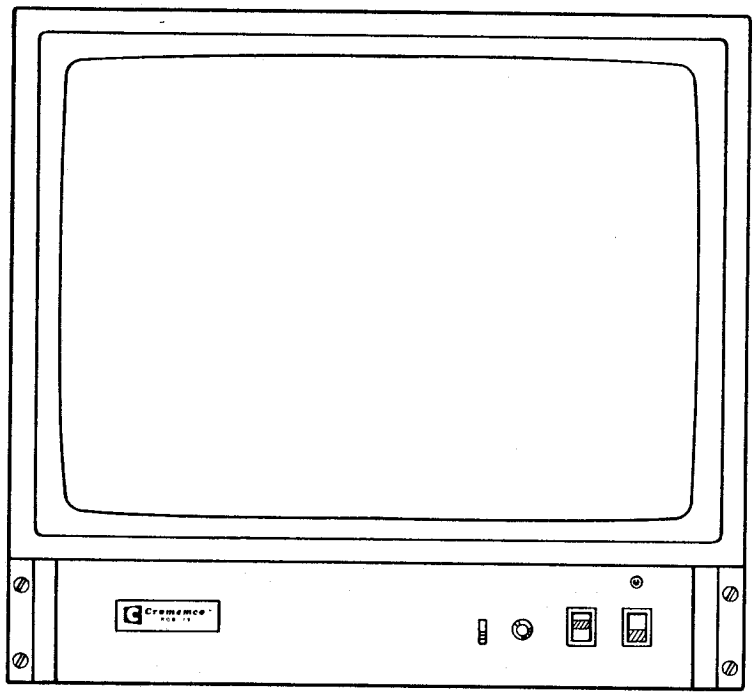
Function: set cursor **COLoR**

format:

FORTRAN:	CALL COLR(c)
Assembly:	CALL COLR(c)
BASIC:	USR(316,32,c)
SBASIC:	.COLR (c)

This call is used with the implicit plot calls to set the cursor color on the active work page. The color of the cursor on one page in no way affects the color of the cursor on the other page.

CALL COLR(15)



no change in screen

Function: **DOT**

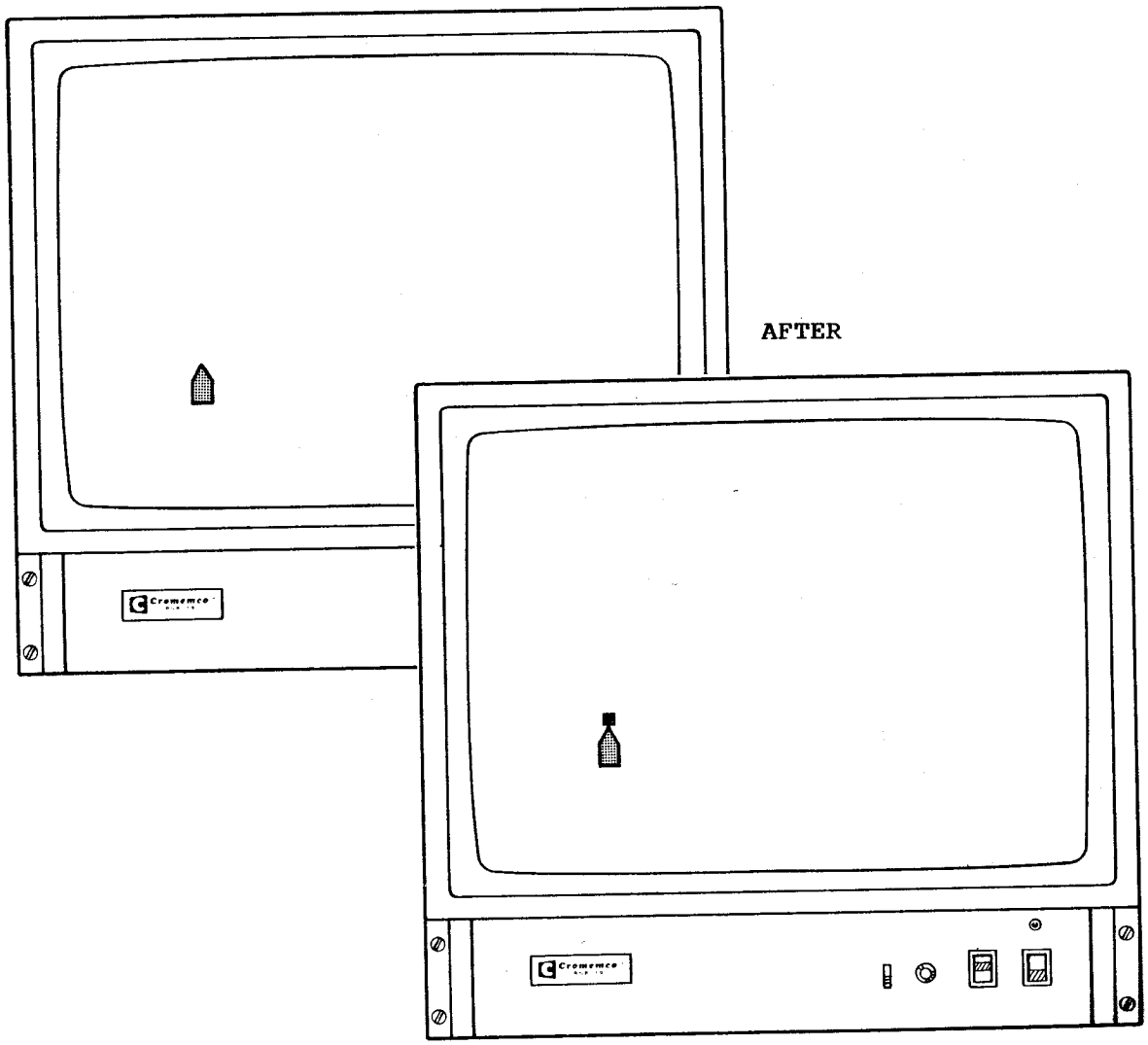
format:

FORTRAN:	CALL DOT
Assembly:	CALL DOT
BASIC:	USR(316,37)
SBASIC:	.DOT

This call plots a single dot on the work page at the location and in the color implied by the cursor.

CALL DOT

BEFORE



color of DOT determined
by most recent call to colr( ),
location determined by cursor

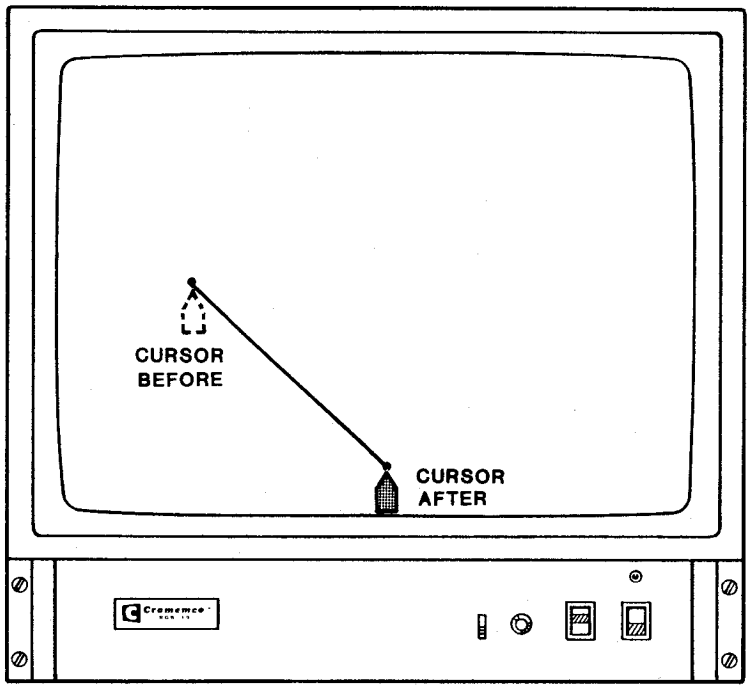
Function: **LINE**

format:

FORTRAN:	CALL LINE(x,y)
Assembly:	CALL LINE(x,y)
BASIC:	USR(316,35,x,y)
SBASIC:	.LINE (x,y)

This call plots a line in the work page in the color of the implied cursor. The line is drawn from the location of the implied cursor to the point designated by x (horizontal) and y (vertical). Then the location of the implied cursor is moved to the point designated by x and y.

