A GENERAL PURPOSE GRAPHICS SUPPORT LIBRARY FOR THE ADA PROGRAMMING LANGUAGE HOSTED ON THE ZENITH H/Z-100 COMPUTER(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA

UNCLASSIFIED T R BROWN DEC 86
THESIS

A GENERAL PURPOSE GRAPHICS SUPPORT LIBRARY FOR THE ADA PROGRAMMING LANGUAGE HOSTED ON THE ZENITH H/2-100 COMPUTER

by

Thomas R. Brown, Jr.

December 1986

Thesis Advisor U.R. Kodres

Approved for public release; distribution is unlimited
This thesis explores the requirements necessary to develop a graphics support library for the Ada programming language hosted on the Zenith H/2-100 microcomputer. A prototype graphics library is implemented in 8086 assembly language embedded in an Ada package. The library operates with JANUS/Ada under the CP/M-86 operating system. Listings of library routines developed are provided as well as a user's guide and demonstration programs. Potential areas for further investigation and development are suggested. It is concluded that an Ada graphics library for microcomputers is feasible and practical.
A General Purpose Graphics Support
Library for the Ada Programming Language
Hosted on the Zenith H Z-100 Computer

by

Thomas R. Brown, Jr.
B.S., Eastern New Mexico University, 1981

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1986

Author: Thomas R. Brown, Jr.

Approved by: Uno R. Kodres, Thesis Advisor

Michael J. Lydi, Second Reader

Vincent Y. Lum, Chairman,
Department of Computer Science

Kneale T. Marshman,
Dean of Information and Policy Sciences
ABSTRACT

This thesis explores the requirements necessary to develop a graphics support library for the Ada programming language hosted on the Zenith HZ-100 microcomputer. A prototype graphics library is implemented in 8086 assembly language embedded in an Ada package. The library operates with JANUS Ada under the CP M-86 operating system.

Listings of library routines developed are provided as well as a user's guide and demonstration programs. Potential areas for further investigation and development are suggested.

It is concluded that an Ada graphics library for microcomputers is feasible and practical.
THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

Some terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each occurrence of a trademark, all trademarks appearing in this thesis will be listed below, following the organization holding the trademark:

1. United States Government (Ada Joint Program Office)
   Ada
2. Intel Corporation
   Intel
3. Digital Research Corporation
   CP/M
   CP/M-86
4. Microsoft Corporation
   MS-DOS
5. Zenith Data Systems Corporation
   Zenith
   H/Z-100
# TABLE OF CONTENTS

## I. INTRODUCTION
- A. BACKGROUND ........................................... 9
- B. PURPOSE .................................................. 10
- C. THESIS ORGANIZATION ................................... 11

## II. ALGORITHMS
- A. II Z-100 ARCHITECTURE .................................. 12
- B. JANUS ADA ............................................... 15
- C. GRAPHICS ALGORITHMS ................................... 17
  1. Clear Screen (CLS) ....................................... 17
  2. Color Selection .......................................... 18
  3. Video Ram Addressing .................................... 18
  4. Pixel Display ........................................... 19
  5. Line Drawing ............................................ 19
  6. Circle Drawing .......................................... 23
  7. Color Testing ........................................... 23
  8. Area Polygon Filling .................................... 25
  9. Cursor Control .......................................... 26

## III. PERFORMANCE AND EVALUATION
- A. DEMONSTRATION PROGRAMS ............................... 28
  1. Program AGTEST1 ........................................ 28
  2. Program AGTEST2 ........................................ 28
  3. Program AGTEST3 ........................................ 28
  4. Program AGTEST4 ........................................ 28
- B. ADA GRAPHICS LIBRARY LIMITATIONS .................... 28
- C. II Z-100 HARDWARE LIMITATIONS ......................... 29
- D. TIMING MEASUREMENTS .................................... 29
LIST OF TABLES

1. COLOR PLANE ADDRESSES ........................................... 15
2. COLOR TABLE .......................................................... 16
3. COLOR CONTROL ....................................................... 18
4. PERFORMANCE RESULTS ............................................. 39
5. COLOR CODES .......................................................... 34
LIST OF FIGURES

2.1    H Z-100 Block Diagram ......................................................... 13
2.2    Display Matrix ................................................................. 14
2.3    Parameter Passing ............................................................. 17
2.4    Bresenham’s Circle Algorithm ............................................... 24
2.5    Pascal Boundary-Fill Procedure ............................................. 25
I. INTRODUCTION

A. BACKGROUND

The Naval Postgraduate School (NPS) utilizes approximately fifty Zenith H Z-100 microcomputers in the microcomputer laboratories of the Computer Science Department. There were many reasons for choosing this particular computer, foremost of which is the hardware architecture [Ref. 1] and the availability of support software. The H Z-100’s central processing unit (CPL) includes an INTEL 8085 8-bit processor and an INTEL 8088 16-bit processor. The simple architecture and instruction set of the 8085 processor supports the popular CP M operating system and is compatible with 8080 code. The more complex architecture of the 8088 processor is compatible with 8086 code and includes more internal registers (some usable as either 8 or 16-bit), extended addressing modes, and a more complex memory management scheme using segment registers. The 8088 processor supports the more advanced CP M-86 and MS-DOS operating systems.

An important feature of the H Z-100 is its display versatility. The basic H Z-100 includes an internal monochrome display and allows the addition of an additional external color monitor. The color monitor is of medium resolution with 640 horizontal and 225 vertical pixels (640 X 512 in the interface mode). Three 64k pages of video RAM memory provide eight colors (or eight intensity levels with a monochrome monitor).

A primary function of the microcomputer laboratory is to support computer science courses providing special emphasis on tactical computer applications. According to Department of Defense policy [Ref. 2], “The Ada programming language shall become the single, common computer programming language for defense mission-critical applications. Effective 1 January 1984 for programs entering full-scale engineering development. Ada shall be the programming language.” To assist in meeting this requirement, NPS provides courses which include the use of Ada in the development of tactical program applications programs. Most of these programs are run on the H Z-100 computers using the JANUS Ada compiler. To date, several validated Ada compilers have been approved for specific computer systems, with many more expected to be approved in the near future. However, as Patrice Wagner points...
out [Ref. 3] there has been little measurable response within the computer graphics industry. An improvement in this area can be expected with the acceptance of the Ada Binding to the Graphical Kernel System (GKS) [Ref. 4] by the American National Standards Institute (ANSI) X313 committee on graphics standards. However, we cannot sit back and wait for industry to provide all the tools necessary to conduct Ada graphics education and research. Widespread use of Ada can be expected in the near future and with that use there will be increasing requirements for individuals with Ada graphics knowledge. The NPS has an obligation to assist the Navy in meeting those educational requirements.

B. PURPOSE

It is the intent of this thesis to develop an Ada language graphics programming capability by developing a low level design and partial implementation of an Ada graphics library which can be expanded to include a subset of the Ada language binding to the GKS. Functions to be included will be those primitives necessary for:

- clearing the screen
- setting a pixel
- drawing a line
- selecting a color
- filling polygons
- cursor control

These basic primitives will be implemented using 8086 assembly language embedded in an Ada package. The CP M-86 operating system and the JANUS Ada compiler will be used for this implementation.

It is expected that the primary use for this software will be in courses which emphasize tactical applications of computers. It is becoming more and more common to find graphical displays in tactical systems. The ability of computers to provide graphical displays which aid in tactical decision making is widely recognized and the list of tactical applications utilizing graphical displays can be expected to grow as faster and better graphics displays are developed.

It is anticipated that this software will assist in gaining an insight into the feasibility of developing a GKS implementation for Ada on a microcomputer. As an immediate benefit, it will aid in the development of Ada graphics programs on the IBM-PC computer.
C. THESIS ORGANIZATION

Chapter II begins with a brief overview of the architecture of the H Z-100 computer. The overview is presented to assist the reader in understanding the selection and implementation of the graphics algorithms. Next there is a brief discussion explaining the use of the JANUS Ada compiler and CP M-86 operating system. Included in this discussion is a description of parameter passing procedures in JANUS Ada. The remainder of Chapter II is devoted to providing a detailed description of the implemented graphics routines.

Chapter III provides a description of the test demonstration programs listed in Appendices B-E. Also included in this chapter is an evaluation of the H Z-100 color graphics capability. Shortcomings and system hardware performance are discussed.

Chapter IV includes recommendations for further research and program development using the H Z-100 computer.

Chapter V is the final chapter and includes the conclusion and general comments pertaining to the use of microcomputers and Ada in graphics programming.

Appendix A is a user’s guide which describes the procedures available in the Ada graphics library. Also included are some programming tips on usage of the library procedures and sample procedure calls are provided as examples.

Appendices B-E are program listings of demonstration programs written in Ada. They are provided to demonstrate the use of the Ada graphics library as well as serving as a supplement to the user’s guide.

Appendix F is a listing of the specification package for the Ada graphics library. The specification package defines the formats of all externally callable library procedures.