CALL LINE(200,10)



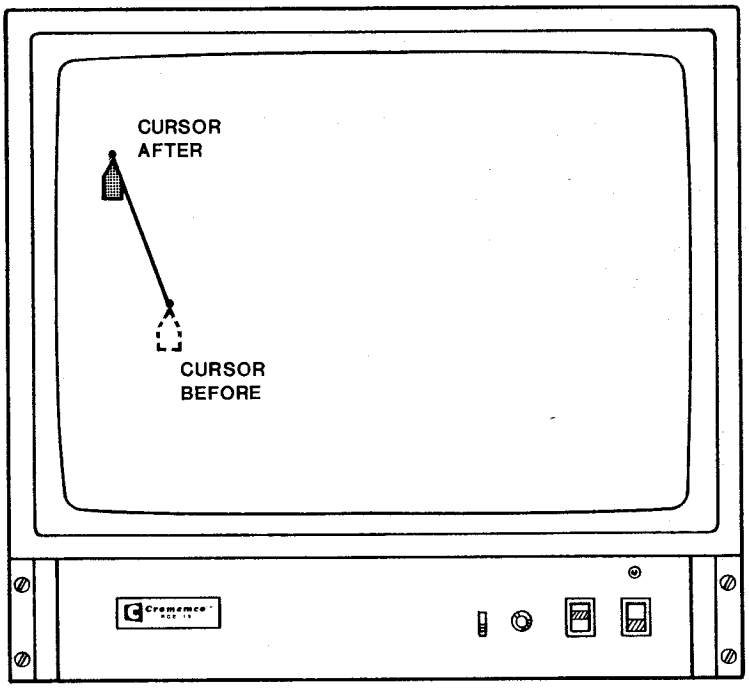
Function: Relative **LINE**

format:

FORTRAN:	CALL RLINE(x,y)
Assembly:	CALL RLINE (x,y)
BASIC:	USR(316,19,x,y)
SBASIC:	.RLINE (x,y)

This call draws a line on the work page in the color of the implied cursor. If the cursor's horizontal and vertical position is designated by x0 and y0 respectively than the line will go from the position x0,y0 to the position x0+x,y0+y. The location of the implied cursor will be updated to x0+x,y0+y.

CALL RLINE(-10,100)



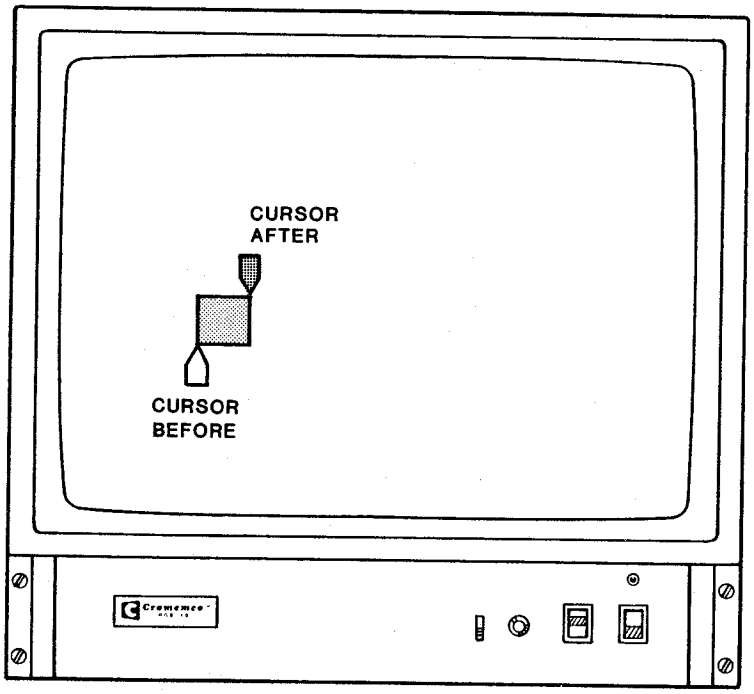
Function: **AREA**

format:

FORTRAN:	CALL AREA(x,y)
Assembly:	CALL AREA(x,y)
BASIC:	USR(316,45,x,y)
SBASIC:	.AREA (x,y)

This call plots a solid rectangle in the work page in the color implied by the cursor. One corner of the rectangle is specified by the cursor and the diagonal corner of the rectangle is designated by the x (horizontal) and y (vertical) coordinates specified in the call. The cursor is then moved to the x,y point designated in the call.

CALL CURSOR(100,100)
CALL AREA(120,120)



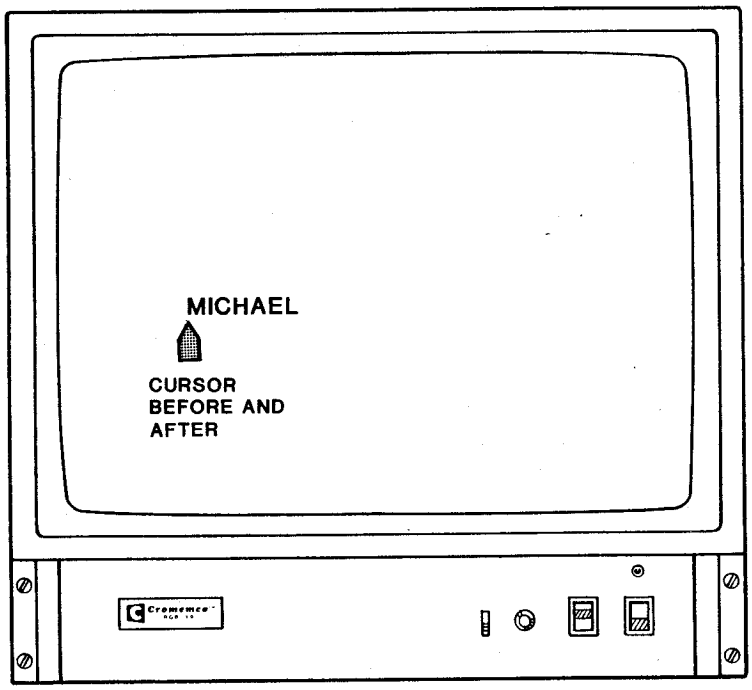
Function: print **TEXT**

format:

FORTRAN:	CALL TEXT("text ^")
Assembly:	CALL TEXT("text ^")
BASIC:	See next page
SBASIC:	See next page

This call prints the text contained within the quotation marks and terminated with the caret delimiter "^" The text starts on the work page with the lower left corner of the text located at the cursor. The color of the text is the color associated with the implied cursor. If no scaling is in force then the location of the implied cursor is updated to the lower right corner of the text. If scaling is in force then the cursor remains at its position prior to the call.

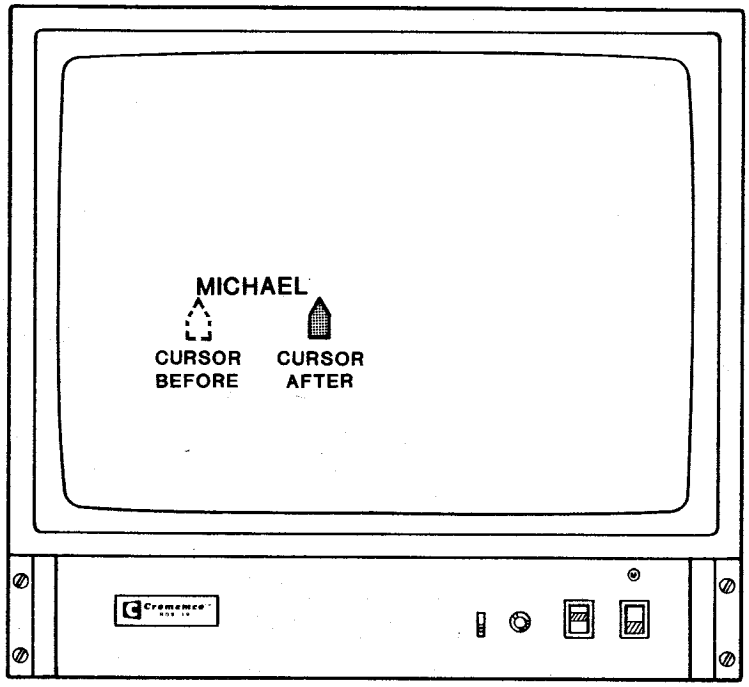
```
CALL SCALE(0,200,0,200)
CALL TEXT("Michael^")
```



in scaled mode

(Please see diagram on next page.)

```
CALL UNSCALE
CALL TEXT("Michael^")
```



in unscaled (default) mode

Function: print **TEXT**

format:

FORTRAN:	See previous page
Assembly:	See previous page
BASIC:	USR(316,47,T1)
SBASIC:	.TEXT (T$)

This call works in a slightly different manner than the other BASIC graphics calls. The user must first dimension a variable sufficient to hold the text to be printed. (I.e. DIM T$(13) where T$ is the phrase to be printed: "This is text.^" and is 14 characters long). Then the actual plot command consists of three sequencial instructions. This is illustrated in the following example.

EXAMPLE:

```
1000 DIM T$(27)
1010 T$="There are many things ...  ^"
  .
  .
  .
3040 T=LEN(T$)
3050 T1=SYS(2)
3060 X=USR(316,47,T1)
  .
  .
  .
5090 END
```

The above program will print the phrase:

There are many things ...

starting at the location of the cursor in the designated cursor color. In 32K Structured BASIC lines 3040 through 3060 would be replaced with the following:

```
3040 T=ADR(T$)
3050 X=USR(316,47,T)
```

If the SDI library package SDILIB has been incorporated into your Structured BASIC program, then lines 3040 through 3060 can be replaced by the single line:

```
3040 .TEXT (T$)
```

Function: **READ** the screen

format:

FORTRAN:	CALL READ(x,y,v)
Assembly:	CALL READ(x,y,v)
BASIC:	n/a
SBASIC:	n/a

Prior to this call, x, y, and v must be declared as integer variables. Upon return from the READ call the parameters x and y are assigned the coordinate value of the cursor and the parameter v is assigned the color map code associated with the cursor's position on the current work page.

Programming Example 6.
The two programs which follow are examples of the implicit graphics calls. Note that the second program uses scaling while the first program doesn't. Note also that SCALE has different effects on the cursor after a TEXT call.

```
# Program : CURSOR1
# This program contains some cursor related routines, without scaling.

        CALL    GRAFIX
        CALL    INIT
        CALL    DEFCLR(8,12,10,0)

        CALL    COLR(8)
        CALL    CURSOR(150,120)
        CALL    LINE(150,200)
        CALL    TEXT('TEXT WITHOUT SCALING^')
        CALL    LINE(1,1)               # when scaling is not in force,
                                        # text updates cursor.

        END
```

Programming Example 7.

```
# Program : CURSOR
# This program contains some cursor related routines, with scaling.

        CALL    GRAFIX
        CALL    INIT
        CALL    DEFCLR(8,12,10,0)
        CALL    DEFCLR(13,0,15,4)
        CALL    SCALE(-200,200,-150,150)

        CALL    COLR(8)
        CALL    CURSOR(10,10)
        CALL    LINE(10,100)
        CALL    TEXT('TEXT WITH SCALING^')
        CALL    LINE(-200,-150)         # when scaling is in force,
                                        # text does not update cursor.

        FOR (I=-15;I<=15;I=I+3)
                {
                CALL    COLR(6)
                CALL    CURSOR(I,0)
                CALL    DOT
                }

        CALL    COLR(15)
        CALL    CURSOR(100,-70)
        CALL    RLINE(25,25)
        CALL    RLINE(25,-25)
        CALL    RLINE(-25,-25)
        CALL    RLINE(-25,25)

        CALL    COLR(13)
        CALL    CURSOR(125,-45)
        CALL    LINE( 65,15)
        CALL    LINE(125, 75)
        CALL    LINE(185,15)
        CALL    LINE(125,-45)

        END
```

Programming Example 8.
This program demonstrates how (by using the RLINE and RHLINE calls) several copies of the same figure can be drawn on different parts of the screen in different sizes.

```
# This is a high resolution cursor routine
# which makes use of the relative line.

        call    grafix
        call    init
        call    initl           #set up pagel ( work/disp/page/color )
        call    defclr(15,0,15,5 )
        call    res(1)
        call    house(30,30)
        call    hscale(-40 ,20 ,-20,20)
        call    house(-20,0)
        call    hscale(-300,300,-300,300)
        call    house(250,250)
        call    hscale(-200,200,-100,100)
        call    house(-150,20)
        end

Subroutine house(m,n)
        # Comment : draws a house
        integer m,n
        call    cursor(m,n)
        call    rhline(15,0)
        call    rhline(0,-20)
        call    rhline(-15,0)
        call    rhline(0,20)
        call    rhline(7,15)
        call    rhline(8,-15)
        return
        end
```

3.3 Fill Mode Calls

It is possible to construct solid (i.e., filled in) objects in the same time required to draw their outline by taking advantage of the auto-fill mode of plotting. There are certain graphics calls (listed below) which are designed especially for this mode. This mode requires special attention as the standard graphics calls (described in other sections) will have unpredictable effects during auto-fill. Examples at the end of this section illustrate the combined use of normal and auto-fill graphics calls in the auto-fill mode of plotting. The calls which utilize this auto-fill mode are listed in Table 3.3.

BASIC Call #	**FORTRAN** or **Assembly Call**	**Argument(s)**
(66)	AFILL	(none)
(65)	FILINT	(none)
(31)	FSEG	(x,y)
(1)	XFPOLY	(c,a)
(17)	HXPOLY	(a)
(67)	XFILL	(none)

Table 3.3
Alphabetical List of the auto-fill Calls

The BASIC call # applies to the 16K Extended version and the Structured version when SDILIB has not been included in the program.

Function: **FIL**l mode **IN**i**T**alization

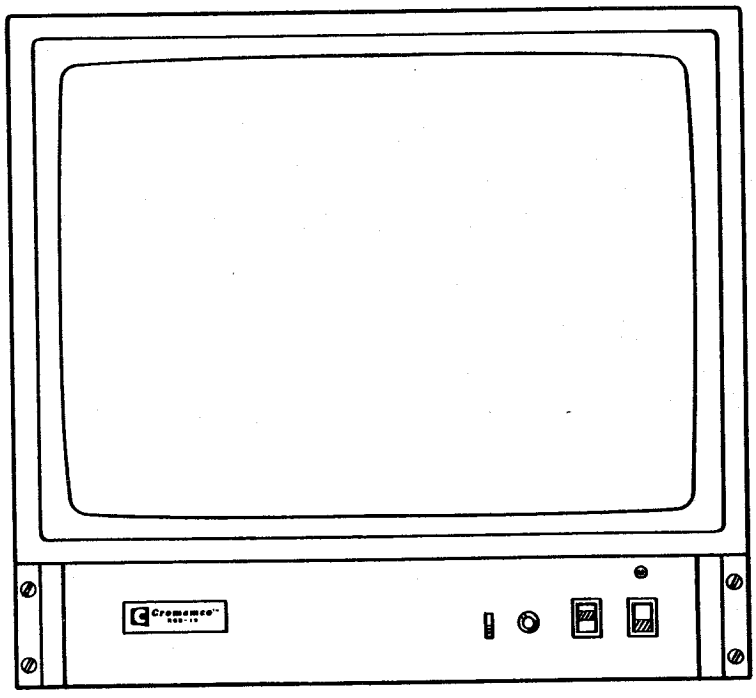
format:

FORTRAN:	CALL FILINT
Assembly:	CALL FILINT
BASIC:	USR(316,65)
SBASIC:	.FILINT

This command is normally used in preparation for drawing solid figures. The operations performed by this call are:

- calls the subroutine INIT
- sets color 15 of the color map to black
- calls the subroutine AFILL

CALL FILINT



Screen is cleared and default conditions are set as in INIT. Auto fill mode enabled.