Appendix G is the assembly code listing of all procedures contained in the Ada graphics library.
II. ALGORITHMS

A. H/Z-100 ARCHITECTURE

The following overview of the H/Z-100 architecture is presented in order to clarify the method of implementation of specific algorithms. Figure 2.1 is a block diagram of the basic architecture of the H/Z-100 computer.

NPS's H/Z-100 computers are in the process of being upgraded to include the Model ZVM-1330 color monitor, 192k of on-board cpu memory and three 64k pages of video RAM.

The display screen is formed by a matrix of 640 horizontal and 225 vertical pixels. Each pixel is mapped to a bit in each of the three color planes (red, green, and blue) in video RAM. Display management is provided by the video processor's CRT controller (CRT-C).

The CRT-C has two modes of operation: the character based mode and the pixel based mode. In the character based mode, the CRT-C is programmed for nine scan lines per character, 80 characters per line and 25 lines per screen. In the pixel based mode, the CRT-C is normally programmed to control a matrix of 640 X 225 pixels.

Mapping of video RAM to the screen is performed by the CRT-C to allow scrolling of the screen and management of displayable non-displayable data. Mapping of the display to physical addresses in video RAM is organized such that 128 (numbered 0-127) consecutive bytes are allowed for each scan line. However, only 80 of the 128 bytes are used to control display of the 640 pixels per scan line. Bytes 80-127 of each line must not be used or erroneous data may be displayed. Additionally, the 225 displayable scan lines may be considered to be in sets of 16 lines of which only the first nine lines of each set are displayable.

When operating in the character based mode, the CRT-C is programmed to map around the non-displayable video RAM. However, in the pixel based mode, it is necessary to incorporate a mapping algorithm into graphics routines. This means that of the 400 scan lines, only 225 are displayable. In addition, the CRT-C is normally programmed so that the bottom character row (9 scan lines) is zeroed during vertical retrace time. This is done to keep uninitialized data from being displayed during scrolling. This means that only 24 of the 25 character rows or 216 of the 225
Figure 2.1  II Z-100 Block Diagram
displayable scan lines actually display data. The Ada graphics library has been designed to work within these restrictions in order to maintain compatibility with existing II Z-100 software. Figure 2.2 illustrates the display matrix including the scan line and pixel numbering system.

<table>
<thead>
<tr>
<th>TOTAL</th>
<th>DISPLAYED</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCAN</td>
<td>SCAN</td>
</tr>
<tr>
<td>LINES</td>
<td>LINES</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>NOT</td>
</tr>
<tr>
<td>12</td>
<td>NOT</td>
</tr>
<tr>
<td>13</td>
<td>NOT</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>892</td>
<td>228</td>
</tr>
<tr>
<td>893</td>
<td>228</td>
</tr>
<tr>
<td>894</td>
<td>224</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>899</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.2 Display Matrix.

There are two ways in which to control writing into the individual color planes. The first method is by programming the video control register of the video logic board. With this method, when a write enabled color plane is accessed, all write enabled color planes are simultaneously written to. For example, if all three color planes are write enabled and the green color plane is written to, then the corresponding addresses in the
blue and red color planes will also be written to. However, this feature was found to be unsatisfactory when constructing a graphics picture consisting of multiple colors and adjacent or overlapping objects. Under these conditions, it was discovered that the only color plane which could be guaranteed to contain correct data was the one which was directly written to. Therefore, the algorithms implemented in the Ada graphics library utilize the second method which involves direct control over each of the three color planes. This means that the implemented algorithms must access all three color planes for each pixel that is set. [Ref. 1: pp. 4.30-4.38]

<table>
<thead>
<tr>
<th>COLOR PLANE</th>
<th>RAM ADDRESSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>blue</td>
<td>000000 - 00FFFF</td>
</tr>
<tr>
<td>red</td>
<td>000000 - 00FFFF</td>
</tr>
<tr>
<td>green</td>
<td>000000 - 00FFFF</td>
</tr>
</tbody>
</table>

To produce color, the video RAM is divided into three main colors: red, green, and blue. Video RAM memory mapping of the three color planes is provided in Table 1 with all addresses in hexadecimal. The pixel seen on the color monitor is actually composed of three superimposed pixels, one in each color plane. By selecting combinations of the three basic colors, eight different colors may be displayed as indicated by Table 2.

B. JANUS/ADA

An overview of the JANUS Ada compiler [Ref. 5] and the CP M-86 operating system [Ref. 6] relating to the implementation of the Ada graphics library is presented to assist the reader in understanding the methods used to implement the graphics algorithms. The graphics routines are available to an Ada language program by linking to library routine ADAGRAPH.LIB (Appendix F).

ADAGRAPH.LIB is implemented as an Ada specification package. That package contains the specifications of the callable library procedures plus a list of the variables used by the Ada graphics library assembly code. The assembly code is
The following steps are used to compile, assemble, and link the Ada graphics library:

1. JANUS ADAGRAPH.LIB
2. JASMIS6 ADAGRAPH
3. JLINK ADAGRAPH

<table>
<thead>
<tr>
<th>TABLE 2</th>
<th>COLOR TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GREEN</td>
<td>RED</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Parameters are passed in Ada by pushing them onto the system stack. Discrete and access data type parameters of mode IN are passed by value. All other data type parameters and modes are passed by reference. Upon entry to a procedure, the top of the stack contains the return address. Parameters or parameter addresses appear on the stack with the last parameter nearest the top of the stack. Figure 2.3 is an example illustrating the format of a procedure specification and the corresponding location of parameters on the system stack when the procedure is executed.

If the parameters passed are only of mode IN, then the assembly procedure must remove those parameters from the stack and leave only the return address on top of the stack before executing a RET instruction to return from the procedure to the calling program. If the parameter list includes any mode OUT or IN OUT parameters, then the stack must be in the same configuration upon exit from the procedure as it was upon entry.

When it is desired to access parameters on the stack without altering the stack contents, then the BP register should be used. The following steps illustrate how this can be performed.

1. MOV BP, SP ;Copy SP register to BP register
procedure INQUIRE_COLOR(x_pos, y_pos: in INTEGER; color: out INTEGER);

--- >--------------c
Top
--- -- -- --

Figure 2.3 Parameter Passing.

(2) MOV AX, [BP+2] ; Move last variable to AX register
(3) MOV BX, [BP+1] ; Move next to last variable to BX

C. GRAPHICS ALGORITHMS

The following paragraphs provide a detailed explanation of each graphics routine in the Ada graphics library and the algorithms used to implement those routines. Several of the low level subroutines used to implement the library are not available to the user and are therefore not listed in the User's Guide (Appendix A). Only those graphics routines that have an apparent use in an Ada language program were made externally callable.

1. Clear Screen (CLS)

The CLS routine will clear the screen in one frame time (approximately 16.7 ms for 60 Hz operation). The CLS algorithm [Ref. 1: p. 4-46] is described in the following steps:

1. Input and save the video status from I O port DS(hex).
2. Blank the screen by outputting 0F(hex) via I O port DS(hex).
3. Output a "0" in bit 3 of the B control port DB(hex). This signals the video controller that the bit planes will be set to zero. This step is performed by executing an input, modify, and output sequence.
4. Output a "0" to bit 3 of the A control port DA(hex). This enables the video processor's CLRSCRN signal.
5. Wait for one frame period to allow time to reset all of video RAM.
6. Output a "1" to bit 3 of port DA(hex). This disables the CLRSCRN signal.
(7) Restore video status that was saved in step 1 by outputting the saved status via port D$1(hex).

2. Color Selection

When the color routine is called, it is passed an integer identifying the selected color. In normal operation, once a color has been selected, it remains the system color until changed by another call to the color procedure.

The color procedure uses the color code parameter to perform two basic functions. First, it sets each of the three color plane variables to base addresses which will be used by other procedures to initialize the E segment register for addressing the required color planes. Second, a control word ("$S hex") is output to the video control register via I O port D$1(hex). The control word disables the simultaneous write capability of the video controller. Table 3 lists the data used to control color selection.

<table>
<thead>
<tr>
<th>Color Code</th>
<th>Color</th>
<th>Var. 1</th>
<th>Var. 2</th>
<th>Var. 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>white</td>
<td>C000</td>
<td>D000</td>
<td>E000</td>
</tr>
<tr>
<td>-1</td>
<td>cyan</td>
<td>C000</td>
<td>D000</td>
<td>E000</td>
</tr>
<tr>
<td>-2</td>
<td>magenta</td>
<td>C000</td>
<td>D000</td>
<td>E000</td>
</tr>
<tr>
<td>-3</td>
<td>blue</td>
<td>C000</td>
<td>D000</td>
<td>E000</td>
</tr>
<tr>
<td>-4</td>
<td>yellow</td>
<td>D000</td>
<td>E000</td>
<td>D000</td>
</tr>
<tr>
<td>-5</td>
<td>green</td>
<td>E000</td>
<td>D000</td>
<td>E000</td>
</tr>
<tr>
<td>-6</td>
<td>red</td>
<td>D000</td>
<td>E000</td>
<td>D000</td>
</tr>
<tr>
<td>-7</td>
<td>black</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
</tbody>
</table>

In normal operation, control word bits 0, 1, 2, 7 are set to "0" and bits 4-6 are used to control the simultaneous write capability of video RAM. The color routine does not affect the contents of the E segment register, but simply initializes variables which will be used to modify the E segment register as required. This is done to maintain compatibility with other II Z-800 software which uses the E segment register, e.g., the standard I O library.