Function: **Filled line SEG**ment

format:

FORTRAN:	CALL FSEG(x,y)
Assembly:	CALL FSEG(x,y)
BASIC:	USR(316,31,x,y)
SBASIC:	.FSEG (x,y)

In the auto-fill mode the command is used to specify the segments which define a polygon that is to be displayed as a solid figure. The segments must be specified in a clockwise order. Note that FSEG is a cursor related call and as such requires a prior call to CURSOR and COLR. Also, the polygon may not overlap a polygon which was previously drawn using FSEG. If auto-fill is disabled, or if FSEG is called when in normal (i.e., non-auto-fill) mode the screen will show portions of the polygon's perimeter only.

EXAMPLE:

To draw a solid green (color code 4) triangle having corners at x=100,y=100; x=200,y=200; and x=200,y=100 we would use the following sequence of calls:

```
CALL GRAFIX
CALL FILINT
CALL CURSOR(100,100)
CALL COLR(4)
CALL FSEG(200,200)
CALL FSEG(200,100)
CALL FSEG(100,100)
```

(Please see diagram on next page.)

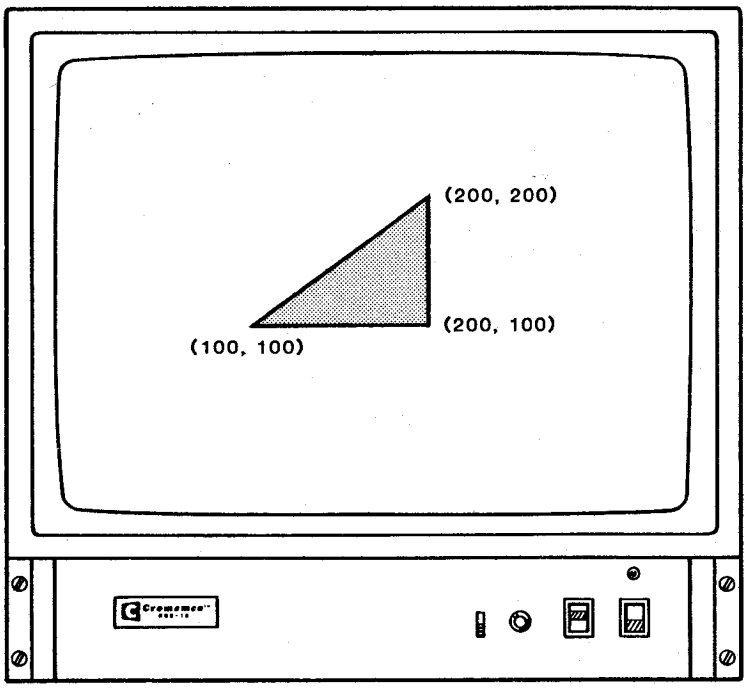


Diagram for **FSEG** and **XFPOLY**

Function: e**X**plicit **F**illed **POLY**gon

format:

FORTRAN:	CALL XFPOLY(c,a)
Assembly:	CALL XFPOLY(c,a)
BASIC:	n/a
SBASIC:	n/a

where:

c = color map code (0-15)

a = an array containing as its first element the number of line segments making up the polygon with subsequent elements of the array containing the x and y components of the polygon's corners.

This call is designed for use in the medium resolution mode. XFPOLY can be used to draw a solid polygon by supplying its vertices in an array 'A' and issuing a single call to XFPOLY. The format of the array 'A' is:

Array	Comment
A(1) = n	number of vertices
A(2) = x(1)} A(3) = y(1)}	vertex #1
A(4) = x(2)} A(5) = y(2)}	vertex #2
.	
.	
.	
A(2n) = x(n)} A(2n+1) = y(n)}	vertex #n

EXAMPLE:

To draw a solid orange triangle having corners at x=100, y=100; x=200, y=200; and x=200, y=100 we would use the following sequence:

```
INTEGER A(7)
DATA A/3,100,100,200,200,200,100/
CALL FILINT
CALL XFPOLY(12,A)
```

(Please see diagram on preceding page.)

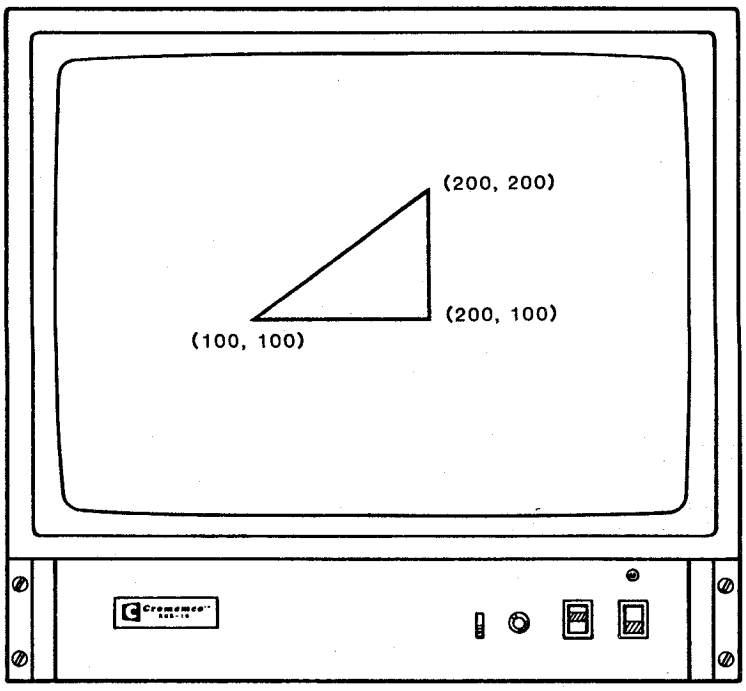
Function: eXit **FILL** mode

format:

	FORTRAN:	CALL XFILL
	Assembly:	CALL XFILL
	BASIC:	USR(316,67)
	SBASIC:	.XFILL

This call will disable the auto-fill display mode. Filled polygons which are drawn using FSEG will appear unfilled (i.e., outlined only) after this call.

CALL XFILL



Interior of auto-filled polygons is erased.

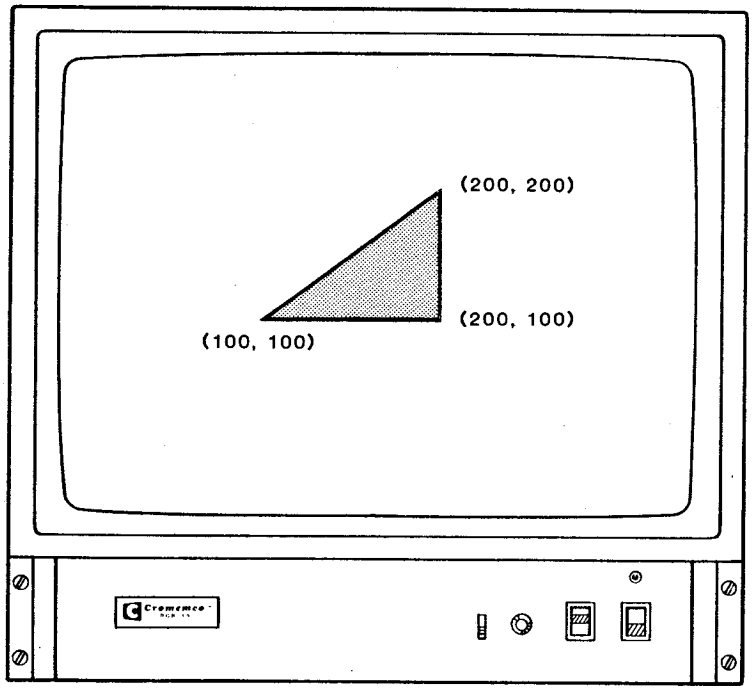
Function: **Auto FILL** mode

format:

FORTRAN:	CALL AFILL
Assembly:	CALL AFILL
BASIC:	USR(316,66)
SBASIC:	.AFILL

This call enables the auto-fill mode. This mode is used for plotting solid figures. It is normally used to reinstitute the auto-fill mode after a previous XFILL call. Note that AFILL need not be called in addition to FILINIT, as one of the functions of FILINT is to enable the auto-fill mode.

CALL AFILL



Interiors of polygons drawn using fseg or or xfpoly are filled.

Programming Example 9.
The following is an example of the use of the auto-fill mode to draw solid shapes. Note that standard graphics calls such as XTEXT can be used in auto-fill if a "15" is deposited in each memory location surrounding the text. (Refer to the section of the program titled Headings.)

```
# Program : POLYGON
# Purpose : to demonstrate the use of the routines:
#                   FSEG
#                   XFPOLY/XPOLY
#                   AFILL
#                   XFILL
#                   FILINT
#             and the autofill-mode.
#
# Remark  : XPOLY is similar to XFPOLY but does not fill in the area.

        integer a(20),b(10)

        data    a       /9,47,180,132,180,132,140,126,140,
                         136,120,136,210,164,210,164,120,132,30/

        data    b       /4,190,180,218,180,164,30,136,30/

        call    grafix
        call    filint                          # enable autofill mode
        call    defclr(8,15,10,0)               # redefine the color map
        call    defclr(7,0,15,4)
        call    defclr(9,0,5,15)
        call    scale(1,378,-20,300)

# Headings
        call    xarea(1,215,378,300,15)         # color code 15 is black
                                                # in auto-fill mode
        call    xtext(130,245,9,'Auto-Fill Mode^')
        call    xtext(131,245,9,'Auto-Fill Mode^')

# Using XFPOLY  draw fill-in polygons
        call    xfpoly(8,a)
        call    xfpoly(8,b)

# Similar to above but using FSEG
        call    colr(7)
        call    cursor(222,180)
        call    fseg(250,180)
        call    fseg(198,30)
        call    fseg(170,30)
        call    fseg(222,180)

        call    cursor(254,180)
        call    fseg(337,30)
        call    fseg(256,30)
        call    fseg(256,70)
        call    fseg(263,70)
        call    fseg(252,90)
        call    fseg(252,1)
        call    fseg(224,1)
        call    fseg(224,91)
        call    fseg(254,180)
```

```
for (loop=1;loop<=5;loop=loop+1)
        {
        for (idelay=1;idelay<=30000;idelay=idelay+1)
            {
            ;
            ;
            }
        call    xfill           #disable fill-in mode

        for (idelay=1;idelay<=30000;idelay=idelay+1)
            {
            ;
            ;
            }
        call    afill           # enable fill-in mode
        }
end
```

Programming Example 10.
The following program illustrates the use of animation mode to simulate the motion of a pendulum.

```
# Program : PENDULUM
# This program uses AINIT and ANIMAT to simulate a pendulum.

        integer xcent(4),ycent(4)
        data    xcent(1),ycent(1)/300,-170/,
                xcent(2),ycent(2)/200,-200/,
                xcent(3),ycent(3)/-200,-200/,
                xcent(4),ycent(4)/-300,-170/

        call    grafix
        call    init

# set up  page0 in animation mode
        call    ainit(0)
        call    defclr(1,0,15,4)
        call    defclr(2,15,15,0)
        call    scale(-400,400,-300,300)

# set up pagel in animation mode
        call    animat
        call    scale(-400,400,-300,300)

# Draw on both pages
        for (loop=1;loop<=10;loop=loop+1)
                {
                for (i=1;i<=4;i=i+1)
                    {
                    call        animat
                    call        xarea(-5,-230,5,300,1)
                    call        xline(0,300,xcent(i),ycent(i),1)
                    call        xfcir(xcent(i),ycent(i),15,2)
                    call        xtext(-100,-280,15,'Animation Mode^')
                    }
                for (i=3;i>=2;i=i-1)
                    {
                    call        animat
                    call        xarea(-5,-230,5,300,1)
                    call        xline(0,300,xcent(i),ycent(i),1)
                    call        xfcir(xcent(i),ycent(i),15,2)
                    call        xtext(-100,-280,15,'Animation Mode^')
                    }
                }
        end
```

Programming Example 11.
Here we have the rotation of a 3-D cube. This is another example of the animation routines:

```
# Program :CUBETURN
# Purpose : demonstrate the use of AINIT,ANIMAT,ANIM for the animation mode.


        integer left,right,top,bottom,limit,cubnum,count
        integer ordx(400),ordz(400)
        real    x(400),y(400),z(400)
        real    zrotx(400),zroty(400),zrotz(400)
        #-------------------------------------------------
        data    x(1),y(1),z(1)/40.,0.,40./,
                x(2),y(2),z(2)/40.,40.,40./,
                x(3),y(3),z(3)/40.,40.,0./,
                x(4),y(4),z(4)/40.,0.,0./,
                x(5),y(5),z(5)/0.,0.,40./,
                x(6),y(6),z(6)/0.,40.,40./,
                x(7),y(7),z(7)/0.,40.,0./,
                x(8),y(8),z(8)/0.,0.,0./
        #-------------------------------------------------

        call    grafix

# Define input parameters

        left=-240
        right=240
        bottom=-180
        top=180

        rotang =8.                              # angle of rotation
        a=(3.1415927/180.)* rotang
        limit =360/ifix(rotang)
        cubnum =8 * limit                       # number of cubes to be drawn
        do      i=1,cubnum
                {
                # DEFINE ROTATION
                zrotx(i)=x(i)*cos(a) + y(i)*sin(a)
                zroty(i)=-x(i)*sin(a)+ y(i)*cos(a)
                zrotz(i)=z(i)

                #DEFINE PLOT POINTS
                ordx(i)=zrotx(i) + zroty(i)*cos(3.1415927/4.0)
                ordz(i)=zrotz(i) + zroty(i)*cos(3.1415927/4.0)

                #REDEFINE X,Y,Z FOR NEXT ITERATION
                x(i+8)= zrotx(i)
                y(i+8)= zroty(i)
                z(i+8)= zrotz(i)
                }
        call    init
        call    defclr(10,10,15,00)
        call    defclr(12,0,15,10)
        call    defclr(13,15,15,15)
        call    defclr(11,15,0,0)
        call    defclr(5,15,10,00)
        call    defclr(6,10,0,15)
        call    defclr(7,13,3,10)
        call    defclr(8,0,10,15)
        call    defclr(1,8,8,8)
        call    defclr(2,10,10,10)
        call    defclr(3,12,12,12)
        call    defclr(4,14,14,14)

        for (i=1;i<=2;i=i+1)
            {
        if (i==1)
           call ainit(0)
```

```
                else
                   call animat

                call        xarea(84,52,94,120,1)
                call        xarea(95,52,189,60,1)
                call        xarea(84,121,94,190,2)
                call        xarea(95,180,189,190,2)
                call        xarea(190,52,283,60,4)
                call        xarea(283,52,293,120,4)
                call        xarea(283,121,293,190,3)
                call        xarea(190,180,283,190,3)

                call        xarea(90,18,290,30,0)
                call        xtext(95,20,15,'Animation by CROMEMCO^')
                call        xtext(96,20,11,'Animation by CROMEMCO^')
                }

        # Rotate the cube
              call    workon(0)
              call    scale(left,right,bottom,top)
              call    workon(1)
              call    scale(left,right,bottom,top)

            # Loop
            for (loop=1;loop<=3;loop=loop+1)
               {
               J = 1                                        # set counter
               count=0
               for (k=1;k<=limit;k=k+1)
                       {
                       count = k * ifix(rotang)
                       call    anim

                       # draw the cube with no hidden lines showing

                       call    xline(ordx(j),ordz(j),ordx(j+1),ordz(j+1),8 )

                       if (^(count >232 & count < 322))
                         call  xline(ordx(j+1),ordz(j+1),ordx(j+2),ordz(j+2),6 )

                       if (^((count >142 & count <=232)|(count>232 & count< 322)))
                         call  xline(ordx(j+2),ordz(j+2),ordx(j+3),ordz(j+3),8 )
                       if (^(count >142 & count <232 ))
                         call  xline(ordx(j+3),ordz(j+3),ordx(j),ordz(j),5)

                       call    xline(ordx(j+4),ordz(j+4),ordx(j+5),ordz(j+5),7 )

                       if (^((count >  7 & count < 52)|(count>322 & count< 367)))
                         call  xline(ordx(j+5),ordz(j+5),ordx(j+6),ordz(j+6),6 )

                       if (^((count >7 & count <52 )|
                             (count >322 & count <=367)|
                             (count >52 & count <142)))
                         call  xline(ordx(j+6),ordz(j+6),ordx(j+7),ordz(j+7),7 )

                       if (^(count > 52 & count < 142))
                         call  xline(ordx(j+7),ordz(j+7),ordx(j+4),ordz(j+4),5 )

                       if (^((count > 52 & count <142)|(count>142 & count< 232)))
                         call  xline(ordx(j+7),ordz(j+7),ordx(j+3),ordz(j+3),5)

                       call    xline(ordx(j+4),ordz(j+4),ordx(j),ordz(j),5)

                       if (^((count >7 & count <52 )|
                             (count >322 & count <=367)|
                             (count >232   & count <322)))
                       call    xline(ordx(j+2),ordz(j+2),ordx(j+6),ordz(j+6),6 )

                       call    xline(ordx(j+1),ordz(j+1),ordx(j+5),ordz(j+5),6 )
                       j =j+8

                       }
               }               ## end of loop
               END
```