3. Video Ram Addressing

Mapping of the video to the screen is accomplished in two steps. The first is performed by the ADJ_SL procedure which accepts a Y position in the range 0-255
and maps that position to a displayable scan line. This mapping is described by the equation \( \text{Scan\_Line\_Number} = Y - (Y \mod 16) \). This function is implemented by a simple counter which adds seven to the scan line count for each block of 16 lines. Step two is performed by the REL\_VID\_ADDR procedure. This procedure uses the X position and scan line number to calculate the corresponding relative byte address in video RAM. This is a relative address since it is independent of the color plane. Relative byte addresses are calculated using the formula:

\[ \text{Relative\_Address} = Y \times 128 + X \mod 8. \]

Prior to address calculation, error checking is performed to insure that:

\[ 0 \leq X \leq 639 \quad \text{and} \quad 0 \leq \text{Scan\_Line\_Number} \leq 376. \]

4. **Pixel Display**

Displaying of individual pixels is performed by procedure SET\_PIXEL. Procedure SET\_PIXEL accepts an \((X, Y)\) coordinate and performs the following steps:

1. Calls procedure ADJ\_SL to map the Y coordinate to a displayable scan line.
2. Calls procedure REL\_VID\_ADDR to get the relative byte address.
3. Calculates the relative bit position within a byte based on the X coordinate.
4. Initializes segment E to provide the base address for the desired color plane.
5. Sets the selected pixel by writing to the relative address.
6. Repeats steps 4 and 5 for each of the three color plane variables.

5. **Line Drawing**

Line drawing may be performed by either procedure DRAW\_LINE or procedure DRAW\_MLINE. The two procedures use the same algorithm for line drawing with the difference being in the way that intersecting lines are displayed. Procedure DRAW\_LINE always displays the most recently drawn line on top while procedure DRAW\_MLINE mixes the colors of crossing lines at the point of intersection. The line drawing algorithm is an adaptation of a general integer digital differential analyzer (integer DDA) algorithm described by Marc Berger [Ref. 7; pp. 41-45]. The general integer DDA has been expanded from four to six cases and a smoothing function has been included. The two additional cases were added to handle the special cases where a line is either vertical or horizontal. The smoothing function was added to improve symmetry in the staircase effect which is inherent with raster scan displays.

The integer DDA algorithm is an iterative process which operates by taking unit steps along the X and or Y axis beginning at line start and continuing to line end.
The direction of each step is determined by an error variable which identifies whether the present position is above or below the ideal line. After each step, the error variable is updated based on the direction of movement. The initial value of the error variable is determined by the smoothing function which biases the error variable so that the first pixel step in one axis is delayed based on the slope of the line. Without this bias the DDA algorithm frequently makes an erroneous first move. An example of an extreme case illustrates this point. For example, if a near horizontal line with a single pixel change in \( Y \) is drawn by the unbiased algorithm the pixel step will occur at line start. However, in the biased algorithm the pixel step will occur at the approximate midpoint of the line. This provides an obvious enhancement in symmetry to a staircase line.

Implementation of the six cases in the line drawing algorithm assumes that the input coordinates are ordered so that \( Y_{\text{START}} \leq Y_{\text{END}} \). This ordering is enforced by the line drawing procedure prior to plotting a line.

In the description of the six cases which follows, initial values of \( X \) and \( Y \) are respectively \( X_{\text{START}} \) and \( Y_{\text{START}} \) and the following definitions apply.

- \( X \) \hspace{0.5cm} \text{X coordinate}
- \( Y \) \hspace{0.5cm} \text{Y coordinate}
- \( X_{\text{START}} \) \hspace{0.5cm} \text{X coordinate of line start}
- \( Y_{\text{START}} \) \hspace{0.5cm} \text{Y coordinate of line start}
- \( X_{\text{END}} \) \hspace{0.5cm} \text{X coordinate of line end}
- \( Y_{\text{END}} \) \hspace{0.5cm} \text{Y coordinate of line end}
- \( \Delta X \) \hspace{0.5cm} \text{X END - X START}
- \( \Delta Y \) \hspace{0.5cm} \text{Y END - Y START}
- \( E \) \hspace{0.5cm} \text{error}

\[
(1) \quad \Delta Y = 0.
\]

This is a horizontal line.

\[
\begin{align*}
\text{initialize} \\
X &= X_{\text{START}} \\
Y &= Y_{\text{START}} \\
\text{end initialize} \\
\text{repeat until } (X = X_{\text{END}}) \\
X &= X + 1 \\
\text{plot}(X, Y)
\end{align*}
\]
end repeat

(2) \( DY \leq DX, DX > 0, \) and \( DY > 0. \)
This is a line with positive slope between 0 and 1.
initialize
\[
\begin{align*}
X &:= X_{\text{START}} \\
Y &:= Y_{\text{START}} \\
E &:= DX/(DY+1)
\end{align*}
\]
end initialize
repeat until \((X=X_{\text{END}} \& Y=Y_{\text{END}})\)
    if \((E<0)\) then
        \[
        \begin{align*}
        X &= X+1 \\
        E &= E+DY
        \end{align*}
        \]
    otherwise
        \[
        \begin{align*}
        X &= X+1 \\
        Y &= Y+1 \\
        E &= E-DX+DY
        \end{align*}
        \]
    end if
end repeat

(3) \( DX < DY, DX > 0, \) and \( DY > 0. \)
This is a line with positive slope greater than 1.
initialize
\[
\begin{align*}
X &:= X_{\text{START}} \\
Y &:= Y_{\text{START}} \\
E &:= DY/(DX+1)
\end{align*}
\]
end initialize
repeat until \((X=X_{\text{END}} \& Y=Y_{\text{END}})\)
    if \((E<0)\) then
        \[
        \begin{align*}
        X &= X+1 \\
        Y &= Y+1 \\
        E &= E-DX+DY
        \end{align*}
        \]
    otherwise
        \[
        \begin{align*}
        Y &= Y+1 \\
        E &= E-DX
        \end{align*}
        \]
    end if
end repeat
(4) \( DX = 0 \).
This is a vertical line.

\begin{verbatim}
initialize
  X:=X_START
  Y:=Y_START
end initialize
repeat until (Y=Y_END)
  Y:=Y+1
end repeat
\end{verbatim}

(5) \( DY \geq \text{ABS}(DX) \), \( DX < 0 \), and \( DY < 0 \).
This is a line with negative slope between -1 and 0.

\begin{verbatim}
initialize
  X:=X_START
  Y:=Y_START
  E:=DX/(DY+1)
end initialize
repeat until (X=X_END & Y=Y_END)
  if (E<0) then
    X:=X-1
    E:=E+DY
  otherwise
    X:=X-1
    Y:=Y+1
    E:=E+DX+DY
  end if
end repeat
\end{verbatim}

(6) \( \text{ABS}(DX) < DY \), \( DX < 0 \), and \( DY < 0 \).
This is a line with negative slope less than -1.

\begin{verbatim}
initialize
  X:=X_START
  Y:=Y_START
  E:=DX/(DX+1)
end initialize
repeat until (X=X_END & Y=Y_END)
  if (E<0) then
    X:=X-1
    E:=E+DY
  otherwise
    X:=X-1
    Y:=Y+1
    E:=E+DX+DY
  end if
end repeat
\end{verbatim}
\[
\begin{align*}
X &= X - 1 \\
Y &= Y - 1 \\
E &= E + DX + DY
\end{align*}
\]

\textit{otherwise}
\[
\begin{align*}
Y &= Y + 1 \\
E &= E + DX
\end{align*}
\]

end repeat

6. Circle Drawing

Circle drawing is performed by passing the (X,Y) coordinate of the circle center and a radius to procedure CIRCLE. The algorithm executed by the circle drawing routine is an implementation of Bresenham's circle algorithm described by Hearn and Baker [Ref. 8; pp. 67-69]. In this implementation, the algorithm has been modified to include a correction factor to compensate for the X:Y pixel ratio of the raster display.

Bresenham's algorithm takes advantage of the symmetry of a circle in providing an efficient incremental method for plotting a circle. In this algorithm, eight points are plotted for each parameter calculation. Although multiplications are required in parameter calculations, the multiplier is a power of 2, so all multiplications can be reduced to a less costly shift operation. Figure 2-4 is a flow chart representation of the implemented version of Bresenham's algorithm.

7. Color Testing

The color of an individual pixel may be determined by passing an (X,Y) coordinate to procedure INQUIRE_COLOR. This procedure will determine the pixel color and return an integer value defining the color code. Color codes are listed in Table 2.