Programming Example 12.
The size and location of the 24 segments spanning the screen is demonstrated in this program.

```
# Program : SEGMENT
# This program illustrates the size of the segments that are controlled
# by the window mode and the use of routine TURN to rotate
# a string.


        call    grafix
        call    init

# Draw on page0
        call    xtext(150,225,15,'24 Segments^')
        call    xtext(125,210,15,'for Page Windowing^')
        call    xtext(120,195,15,'or Resolution Boxes^')
        call    xarea(1,30 ,378,185,6)

# Draw on pagel
        call    workon(1)
        call    page
        call    xarea(1,30,378,185,8)
        call    xline(1,1,378,241,15)
        call    xline(1,241,378,1,15)

#  Open window to pagel
        call    winit(0)
        for (iwopen=2;iwopen<=24;iwopen=iwopen+2)
                call    wopen(iwopen,30 ,iwopen,185)

# Draw 24 segments on page0 and compare them with
# the segments controlled by "WOPEN"

        call    xarea(1,1,10,29 ,2)
        ix=11
        iwidth=26
        for (iseg=2;iseg<=24;iseg=iseg+1)
                {

                iclr=mod(iseg,14)+1
                call    xarea(ix,1,iwidth,29 ,iclr)
                ix =ix+16
                iwidth =iwidth+16
                }

        # Numbering the segments
        call    xtext(2,8,0,'1^')
        call    xtext(15,8,0,'2^')
        call    xtext(31,8,0,'3^')
        call    xtext(47,8,0,'4^')
        call    xtext(63,8,0,'5^')
        call    xtext(79,8,0,'6^')
        call    xtext(95,8,0,'7^')
        call    xtext(111,8,0,'8^')
        call    xtext(127,8,15,'9^')

        call    workon(1)
        call    xarea(1,1,378,20,0)
        call    xtext(1,4,15,'10^')
        call    turn(154,4,90)

        call    workon(1)
        call    xarea(1,1,378,20,0)
        call    xtext(1,4,15,'11^')
        call    turn(170,4,90)
```

```
call    workon(1)
call    xarea(1,1,378,20,0)
call    xtext(1,4,15,'12^')
call    turn(186,4,90)

call    workon(1)
call    xarea(1,1,378,20,0)
call    xtext(1,4,15,'13^')
call    turn(202,4,90)

call    workon(1)
call    xarea(1,1,378,20,0)
call    xtext(1,4,15,'14^')
call    turn(218,4,90)

call    workon(1)
call    xarea(1,1,378,20,0)
call    xtext(1,4,15,'15^')
call    turn(234,4,90)

call    workon(1)
call    xarea(1,1,378,20,0)
call    xtext(1,4,15,'16^')
call    turn(250,4,90)

call    workon(1)
call    xarea(1,1,378,20,0)
call    xtext(1,4,15,'17^')
call    turn(266,4,90)

call    workon(1)
call    xarea(1,1,378,20,0)
call    xtext(1,4,15,'18^')
call    turn(282,4,90)

call    workon(1)
call    xarea(1,1,378,20,0)
call    xtext(1,4,15,'19^')
call    turn(298,4,90)

call    workon(1)
call    xarea(1,1,378,20,0)
call    xtext(1,4,15,'20^')
call    turn(314,4,90)

call    workon(1)
call    xarea(1,1,378,20,0)
call    xtext(1,4,15,'21^')
call    turn(330,4,90)

call    workon(1)
call    xarea(1,1,378,20,0)
call    xtext(1,4,15,'22^')
call    turn(346,4,90)

call    workon(1)
call    xarea(1,1,378,20,0)
call    xtext(1,4,15,'23^')
call    turn(362,4,90)

call    workon(1)
call    xarea(1,1,378,20,0)
call    xtext(1,4,15,'24^')
call    turn(378,4,90)
end
```

Programming Example 13.
The following is a program which runs through many of the options in window mode.

```
# Program : WINDOW
# Purpose : This program illustrates the use of the subroutines:
#                          WINIT(0-1)
#                          WOPEN(x1,y1,x2,y2)
#                          WCLOSE(x1,y1,x2,y2)
#                          WENAB
#                          WDISAB(0-1)
#                          WEXIT(0-1)
#         & redefines and changes the color map.


        call    grafix
        call    init
        call    scale(1,400,1,300)
# Redefine the color map
        call    defclr(1,15,0,0)
        call    defclr(2,15,10,0)
        call    defclr(3,15,15,0)
        call    defclr(4,0,0,15)
        call    defclr(5,0,10,15)
        call    defclr(6,0,13,0)
        call    defclr(7,0,13,6)
        call    defclr(8,15,0,10)
        call    defclr(9,12,0,6)
        call    defclr(10,12,0,15)

# Write on page 0
        ix1=101
        ix2=120
        for (icolr=1;icolr<=10;icolr=icolr+1)
                {
                call    xarea(ix1,1,ix2,300,icolr)
                ix1=ix1+20
                ix2=ix1+19
                }

# Write on page1
        call    workon(1)
        call    page
        call    res(1)
        call    hscale(1,400,1,300)
        call    hxtext(170,120,'WINDOW MODE^')
        call    hxtext(170,105,'  enabled^')

# Look from page0 to page1
        call    winit(0)
        call    wopen(10,100,14,135)
        for (idelay=1;idelay<=500;idelay=idelay+1)
                {
                for (i=1;i<=150;i=i+1)
                        ;
                }
        call    wclose(10,90,14,115)
        for (idelay=1;idelay<=500;idelay=idelay+1)
                {
                for (i=1;i<=150;i=i+1)
                        ;
                }

        # Change color map
        for (iclr=10; iclr<=100 ;iclr=iclr+1)
                {
                i1=mod(iclr,10)+1
                i2=mod(iclr+1,10)+1
```

```
                i3=mod(iclr+2,10)+1
                i4=mod(iclr+3,10)+1
                i5=mod(iclr+4,10)+1
                i6=mod(iclr+5,10)+1
                i7=mod(iclr+6,10)+1
                i8=mod(iclr+7,10)+1
                i9=mod(iclr+8,10)+1
                i10=mod(iclr+9,10)+1
                call    defclr(i1,15,0,0)
                call    defclr(i2,15,10,0)
                call    defclr(i3,15,15,0)
                call    defclr(i4,0,0,15)
                call    defclr(i5,0,10,15)
                call    defclr(i6,0,13,0)
                call    defclr(i7,0,13,6)
                call    defclr(i8,15,0,10)
                call    defclr(i9,12,0,6)
                call    defclr(i10,12,0,15)

                for (idelay=1;idelay<=800 ;idelay=idelay+1)
                        ;
                }

        call    wdisab(1)               # look at page1
        for (idelay=1;idelay<=500;idelay=idelay+1)
                {
                for (i=1;i<=150;i=i+1)
                        ;
                }

        call    wenab
        for (idelay=1;idelay<=500;idelay=idelay+1)
                {
                for (i=1;i<=150;i=i+1)
                        ;
                }

# Look from page1 to page0
        call    winit(1)
        for (idelay=1;idelay<=500;idelay=idelay+1)
                {
                for (i=1;i<=150;i=i+1)
                        ;
                }
        call    wopen(8,150,16,250)
        for (idelay=1;idelay<=500;idelay=idelay+1)
                {
                for (i=1;i<=150;i=i+1)
                        ;
                }
# Back to page 0
        call    wexit(0)
        end
```

Chapter 4

Color Map Generation

The CMAPGEN file included in the SDI Graphics software package makes it convenient for the programmer to create his own color maps. The color maps created using this program are stored as relocatable machine language files which can be linked to FORTRAN or Assembly language programs. The program can be thought of as an editor that takes an existing relocatable machine language file which represents a color map, copies it, allows the user to modify the color map, and then writes this new map on to the disk as a relocatable file with a user designated name. The designated name of the relocatable file must be exactly four (4) characters long. The output obtained from this program can not be incorporated into a BASIC program. The only option available from BASIC for changing colors is the DEFCLR call.

Using CMAPGEN is fairly easy since it is menu driven and the options available are clearly specified. Execution of the program results in the main menu

MAIN MENU

p -- pick a new color code to work on
z -- display current values of color map
w -- write new file and exit from program

being displayed on the terminal screen. From the main menu we have the choice of either modifying the color map (p), displaying the relative intensities associated with each color in the color map (z), or writing the color map generated as a relocatable machine language file (w). If the programmer elects the "p" option then a color map will be displayed on the video screen with the hexadecimal designation for each of the color codes. This display is shown in Figure 4.1.

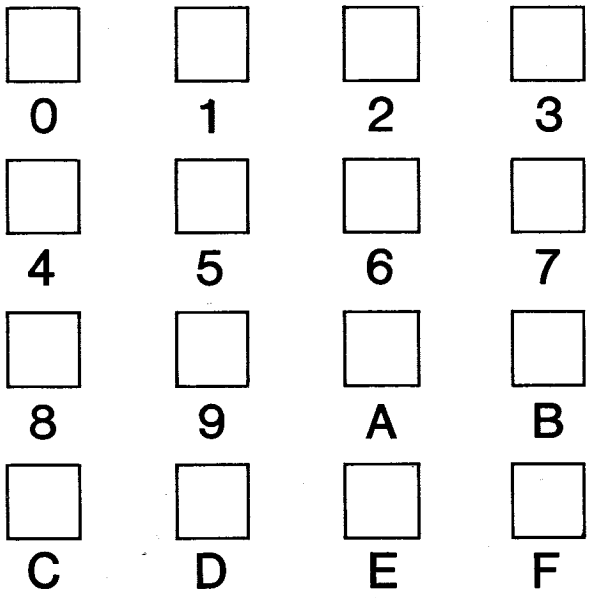


Figure 4.1

The programmer is then instructed on the CRT terminal:

Enter hexadecimal character below the box to be modified

Typing the hexadecimal code of the color to be modified will result in the following color code modification sub-menu being displayed on the CRT terminal:

g -- to select green
r -- to select red
b -- to select blue
p -- pick a new color code to work on
< -- decrease intensity
> -- increase intensity
t -- exit to main menu

and the appearance of double hash marks on the video display to denote the color code being modified.

The programmer now selects the color component to be adjusted by typing r, g, or b for red, green, or blue respectively. Then either increases or decreases the intensity of that component by typing

a > or < respectively. These last two characters can be used interchangeably with a "." for the ">" and a "," for the "<". For all of the above menus, it is not necessary to type a "return" following a response.

As the intensity is adjusted the video display is constantly updated and shows the resulting color for the selected color code. At the bottom of the display the intensity of the color component being adjusted is shown along with its current hexadecimal intensity value. After the selected color code has been adjusted to the desired value the programmer can pick another color code for adjustment by typing a "p" and continue the previous procedure until the desired color map has been created. He then returns to the main menu and writes the color map as a relocatable file to disk. The file thus created can be linked to the programmer's program. Using this approach the programmer can have any number of color maps incorporated into his program and changed by the program at the appropriate time.

EXAMPLES:
(programmer input is shown in **bold type**)

To invoke CMAPGEN

A.**CMAPGEN XXXX.REL YYYY.REL**

XXXX.REL is the color map file to be used to create a new map. (MAP1.REL is the name of the relocatable color map file supplied with the graphics software package)

YYYY.REL is the name to be assigned to the color map generated.

Note: Remember the name of the REL files must be four characters long.

To create a color map called MAP1.REL using the MAP0.REL file as a starting point the programmer would type:

A.**CMAPGEN MAP1.REL MAP2.REL**

then to create a third color map named FERN.REL using the MAP2.REL file as the starting point the programmer types:

```
A.CMAPGEN MAP2.REL FERN.REL
```

To use these new color maps the programmer would link MAP2.REL and FERN.REL with the application program written in either FORTRAN, RATFOR, or Assembler. The source code of the application program might look like this:

```
PROGRAM BARCHART
      .
      .
      .
100 CALL MAP1
      .
      .
      .
400 CALL FERN
410 GOTO 10
420 END
```

This program will change the color codes in the color map back and forth between MAP2 values and FERN values. After compiling this program the linking command would be:

```
LINK MAP2, FERN, SDIFOR, BARCHART
```

MAP2 followed by FERN must be the first item in the list of modules to be linked. This is the only instance when SDIFOR is not the first item in the link list.

Chapter 5

SAVING AND LOADING IMAGES FROM DISK

The command files PIXSAVE and PIXLOAD are used primarily for saving to disk and loading from disk pictures which have been created using the SDI graphics package. They are both command files and are called from CDOS (the Cromemco Disk Operating System). Both the PIXSAVE call and the PIXLOAD call include several options which increase the convenience and usefulness of the graphics package.

5.1 PIXSAVE

The PIXSAVE call is used to compress an image and to save it on a disk or to determine the amount of disk space required to save it. The call has the form:

PIXSAVE [/<options>] [<picture filename>]

where the options available are **0, 1, H, S,** and **U.** An explanation of these options is contained in the following paragraphs. Note that any number of options (in any order) can be selected after the / symbol.

/0: This option saves the image stored in page 0 of the Two Port memory.

/1: This option saves the image stored in page 1 of the Two Port memory.

/H: This Help option lists the available options. No image is stored when this is selected.

/S: This option results in statistics (see subsequent example) for the designated image (default value is page 0) being displayed on the CRT terminal. No image is stored when this option is selected.

/U: This option results in a U attribute being attached to the saved image. This attribute is displayed any time the disk directory is printed and as a result makes it easy to pick out an image file.

The PIXSAVE call will normally verify that the saved image agrees with the image stored in memory, and if verification is not possible, the saved file will be given the file name extension $$$. This can occur in either of two ways. First, if the verify routine is not on the current disk drive, it will not be able to be used. (Note that the verify routine is contained in the PIXLOAD program only and therefore PIXLOAD must be on the current disk in order for the verify routine to work.) Second, hardware or software problems may result in a failure to verify. Consider the case where we have an image stored under the name MIKE.PXS. After loading the image and modifying it, we resave it with the same name. If PIXSAVE verifies that the saved image is correct, then the new file MIKE.PXS replaces the old file having the same name and the disk directory remains unchanged. However, if PIXSAVE is not able to verify that the saved image is correct, then this unverified image will be saved with the file name MIKE.$$$ and the original file MIKE.PXS will remain unchanged.

Examples of the PIXSAVE format are:

PIXSAVE PICTURE.PXS
PIXSAVE PICTURE
PIXSAVE/0 PICTURE.PXS

These calls will save the image stored in page 0 with the file name PICTURE.PXS. If the page to be stored is not designated, the program will default to page 0. If the file extension .PXS is omitted, PIXSAVE will automatically default to this extension. Any other extension must be explicitly written, such as:

PIXSAVE PICTURE.SAM

The call:

PIXSAVE/1 PICTURE.PXS

will save the image stored in page 1 (Bank 6) of RAM under the file name PICTURE.PXS.