The color testing algorithm uses the (X,Y) coordinate to calculate a relative byte address. It then uses the X coordinate to generate a bit mask to identify the specific bit location within a byte. The relative byte address and bit mask are then used to test the corresponding address in each of the three color planes. This test identifies the color components contained in that pixel and a color code is returned to the calling program.
START

INIT. VARIABLES
X := 0
Y := RADIUS
P := 3 - 2 * RADIUS

PLOT CIRCLE POINTS
SET_PIXEL(XCENTER - X, YCENTER + Y/2)
SET_PIXEL(XCENTER - X, YCENTER + Y/2)
SET_PIXEL(XCENTER + X, YCENTER - Y/2)
SET_PIXEL(XCENTER - X, YCENTER - Y/2)
SET_PIXEL(XCENTER + Y, YCENTER + X/2)
SET_PIXEL(XCENTER - Y, YCENTER + X/2)
SET_PIXEL(XCENTER + Y, YCENTER + X/2)
SET_PIXEL(XCENTER - Y, YCENTER - X/2)
SET_PIXEL(XCENTER + Y, YCENTER - X/2)

X < Y

EXIT

P < 0
P := P + 4 * (X - Y) + 10

Y := Y - 1
P := P + 4 * X + 6

X := X + 1

Figure 2.4 Bresenham's Circle Algorithm.
S. Area Polygon Filling

Filling may be accomplished by calling either the procedure
BOUNDARY_FILL or procedure AREA_FILL. Both of these procedures use
algorithms based on a 4-connected boundary-filling algorithm described by Heiris and
Baker (Ref. 8, pp. 42-43). The 4-connected boundary-filling algorithm accepts as input
the (X,Y) coordinate of an interior point of an area bounded by the specified boundary
color and a fill color. Starting at the initial (X,Y) point, neighboring points (X-1,Y),
(X,Y-1), (X,Y+1), and (X+1,Y) are tested to determine if a different color than the boundary, then it is set to the fill color of the new point are tested. This procedure continues until all points within the area have been tested. Figure 2.5 is a recursive Pascal 4-connected boundary-filling procedure from Heiris and Baker (Ref. 8, p. 43), which forms the basis for the following procedures.

```
procedure boundary_fill(x, y, fill_color, boundary_color; integer);
var present_color: integer;
begin
  present_color := inquire_color(x, y);
  if (present_color <> boundary_color) and
    (present_color <> fill_color) then
  begin
    set_pixel(x, y, fill_color);
    boundary_fill(x-1, y, fill_color, boundary_color);
    boundary_fill(x+1, y, fill_color, boundary_color);
    boundary_fill(x, y-1, fill_color, boundary_color);
    boundary_fill(x, y+1, fill_color, boundary_color);
  end
end;
```

Figure 2.5 Pascal Boundary-Fill Procedure.

Procedure BOUNDARY_FILL is a recursive routine whose input parameters include a beginning (X,Y) coordinate, a fill color, and a boundary color. This routine uses the 4-connected fill algorithm to fill an area bounded by the specified boundary color.

Procedure AREA_FILL is a recursive routine whose input parameters include only a beginning (X,Y) coordinate and a fill color. This routine uses the 4-connected fill algorithm to fill a contiguous area of the same color as that at the beginning (X,Y) coordinate.
The actual implementation of each of the two fill procedures is subdivided into two procedures. The first procedure initializes constant variables and then calls the second procedure. The second procedure is used for successive recursive calls. This technique minimizes the number of parameters which must be passed on each successive recursive call.

An individual call to either of the fill procedures can fill an area of at most a few square inches. Attempting to fill a larger area may result in a stack overflow error. This is an implementation limitation which can be circumvented by subdividing a large area into smaller areas, each of which may be filled separately. An exact upper limit on the size of an area which may be filled by a single call to a fill procedure is difficult to predict since it depends on the starting point within a specified area. Additionally, area filling is an inherently slow process due to the large number of calculations involved.

9. Cursor Control

The "+" symbol is used to represent a graphics cursor. Cursor positioning is controlled by procedures SET_CURS and RESET_CURS. Procedure SET_CURS accepts an (X,Y) coordinate and displays a cursor centered at that coordinate. Procedure RESET_CURS accepts an (X,Y) coordinate as input and erases a block of addresses surrounding that coordinate. Both procedures affect only the color planes that are enabled by the current system color.

Management of the cursor display and positioning is a responsibility of the user program. A cursor is not automatically displayed nor is there a limit on the number of cursor symbols which may be displayed concurrently.

The cursor symbol is composed of nine pixels with the center of the cursor being the reference point for positioning. In order to be able to position the cursor at any (X,Y) screen coordinate, it is necessary to consider individual components of the cursor. The vertical dimension (5 pixels) of the cursor spans five different relative byte addresses in video RAM. Additionally, the horizontal dimension (5 pixels) of the cursor may require an additional byte of memory. Whether an additional byte is required depends on the relative position within a byte of the cursor center. If the cursor center is near either end of a byte, then either the next lower or next higher byte from cursor center will also be required to display the cursor.

The set cursor algorithm uses the (X,Y) coordinate to calculate five relative byte addresses. These addresses represent a block on the screen which will contain the
cursor. The algorithm then uses the X coordinate to determine which of the eight bit positions within a byte represents cursor center. Then one of eight possible bit patterns is generated to represent the cursor. Resetting a cursor requires less program steps since a block of seven relative byte addresses are erased without regards to the exact position of the cursor within the block.
III. PERFORMANCE AND EVALUATION

A. DEMONSTRATION PROGRAMS

Four demonstration programs are provided in Appendices B-E. Those programs, along with the user’s guide in Appendix A, will aid the user in writing graphics programs using the Ada language.

1. Program AGTEST1

Program AGTEST1 draws a test pattern demonstrating the use of color selection, line drawing, circle drawing, and filling. This test uses ADAGRAPH procedures COLOR, SET_CURS, CIRCLE, DRAW_LINE, BOUNDARY_FILL, and AREA_FILL.

2. Program AGTEST2

Program AGTEST2 draws a pattern consisting of two sets of curves and a color wheel illustrating the eight available colors. AGTEST2 uses ADAGRAPH procedures COLOR, DRAW_LINE, DRAW_MLINE, CIRCLE, and AREA_FILL.

3. Program AGTEST3

Program AGTEST3 demonstrates interactive control of the graphics cursor. The user is asked repetitively to enter an (X,Y) coordinate. The program then displays a cursor centered at that coordinate and erases the previous cursor.

4. Program AGTEST4

Program AGTEST4 begins by drawing a circle of the approximate maximum area that can be filled by a single call to one of the fill procedures. It then fills the circle and goes on to demonstrate how polygons may be drawn by setting resetting individual pixels. It also demonstrates filling an area containing an object. AGTEST4 uses ADAGRAPH procedures COLOR, SET_PIXEL, REQUEST_PIXEL, CIRCLE, BOUNDARY_FILL, and AREA_FILL.

B. ADA GRAPHICS LIBRARY LIMITATIONS

The Ada graphics library presented in this thesis was developed as a prototype. Limitations of the library procedures should not cause significant problems to the user as long as the guidelines presented in the user’s guide (Appendix A) are followed. The only known procedures which can cause a catastrophic failure if limits are exceeded are the fill procedures. The fill procedures can fill at most a contiguous area of
approximately seven square inches. Even filling a seven square inch area may cause an error if the starting point is not near the center of the area. However, if the user will limit the size of contiguous areas to be filled to approximately 3-4 square inches, then the probability of failures is very low. The reason for this limitation is that the fill routines are implemented as recursive procedures. As a recursive procedure, each level of recursion requires that three more parameters be pushed onto the system stack. Attempting to fill too large an area simply results in a stack overflow.

Another area of inconvenience for the user may occur in attempting to use the graphics library along with other libraries, e.g., the standard I O library. The standard I O library procedure GET automatically causes scrolling of the display screen by one character row each time it is executed. This is an obvious problem when used in concurrence with a fixed position graphics display.

C. H/Z-100 HARDWARE LIMITATIONS

A potential problem area for the user is due to the limited number of colors ($S$) available for display. This may be particularly inconvenient when writing interactive programs using the graphics cursor or any other symbols which are to be moved within the display. Management of movable objects can become very costly in terms of both processing time and amount of program code. The H Z-100, while comparable to many other microcomputers, is still quite slow in displaying a complex graphics picture. One method of optimizing the management of moving objects in the display is to restrict those objects to a dedicated color plane. If this is done, then updating the position of a moved object can be done without disturbing the graphics objects defined in the other color planes. A disadvantage of this approach is that there are only three color planes available to begin with and by dedicating one color plane to something like a cursor reduces the colors available for other objects from eight to four.

D. TIMING MEASUREMENTS

Table 4 provides a summary of timing measurements performed to determine average graphics processing times for the most critical operations. Times represented in the table are average times that one would expect to encounter in a call to Ada graphics library procedures when drawing a graphics picture.

The fill procedures are quite slow due to the large number of steps involved in testing and setting each pixel and the additional overhead related to a recursive procedure. The timing differences between drawing horizontal/vertical lines and
diagonal lines is due to the two additional cases which were implemented in the line drawing algorithm. Those cases were added to handle the special cases where a line is either vertical or horizontal. In those cases, since the slope of the line is not used in plotting the line, less calculations are required than when plotting a diagonal line.

Setting resetting of individual pixels by all of the graphics procedures is slowed by the overhead in address calculation of the bit mapped display. This is primarily a weakness in the design of the video processor of the H Z-100. However, it would be interesting to see if a significant speedup could be realized if a lookup table was used to aid in pixel addressing instead of a pure mathematical addressing algorithm.
IV. RECOMMENDATIONS FOR FUTURE DEVELOPMENT

Any future additions or modifications to the Ada graphics library should consider the following:

1) The feasibility of implementing a nonrecursive fill routine.

2) Development and implementation of different I/O procedures that are more compatible with a graphics display to replace those in the standard I/O library. In particular, these procedures should allow placement of alphanumeric symbols at any screen coordinate, not just the predefined character positions. Any scrolling of the screen should also be strictly under the control of the user program.

3) The possibility of optimizing some of the existing procedures. For example, perhaps the mapping from a Y coordinate to a displayable scan line could be performed more efficiently by a lookup table. But, would the time savings justify the additional memory required?

4) The feasibility of developing and implementing a subset of the GKS standard at a level of abstraction above the primitives provided in the Ada graphics library.

5) The feasibility of networking two or more HZ-100 computers to provide interactive operator communication on one system while providing a graphics display on another.

6) The feasibility of modifying the structure of the Ada graphics library so that it is compatible with the MS-DOS operating system hosted on the HZ-100.
V. CONCLUSION

The implementation of the Ada graphics library on the H Z-100 computer has proven the feasibility of developing Ada graphics programs on a microcomputer. The H Z-100 is certainly suitable for educational purposes but its ability to meet real-time tactical requirements is limited. This limitation is not unique to the H Z-100, but is a general limitation of most microcomputers available today. We can expect this limitation to ease significantly over the next several years as low cost multiprocessor systems are developed.

The Ada graphics library was implemented as low level primitives necessary to interface with and control the H Z-100 hardware. The library functions were not designed in accordance with any recognized standards. However, the functions were designed with graphics standards in mind. The implementation of a higher level GKS standard graphics library would be able to use most of the present Ada graphics library primitives to interface with the H Z-100 hardware. The implementation of a GKS standard as a level above the machine dependent Ada graphics library would assist in making any developed Ada language graphics programs transportable to other computers. The benefits which may be realized from a standard interface to a language are obvious with the high costs of developing software. Although the architectural limitations of the H Z-100 are not compatible with a full implementation of the GKS standard, the educational benefits that could be realized from even an implementation of a subset of the GKS standard warrants further investigation.

The limited capabilities of the H Z-100 may limit its suitability for many operational tactical applications, however, it can be a useful system for program development. With the support of the Ada graphics library, the H Z-100 can assist in performing further research into such tactical programming applications as the Navy Tactical Data System (NTDS). The ability to perform research in this area could be further enhanced by the development of a GKS standard graphics library on the H Z-100 computer.
APPENDIX A
USER'S GUIDE

1. INTRODUCTION

Twelve graphics procedures are available to the user. These procedures are included in ADAGRAFHLIB which is callable from the Ada programming language. Appendices B-E provide four example programs demonstrating the linkage and calling procedures necessary to use the Ada graphics library. Procedures available to the user are:

(1) CLS
(2) COLOR(color_code: in INTEGER)
(3) INQUIRE_COLOR(x_coord, y_coord: in INTEGER;
   color_code: out INTEGER)
(4) SET_PIXEL(x_coord, y_coord: in INTEGER)
(5) RESET_PIXEL(x_coord, y_coord: in INTEGER)
(6) DRAW_LINE(x_start, y_start, x_end, y_end:
in INTEGER)
(7) DRAW_MLINE(x_start, y_start, x_end, y_end:
in INTEGER)
(8) CIRCLE(x_center, y_center, radius: in INTEGER)
(9) BOUNDARY_FILL(x_coord, y_coord, fill_color,
   boundary_color: in INTEGER)
(10) AREA_FILL(x_coord, y_coord, fill_color, boundary_color: in INTEGER)
(11) SET_CURS(x_coord, y_coord: in INTEGER)
(12) RESET_CURS(x_coord, y_coord: in INTEGER)

Following the description of each function is an example of the Ada language call to the described procedures. It should be noted that for all procedures that require in (X,Y) coordinates that the allowable range of X is \((0,399)\) and the allowable range of Y \((0,240)\).

2. CLEARING THE SCREEN

Most graphics programs will require clearing the display screen before displaying graphics data. This is best performed by calling procedure CLS. This procedure takes
advantage of a built-in hardware feature which allows resetting all of video RAM in one frame period (approximately 16.7 ms).

-- Clear the screen
CLS:

3. COLOR SELECTION

The H Z-100 has the capability of displaying up to eight different colors. Color selection is performed by calling procedure COLOR and passing a color code parameter identifying the desired color. Table 5 lists the colors available and their respective color codes. Black is not listed as a color option since it is the only color which is displayed by resetting a pixel, i.e., writing a '0' to that pixel's corresponding addresses in video RAM. All other colors are displayed by writing a '1' in the appropriate relative address of the color planes required to produce the desired color. If a pixel is to be reset to black then the RESET_PIXEL procedure should be used.

--Set system color to green
COLOR(5).

<table>
<thead>
<tr>
<th>COLOR CODE</th>
<th>COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>white</td>
</tr>
<tr>
<td>1</td>
<td>cyan</td>
</tr>
<tr>
<td>2</td>
<td>magenta</td>
</tr>
<tr>
<td>3</td>
<td>blue</td>
</tr>
<tr>
<td>4</td>
<td>yellow</td>
</tr>
<tr>
<td>5</td>
<td>green</td>
</tr>
<tr>
<td>6</td>
<td>red</td>
</tr>
</tbody>
</table>

4. COLOR TESTING

To determine the color to which any particular pixel has been set, use procedure INQUIRE_COLOR. Input parameters to this procedure are the (X,Y) coordinate of the selected pixel. The procedure returns the color code (Table 5) of the pixel located at that coordinate.

--Get color of pixel at (X,Y) coordinate (5,100)
INQUIRE_COLOR(59, 60, color);

5. PIXEL SETTING/RESETTING

Procedures SET_PIXEL and RESET_PIXEL are provided for controlling individual pixels. Procedure SET_PIXEL accepts an (X,Y) coordinate as input parameters and sets the corresponding pixel to the current system color. Procedure RESET_PIXEL accepts an (X,Y) coordinate as input parameters and resets the corresponding pixel to black.

-- Set pixel at location (70,50) to current system color
SET_PIXEL(70, 50);
-- Reset pixel at location (50,50) to black
RESET_PIXEL(50, 50);

6. LINE DRAWING

Procedures DRAW_LINE and DRAW_MLINE are provided for drawing lines. Both procedures accept as input parameters the (X,Y) coordinates of the end points of a line. A line of the current system color is then drawn connecting those end points. If procedure DRAW_LINE is used, then intersecting lines will be displayed with the last line drawn on top. If procedure DRAW_MLINE is used for drawing lines then the color at the point of intersection of intersecting lines is the combined color of the individual lines.

-- Draw a line from coordinate (100,50) to
-- coordinate (200,75). This line will be
-- drawn on top of any line it intersects.
DRAW_LINE(100, 50, 200, 75);
-- Draw a line from coordinate (100,25) to
-- coordinate (19,15). This line will mix
-- colors at the point of intersection with
-- any lines that it intersects.
DRAW_MLINE(100, 25, 19, 15);

7. CIRCLE DRAWING

Procedure circle will draw a circle of the present system color. Input parameters to this procedure include an (X,Y) coordinate identifying the circle's center position and a radius length measured in units based on the number of pixels (640) in the X axis.
of the video display. The circle drawing algorithm draws a circle with automatic corrections made based on the xy linear pixel ratio (2:1) of the color monitor. i.e., the circle radius is twice as many pixels in the X axis as it is in the Y axis.

--Draw a circle of radius 50 units and
--centered at coordinate (300, 100).
CIRCLE(300, 100, 50);

8. AREA/POLYGON FILLING

Procedures BOUNDARY_FILL and AREA_FILL are available to perform filling operations. The user is cautioned that the filling algorithm has an upper limit on the size of an area (approximately seven square inches) which may be filled by a single call to either of the fill procedures. Larger areas may be filled by partitioning the larger area and filling the individual partitions. The user can generally avoid problems by limiting all fills to a contiguous area of not more than 3-4 square inches.

Procedure BOUNDARY_FILL accepts as input parameters an (X,Y) coordinate identifying the starting point, a fill color code, and a boundary color code (Table 5). The procedure begins filling from the starting (X,Y) coordinate and sets all pixels within the defined boundary to the designated fill color. The boundary must be a single color defined by the boundary color code.