In response to the PIXSAVE/S call:

PIXSAVE/S PICTURE.PXS

the program will display the following statistics on the image contained in page 0 of the Two Port

memory, but will not store the image.

```
Original file in bytes    = 49152
Compressed file in bytes =  3887
Total compression factor = 12.64
Number of bytes stored    =     0
```

Note that in a non-S call to PIXSAVE, the "number of bytes stored" value is continually updated while the image is being stored. After the image has been stored the statement:

Picture Compressed and Stored

will be printed on the terminal screen.

Either of the following two instructions:

```
PIXSAVE/1S PICTURE.PXS
PIXSAVE/S1 PICTURE
```

could have been used to obtain the statistics on the image contained in page 1 of TP memory.

5.2 PIXLOAD

The PIXLOAD call can be used to decompress, load and/or display an image which has previously been compressed and stored using the PIXSAVE command. The form of the command is:

PIXLOAD[/<options>] [<picture filename>] [<link filename>]

where the options available are 0, 1, C, D, G, H, L, M, N, O, P, S, V, and W. An explanation of these options is contained in the following paragraphs. Note that any number of options can be selected after the / symbol (and in any order). If no option is designated, then the specified file is loaded into page 0 of RAM along with control information (i.e., color map and control port information) and displayed. If the extension is omitted from the first filename, then .PXS will be the default choice of PIXLOAD. Other extensions are possible but must be included explicitly in the file name.

/0: Load the specified file into page 0 of RAM.

/1: Load the specified file into page 1 of RAM.

/C: Suppress the incoming color map. The incoming map is loaded into the Two Port memory but is not sent to the SDI. Therefore the image will be displayed using the color map present in the SDI.

/D: Disable the monitor screen during picture loading. If the /P or /N options have not been specified, the screen will be turned on after loading.

/G: Output the values for the default gray colormap and medium resolution control mode.

/H: This Help option will cause a complete list of the available options to be printed on the console screen for reference.

/L: This option will cause control to be transferred to the last file name after the completion of the picture-loading process.

/M: Console messages will be suppressed when this option is specified.

/N: Load the image without updating the color or control information. This call is equivalent to a PIXLOAD/CP call.

/O: Send the control and colormap information presently in the Two Port Memory to the SDI only. This can be used to change only the display page without loading an image.

/P: Suppress the control port information. That is, if we have been displaying a high resolution picture and we load a medium resolution picture with the PIXLOAD/P option selected, then the high resolution mode remains in force. For example, we may be loading an image into the non-viewed page of memory and do not want to disturb the image being displayed.

/S: Output the values for the default colormap and medium resolution control mode to the SDI without loading an image. That is, turn on the screen and display the image with the standard colormap and medium resolution.

/V: Compare the file specified with the image stored in memory without loading an image.

/W: Load the image but not the control area or control data (i.e., load the decompressed picture file into 4300H through FFFFH, skipping 4000H through 42FFH). **Caution:** Unless the control and color information is suppressed with /N, /C, /P, or /CP the new (incoming) control data will be sent to the SDI even though it will not be loaded into the control area of the Two Port RAM.

As an example, the PIXLOAD instruction:

PIXLOAD/L MIKE.PXS DISPLAY.COM

will result in the image stored under the file name MIKE.PXS being loaded from disk, displayed, and control transferred to the command file DISPLAY. Either of the following instructions will change the displayed image from page 1 to page 0:

PIXLOAD/O0
PIXLOAD/0O

Either of the following instructions can be used to change the display back to page 1:

PIXLOAD/O1
PIXLOAD/1O

Appendix A.
List of Housekeeping and Graphics Calls

The following is an alphabetical listing of all calls available using the SDI graphics package. The graphics calls are indicated by an 'i' for the implicit calls and an 'e' for the explicit calls. Since the BASIC form of the text calls require additional programming steps, they are denoted by an '*' in front of the BASIC format in an attempt to call your attention to this fact.

	FORTRAN Assembly SBASIC+SDILIB	BASIC SBASIC w/o SDILIB
	AFILL	X=USR (316,66)
	AINIT(p)	X=USR (316,72)
	ANIM	none
	ANIMAT	X=USR (316,73)
i	AREA(x,y)	X=USR (316,45)
	CLIP	X=USR (316,51)
	COLR(c)	X=USR (316,32,c)
	CURSOR(x,y)	X=USR (316,33,x,y)
	DEFCLR(c,R,G,B)	X=USR (316,84,c,R,G,B)
	DISP(p)	X=USR (316,70,p)
i	DOT	X=USR (316,37)
	ERABOX(x1,y1,x2,y2)	X=USR (316,82,x1,y1,x2,y2)
	ESTRNG("p")	none
	FILINT	X=USR (316,65)
i	FSEG(x,y)	X=USR (316,31,x,y)
	GRAFIX	none
i	HAREA(x,y)	X=USR (316,46,x,y)
	HCRSOR(x,y)	X=USR (316,34,x,y)
i	HDOT	X=USR (316,38)
i	HLINE(x,y)	X=USR (316,36,x,y)
i	HRLINE(x,y)	X=USR (316,20,x,y)
	HSCALE(x1,x2,y1,y2)	X=USR (316,54,x1,x2,y1,y2)
i	HTEXT("text ^")	* X=USR (316,48,T1)
e	HXAREA(x1,y1,x2,y2)	X=USR (316,5,x1,y1,x2,y2)
e	HXCIRC(x,y,r)	X=USR (316,7,x,y,r)
e	HXDOT(x,y)	X=USR (316,3,x,y)
e	HXFCIR(x,y,r)	X=USR (316,9,x,y,r)
e	HXLINE(x1,y1,x2,y2)	X=USR (316,1,x1,y1,x2,y2)
e	HXPOLY(a)	X=USR (316,17,a)
e	HXREAD(x,y,V)	none

```
e HXTEXT(x,y,"text ^")   * X=USR (316,13,T1)
  INIT                     X=USR (316,68)
  INIT1                    none
i LINE(x,y)                X=USR (316,35,x,y)
  PAGE                     X=USR (316,71)
i READ(x,y,V)              none
  RES(r)                   X=USR (316,80,r)
  RESBOX(x1,y1,x2,y2)      X=USR (316,81,x1,y1,x2,y2)
i RLINE(x,y)               X=USR (316,19,x,y)
  SCALE(x1,x2,y1,y2)       X=USR (316,49,x1,x2,y1,y2)
  SCROFF                   X=USR (316,64)
  SCRON                    X=USR (316,63)
i TEXT("text ^")         * X=USR (316,47,T1)
  UNCLIP                   X=USR (316,52)
  UNSCAL                   X=USR (316,50)
  WAITHG                   X=USR (316,86)
  WAITOD                   X=USR (316,88)
  WAITVG                   X=USR (316,85)
  WAITVS                   X=USR (316,87)
  WCLOSE(x1,y1,x2,y2)      X=USR (316,76,x1,y1,x2,y2)
  WDISAB(p)                X=USR (316,77,p)
  WENAB                    X=USR (316,78)
  WEXIT(p)                 X=USR (316,79,p)
  WINIT(p)                 X=USR (316,74,p)
  WOPEN(x1,y1,x2,y2)       X=USR (316,75,x1,y1,x2,y2)
  WORKON(p)                X=USR (316,75,p)
e XAREA(x1,y1,x2,y2,c)     X=USR (316,4,x1,y1,x2,y2,c)
e XCIRC(x,y,r,c)           X=USR (316,6,x,y,r,c)
e XDOT(x,y,c)              X=USR (316,3,x,y,c)
e XFCIR(x,y,r,c)           X=USR (316,8,x,y,r,c)
  XFILL                    X=USR (316,67)
e XFPOLY(c,a)              none
e XLINE(x1,y1,x2,y2,c)     X=USR (316,0,x1,y1,x2,y2,c)
e XPOLY(c,a)               X=USR (316,16,c,a)
e XREAD(x,y,V)             none
e XTEXT(x,y,c,"text ^") * X=USR (316,12,x,y,c,t1)
```

Cromemco™
incorporated
Tomorrow's Computers Today
280 BERNARDO AVE., MOUNTAIN VIEW, CA 94043

023-4015