Procedure AREA_FILL accepts as input parameters an (X,Y) coordinate identifying the starting point and a fill color code (Table 5). The procedure starts at the input (X,Y) coordinate and sets all pixels in a contiguous area which are the same color as the pixel at the (X,Y) starting coordinate to the specified fill color. This procedure can fill an area which is bounded by different colors since filling is based on the color of the area to be filled.

--Beginning at coordinate (50,75), fill an area
--with blue that has a red boundary.
BOUNDARY_FILL(50, 75, 5, 6);
--Fill a contiguous area with green. An internal
--point in that area is coordinate (100,100).
AREA_FILL(100, 100, 5);

1. CURSOR CONTROL

Procedures SET_CURS and RESET_CURS are provided for cursor control. The graphics cursor controlled by these procedures is a "-". The center of the cursor is
used as a reference point for cursor placement. All management of cursor display is the responsibility of the user. The cursor is simply a special graphics symbol which the user may display and erase. The number of cursor symbols which may be displayed at any point in time is determined solely by the user's program.

Procedure SET_CURS accepts as input an (X,Y) coordinate and displays a cursor symbol of the present color at that coordinate. Procedure RESET_CURS accepts as input an (X,Y) coordinate and resets a block of relative memory addresses centered at the corresponding (X,Y) coordinate. The color planes affected are determined by the present system color. It is suggested that if the user desires to use the cursor as a background symbol then the cursor color should be restricted to a single color plane (red, green, or blue) that is dedicated to cursor display. If this is not done then a management scheme must be implemented in the user program to maintain the integrity of non-cursor graphics data when setting resetting the cursor.

--Display a green cursor at coordinate (150,100).
COLOR(5);
SET_CURS(150, 100);

--Erase a green cursor at coordinate (50,75).
COLOR(5);
RESET_CURS(50, 75);
APPENDIX B

GRAPHICS TEST 1

WITH IO, UTIL, ADAGRAPH;
PACKAGE BODY AGTEST1 IS
USE IO, UTIL, ADAGRAPH;
--
-- THIS PROGRAM DRAWS A TEST PATTERN TO TEST COLOR SELECTION,
-- CURSOR SETTING, CIRCLE DRAWING, LINE DRAWING, AREA FILLING,
-- AND BOUNDARY FILLING
--
x_pos, y_pos : integer;
quit : character;
begin
CLS;
x_pos := 320;
y_pos := 108;
   color(5): -- GREEN
set_curs(300, 135);
color(3); -- BLUE
-- DRAW A CIRCLE
   circle(300, 102, 142);
color(6); -- RED
-- DRAW A RECTANGLE
   draw_line(153, 31, 442, 31);
   draw_line(442, 31, 442, 173);
   draw_line(442, 173, 158, 173);
   draw_line(158, 173, 158, 31);
-- DRAW A CIRCLE AND FILL IT
   circle(300, 102, 40);
   area_fill(300, 102, 3);
-- DRAW AN IRREGULAR POLYGON AND FILL IT
   draw_line(20.20, 30.32);
   draw_line(60.22, 84.30);
   draw_line(84.50, 174.33);
   draw_line(174.43, 20.20);
   boundary_fill(50.35, 6.6);
-- DRAW A TRIANGLE AND FILL IT
   draw_line(50.100, 50.130);
   draw_line(50.130, 25.130);
   draw_line(25.130, 50.120, 5);
   area_fill(50.120, 5);
   color(2); -- MAGENTA
-- DRAW A RECTANGLE
   draw_line(10, 10, 590, 10);
   draw_line(590, 10, 590, 205);
   draw_line(590, 205, 10, 205);
   draw_line(10, 205, 10, 10);
   color(5);
-- DRAW A RECTANGLE WITH DIAGONAL LINES
   draw_line(200, 52.400, 52);
   draw_line(400, 52.400, 152);
   draw_line(400, 152.200, 152);
   draw_line(200, 152.200, 52);
   draw_line(255, 152.400, 152);
   get(quit);
end AGTEST1;
WITH IO, UTIL, ADAGRAPH;
PACKAGE BODY AGTST2 IS
- begin  
-- THIS PROGRAM EXERCISES THE LINE DRAWING, CIRCLE,
-- AND AREA FILLING PROCEDURES TO PRODUCE A PICTURE.
-- WITH COLORED CURVES AND A COLOR WHEEL.

  x, y, y1, y2, pool : integer;
  quit : character;
  begin
    x := 0;
    y1 := 0;
    y := 0;
    y1 := 0;
    color(0);
    -- LOOP TO DRAW CURVE 1
    for i in 0..63 loop
      x := i * 10;
      y2 := 199 - y1;
      draw_line(x, y, 0, y2);
      y := y + 2;
      y1 := y1 + 3;
    end loop;

    y := 0;
    y1 := 0;
    color(1);
    -- LOOP TO DRAW CURVE 2
    for i in 0..63 loop
      x := 630 - (i * 10);
      y2 := 199 - y1;
      draw_line(x, y, 630, y2);
      y := y + 2;
      y1 := y1 + 3;
    end loop;

    -- DRAW A COLOR WHEEL
    color(0); -- WHITE
    circle(320, 150, 100);
    draw_line(320, 100, 320, 200);
    draw_line(220, 150, 420, 150);
    draw_line(249, 115, 391, 185);
    draw_line(391, 115, 249, 185);
    area_fill(91, 140, 0); -- COLOR 0 = WHITE
    area_fill(91, 140, 1); -- COLOR 1 = CYAN
    area_fill(91, 140, 2); -- COLOR 2 = MAGENTA
    area_fill(91, 140, 3); -- COLOR 3 = BLUE
    area_fill(91, 140, 4); -- COLOR 4 = YELLOW
    area_fill(91, 140, 5); -- COLOR 5 = GREEN
    get faults;
  end AGTST2;
WITH IO, UTIL, ADAGRAPH;
PACKAGE BODY AGTEST3 IS
USE IO, UTIL, ADAGRAPH;
--
-- THIS PROGRAM IS USED TO TEST SETTING AND RESETTING
-- OF A GRAPHICS CURSOR. THE PROGRAM REQUESTS CURSOR
-- POSITION INPUTS, ERASES THE PREVIOUS CURSOR, AND
-- DRAWS A CURSOR AT THE INPUT (X,Y) COORDINATE
--
i : integer;
x_pos, y_pos, x_old, y_old : integer;
begin
CLS;
x_old := 0; --INITIALIZE X & Y
y_old := 0;
for i in 1..50 loop
  put("ENTER X POSITION ");
  get(x_pos);
  put("ENTER Y POSITION ");
  get(y_pos);
  color(5); --GREEN
  reset_curs(x_old, y_old);
  set_curs(x_pos, y_pos);
  x_old := x_pos;
  y_old := y_pos - 13;
end loop;
end AGTEST3;
WITH IO, UTIL, ADAGRAPH;
PACKAGE BODY AGTEST4 IS
USE IO, UTIL, ADAGRAPH;
-- THIS PROGRAM IS USED TO DISPLAY A CIRCLE OF THE APPROXIMATE
-- MAXIMUM AREA THAT CAN BE FILLED BY A SINGLE CALL TO ONE OF
-- THE FILL ROUTINES. IT THEN DEMONSTRATES FILLING AND
-- DRAWING OF POLYGONS USING ONLY THE SET_PIXEL AND RESET_PIXEL
-- PROCEDURES.
x, y : INTEGER;
quit : character;
Begin

-- SELECT COLOR
color(0); "WHITE"
-- DRAW A CIRCLE
circle(300, 102, 115);
-- FILL THE CIRCLE WITH RED
boundary_fill(300, 102, 6, 0);
color(1); "Cyan"
-- DRAW A RECTANGLE USING SET PIXEL
for x in 250..350 loop
  set_pixel(x, 77);
  set_pixel(x, 127);
end loop;
for y in 77..127 loop
  set_pixel(250, y);
  set_pixel(350, y);
end loop;
-- FILL THE RECTANGLE
area_fill(300, 132, 5); "Color 5 = Green"
-- ERASE A RECTANGLE USING RESET PIXEL
for x in 280..320 loop
  for y in 92..112 loop
    reset_pixel(x, y);
  end loop;
end loop;
area_fill(375, 102, 2);
color(8);
get(quite);
end AGTEST4;
APPENDIX F
SPECIFICATION PACKAGE

PACKAGE ADAGRAPH IS
  -- RETURN ADDRESS VARIABLES
  RET_SLA : RET_SLA, RET_SLA, RET_IC, RET_XN: INTEGER;
  RET_SLA : RET_AF, RET_CL, RET_SPE, RET_CPE: INTEGER;
-- CURSOR ADDRESS VARIABLES
  CURS_0, CURS_1, CURS_2, CURS_3, CURS_4: INTEGER;
  XLOC: INTEGER;
-- X, Y POSITION AND GENERAL PURPOSE VARIABLES
  X_POS, Y_POS, DELTA_X, DELTA_Y, INC_CTR: INTEGER;
  ERROR, END_CUT: INTEGER;
  X_TR, Y_CTR, RADIUS, X_REL, Y_REL, P.VAL: INTEGER;
-- COLOR VARIABLES
  P_COLOR, F_COLOR, B_COLOR: INTEGER;
-- SYSTEM STATUS
  SYS_STAT: INTEGER;
-- SEGMENT STATUS
  SEGMENT_FL, SEGMENT_FL, SEGMENT_IE: INTEGER;
-- COLOR PLANE BASE ADDRESS
  COL_P_ADDR: INTEGER;
  COL_P_ADDR, COL_P_ADDR, COL_P_ADDR: INTEGER;
-- FILL ROUTINE CONTROL VARIABLE
  COL_NX: INTEGER;
-- GRAPHICS LIBRARY PROCEDURES
  PROCEDURE CLS;
  PROCEDURE COLOR(color_code : in INTEGER);
  PROCEDURE SET_CURS(x_pos, y_pos : in INTEGER);
  PROCEDURE RESET_CURS(x_pos, y_pos : in INTEGER);
  PROCEDURE DRAW_LINE(x_start, y_start, x_end, y_end : in INTEGER);
  PROCEDURE DRAW_MLINE(x_start, y_start, x_end, y_end : in INTEGER);
  PROCEDURE CIRCLE(x_ctr, y_ctr, radius : in INTEGER);
  PROCEDURE INQUIRE_COLOR(x_pos, y_pos : in INTEGER);
  PROCEDURE BOUNDARY_FILL(x_pos, y_pos, f_color, b_color : in INTEGER);
  PROCEDURE AREA_FILL(x_pos, y_pos, f_color : in INTEGER);
  PROCEDURE SET_PIXEL(x_pos, y_pos : in INTEGER);
  PROCEDURE RESET_PIXEL(x_pos, y_pos : in INTEGER);
END ADAGRAPH;

42
APPENDIX G
ASSEMBLY CODE LISTING

PACKAGE ASSEMBLY ADAGRAPH

;**********************************************************************************************
;**********************************************************************************************
;PROCEDURE CLS IS USED TO PERFORM A CLEAR SCREEN OPERATION
PROC CLS;
    PUSH AX ;SAVE VIDEO STATUS
    PUSH AX
    MOV AX, OD8H ;BLANK THE SCREEN
    MOV AX, OFH
    OUT OD8H, AL ;SET BIT 3 = 0
    IN AX, OD9H
    AND AX, OF7H
    OUT OD9H, AL ;ENABLE CLRSCRN SIGNAL
    MOV CX, 660 ;INIT COUNTER FOR TIME DELAY
    DELAY:
    LOOP DELAY ;TIME DELAY TO ALLOW VIDEO PROCESSOR
    IN AX, OD9H
    OR AX, 08H
    OUT OD9H, AL ;DISABLE CLRSCRN SIGNAL
    POP AX
    OUT OD3H, AL ;RESTORE SYSTEM STATUS
    POP AX
    RET

END PROC CLS;

;**********************************************************************************************
;**********************************************************************************************
;PROCEDURE COLOR IS USED TO ENABLE COLOR PLANES TO PROVIDE THE SELECTED COLOR AND THE E SEGMENT REGISTER IS INITIALIZED TO ALLOW WRITING TO THE ENABLED COLOR PLANES.
PROC COLOR;
    POP DX ;SAVE RETURN ADDRESS
    POP AX ;GET COLOR CODE
    PUSH AX ;RESTORE RETURN ADDRESS
    MOV AX, 7 ;MASK COLOR CODE TO SET STATUS FLAGS
    JNZ COLOR1
    MOV [COL_PL1], OC000H ;COLOR IS WHITE
    MOV [COL_PL2], OD000H
    MOV [COL_PL3], OE000H
    JMP EXIT_COLOR1
COLOR1:
    SUB AL, 1
    JNZ COLOR2
    MOV [COL_PL1], OC000H ;COLOR IS CYAN
    MOV [COL_PL2], OE000H
    MOV [COL_PL3], OD000H
    JMP EXIT_COLOR1
COLOR2:
    SUB AL, 1
    JNZ COLOR3
    MOV [COL_PL1], OC000H ;COLOR IS MAGENTA
    MOV [COL_PL2], OD000H
    MOV [COL_PL3], OC000H
    JMP EXIT_COLOR1
COLOR3:
    SUB AL, 1
    JNZ COLOR4
    MOV [COL_PL1], OC000H ;COLOR IS BLUE
    MOV [COL_PL2], OC000H
    MOV [COL_PL3], OC000H
    JMP EXIT_COLOR1
    EXIT_COLOR1

}
COLOR4: SUB AL, 1
    MOV [COL_PL1], OD000h ;COLOR IS YELLOW
    MOV [COL_PL2], OD000h
    MOV [COL_PL3], OD000h
    JMP EXIT_COL
COLOR5: SUB AL, 1
    MOV [COL_PL1], OD000h ;COLOR IS GREEN
    MOV [COL_PL2], OD000h
    MOV [COL_PL3], OD000h
    JMP EXIT_COL
COLOR6: SUB AL, 1
    MOV [COL_PL1], OD000h ;COLOR IS RED
    MOV [COL_PL2], OD000h
    MOV [COL_PL3], OD000h
    JMP EXIT_COL
EXIT_COL: MOV AL, 78h ;DISABLE SIMULTANEOUS WRITE
    OUT OD8h, AL
ENDPROC_COL:

PROCEDURE RESET_CURS IS USED TO ERASE THE GRAPHICS CURSOR:
AN (X,Y) POSITION IS INPUT TO THE PROCEDURE AND THE A BLOCK OF
ADDRESSES IS CLEARED AT THE CURSOR LOCATION.
PROC RESET_CURS:
    MOV [SEGMENT_E], ES ;SAVE SEG E
    POP AX ;SAVE RETURN ADDRESS
    POP [Y_POS] ;GET Y POSITION
    POP [X_POS] ;GET X POSITION
    PUSH AX ;RESTORE RETURN ADDRESS
    PUSH [Y_POS]
    PUSH [X_POS]
    CALL ADJ_SL ;ADJUST Y POSITION TO A DISPLAYABLE
    CALL REL_VID_ADDR ;SCAN LINE & GET VIDEO ADDRESS
    POP BX
    CALL BLANK_GCURS
    SUB BX, 1
    CALL BLANK_GCURS
    SUB [Y_POS], 1
    CALL ADJ_SL
    CALL REL_VID_ADDR
    POP BX
    CALL BLANK_GCURS
    ADD [Y_POS], 3
    CALL ADJ_SL
    CALL REL_VID_ADDR
    POP [Y_POS]
    CALL BLANK_GCURS
    CALL BLANK_GCURS
    CALL BLANK_GCURS

44
MOV ES, [SEGMENT_E] ;RESTORE SEG E

END_PROC RESET_CURS;

;**********************************************************************************
;PROCEDURE BLANK_GCURS IS USED BY PROCEDURE RESET_CURS TO ERASE A CURSOR

PROC BLANK_GCURS:
MOV AX, [COL_P11]
MOV ES, AX
SEG ES
MOV [BX], 0
MOV AX, [COL_P2]
MOV ES, AX
SEG ES
MOV [BX], 0
MOV AX, [COL_P3]
MOV ES, AX
SEG ES
MOV [BX], 0
RET

END_PROC BLANK_GCURS;

;**********************************************************************************
;PROCEDURE SET_CURS IS USED TO DISPLAY A CURSOR AT THE INPUT

; (X,Y) POSITION. THE GRAPHICS CURSOR DISPLAYED IS A "+".

PROC SET_CURS:
;CALCULATE MEMORY ADDRESS OF NEW CURSOR POSITION
MOV [SEGMENT_E], ES ;SAVE SEG E STATUS
POP AX ;SAVE RETURN ADDRESS
POP [Y_POS] ;GET Y POSITION FROM STACK
POP [X_POS] ;GET X POSITION FROM STACK
PUSH AX ;REPLACE RETURN ADDRESS
MOV DX, [Y_POS]
PUSH [Y_POS]
call ADJ_SL
PUSH BX
;GET CORRECTED Y LINE NUMBER

;THE CURSOR IS MADE UP OF FIVE COMPONENTS LABELED CURS_0 TO CURS_4
MOV [CURS_0], DX ;PUT THE LINE NUMBER INTO
MOV [CURS_1], DX ;EACH OF THE CURSOR ROWS
MOV [CURS_2], BX
MOV [CURS_3], DX
MOV [CURS_4], DX
SUB [CURS_0], 2
MOV AX, [CURS_0]
call CO
MOV AX, 3
ADD [CURS_0], AX ;ADJUST EACH CURSOR ROW TO REFLECT IT
CALL ADJ_SL ;WHICH IS CURS_2
POP [CURS_0] ;THEN CALL PROCEDURE ADJ_SL TO ASSURE
SUB [CURS_1], 1 ;THAT EACH CURSOR ROW IS LOCATED ON
MOV AX, [CURS_1] ;A DISPLAYABLE SCAN LINE
call CI
PUSH [CURS_1] ;2
CALL ADJ_SL
POP [CURS_1]
ADD [CURS_3], 1
call AC
PUSH [CURS_3] ;1
CALL ADJ_SL
ADD [CURS_4], 2
PUSH [CURS_4]
PUSH BX
CALL REL_VID_ADDR

RTRUN
POP BX
PUSH [X_POS]
PUSH [Y_POS]
POP AX
MOV AL, [CURS_2]
SUB AL, 1
CALL DISP_GCURS
MOV AL, 7FH
ACD BX, 1
CALL DISP_GCURS
MOV AL, 80H
MOV BX, [CURS_0]
CALL DISP_GCURS
MOV BX, [CURS_1]
CALL DISP_GCURS
MOV BX, [CURS_3]
CALL DISP_GCURS
MOV BX, [CURS_4]
CALL DISP_GCURS
JMP EXIT

BIT1:
SUB AL, 1
JMP BIT2

BIT2:
SUB AL, 1
JMP BIT3

BIT3:
SUB AL, 1
JMP BIT2
BIT3:
SUB AL, 1
JNZ BIT4
; DECREMENT BIT COUNT FOR NEXT TEST
; TEST FOR CURSOR IN X PIXEL POS. 3

BIT4:
SUB AL, 1
JNZ BIT5
; DECREMENT BIT COUNT FOR NEXT TEST
; TEST FOR CURSOR IN X PIXEL POS. 4

BIT5:
SUB AL, 1
JNZ BIT6
; DECREMENT BIT COUNT FOR NEXT TEST
; TEST FOR CURSOR IN X PIXEL POS. 5

BIT6:
SUB AL, 1
JNZ BIT7
; DECREMENT BIT COUNT FOR NEXT TEST
; TEST FOR CURSOR IN X PIXEL POS. 6
MOV BX, [CURS_4]
CALL DISP_CURS
MOV AL, 7
CALL DISP_CURS
MOV AL, [CURS_2]
CALL DISP_CURS
ADD AL, BX
CALL DISP_CURS
MOV BX, [CURS_0]
CALL DISP_CURS
MOV BX, [CURS_2]
CALL DISP_CURS
MOV BX, [CURS_3]
CALL DISP_CURS
MOV BX, [CURS_4]
CALL DISP_CURS

EXIT:
MOV ES, [CURS_4] ;RESTORE SEG E

END_PROC SET_CURS;

PROCEDURE DISP_CURS IS USED BY PROCEDURE SET_CURS TO DISPLAY A CURSOR

PROC DISP_CURS:

MOV [COL_PLL], BX
MOV [COL_PLL], AL
MOV [COL_PLL], BX
MOV [COL_PLL], AL
MOV [COL_PLL], BX
MOV [COL_PLL], AL

END_PROC DISP_CURS:

PROCEDURE ADJ_SL IS USED TO CORRECT Y POSITION TO INSURE IT OCCURS ON A DISPLAYABLE SCAN LINE

PROC ADJ_SL:

POP BX ;SAVE RETURN ADDRESS
POP BX
POP BX ;GET Y POSITION
TST_Y:
SUB AL, 9
JNS ADJ_Y
JMP EXIT_SL
ADJ_Y:
ADD BX, 1
JMP EXIT_SL
EXIT_SL:
PUSH BX ;RETURN CORRECTED Y POSITION
PUSH [RET_ADJ_SL] ;RESTORE RETURN ADDRESS

END_PROC ADJ_SL:

PROCEDURE REL_Y_X_COORDS ACCEPTS X AND Y SCREEN COORDINATES AND CONVERTS ADDRESS IN MEMORY CORRESPONDING TO THE X, Y POSITION

PROC REL_Y_X_COORDS:

POP [RET_X VA] ;SAVE RETURN ADDRESS
POP BX ;GET Y POSITION
TEST BX, 800CH ;TEST FOR Y < 0
JNZ TST_Y_NFR
TEST BX, 0 ;TEST FOR Y > 376
JZ TST_Y_NFR
TST_Y_NFR:
MOV AX, 376
SUB AX, BX
JNS Y_IN_END

Y_IN_END:
MOV BX, 376  ;PUT SCALE FACTOR IN AX
MOV AX, 60H  ;AND SCALE Y BASED ON LINE NUMBER
POP BX
POP AX
TEST BX, 8000H  ;TEST FOR X < 0
JNS TST_XUPR
TST_XUPR:
MOV BX, 3
SUB DX, BX  ;PUT SCALE FACTOR IN AX
X_IN_END:
CLD  ;AND DETERMINE BYTE X OFFSET
SHR BDX, CL  ;BY DIVIDING X POSITION BY 3
ADD BX, AX  ;ADD X OFFSET TO GET BYTE ADDRESS
PUSH AX  ;RETURN RELATIVE BYTE ADDRESS
PUSH [RET_RVA]  ;RESTORE RETURN ADDRESS
RET
END PROC_REL_ADDR;

LINE DRAWING PROCEDURE WITH COLOR MIXING FOR CROSSING LINES

PROC DRAW_LINE:
POP [RET_DDL]  ;SAVE RETURN ADDRESS
MOV [COLOR_MIX], 1  ;ENABLE COLOR MIXING
PUSH DRAW_LINE  ;GO DRAW THE LINE
PUSH [RET_DDL]  ;RESTORE RETURN ADDRESS
RET
END PROC_DRAW_LINE;

LINE DRAWING PROCEDURE WITHOUT COLOR MIXING FOR CROSSING LINES

PROC DRAW_LINE:
POP [RET_DDL]  ;SAVE RETURN ADDRESS
MOV [COLOR_MIX], 0  ;DISABLE COLOR MIXING
CALL DRAW_LINE  ;GO DRAW THE LINE
PUSH [RET_DDL]  ;RESTORE RETURN ADDRESS
RET
END PROC_DRAW_LINE;

PROCEDURE DRAW_LINE PROVIDES PRIMITIVE LINE DRAWING CAPABILITIES
THE SUBROUTINE ACCEPTS AS INPUT A PAIR OF ENDPOINTS IDENTIFIED BY
(X, Y) COORDINATES

PROC DRAW_LINE:
MOV BX, [BP+3]  ;GET X START FROM STACK
MOV BX, [BP+6]  ;GET Y START FROM STACK
MOV BX, [BP+9]  ;GET X END FROM STACK
MOV BX, [BP+12]  ;GET Y END FROM STACK
TEST ORDERING OF Y COORDINATES. THE ALGORITHM EXPECTS TO FIND THE Y
COORDINATE IN INCREASING ORDER OR 0.
SUB AX, [Y_END]  ;JUMP IF Y ORDERING OK
JNS STO_DY
MOV AX, [X_START]  ;ELSE SWAP START AND END COORD.
MOV AX, [X_END]  ;SO Y START VALUE IS LTE Y END
MOV AX, [Y_START]  ;VALUE. THEN CALCULATE DELTA Y.
MOV AX, [Y_END]
SUB AX, [Y_START]
STO_DY:
MOV [DELTA_Y], AX
AN
;CALCULATE DELTA X

B AN:

,.,., ::o':

ELt., 

AX

IY

N

A,

st 1

RD .

"?P-SH [:.PsT

A!

A ,EPLOT FIRST POINT

E

1A

SL

;TEST FOR NEGATIVE SLOPE

TST_DY:

JNZ 2 OR_3

;TEST FOR DELTA Y = 0

CASE_1:

JNS CASE_2

;DRAW LINE FOR SPECIAL CASE WHERE

DX > 0 AND DY = 0

C_2 OR_3:

JNS CASE_3

;DX > 0 AND DY > 0

CASE_2:

MOV AX, [DELTA_X]

MOV [END_CNT], AX

;DX >= DY

MOV BX, [DELTA_Y]

ADD BX, 1

;SMOOTHING

JNZ C_CASE2

;CORR = DX/(DY+1)

C_CASE2:

MOV AX, [DELTA_X]

MOV [L_ERROR], AX

;INIT ERROR VAL. WITH CORR. FACT.

DR_C2:

MOV AX, [END_CNT]

;TEST INCREMENT COUNTER

SUB AX, [INCCTR]

JNS DR_C2C

EXIT_DL

DR_C2C:

TEST [L_ERROR], 8000H

;TEST IF ERROR < 0

CASE_2C:

ADD [X_POS], 1

;ERROR < 0

ADD [Y_POS], 1

CALL LINE_SEG

ADD AX, [DELTA_Y]

ADD AX, [DELTA_X]

ADD [INCCTR], 1

JNS DR_C2

CASE_2C:

ADD [X_POS], 1

;ERROR >= 0

ADD [Y_POS], 1

CALL LINE_SEG

ADD AX, [DELTA_Y]

ADD AX, [DELTA_X]

ADD AX, [DELTA_X]

ADD AX, [DELTA_Y]

ADD AX, [INCCTR], 1

JNS DR_C2

CASE_3:

MOV AX, [DELTA_Y]

MOV [DELTA_X], AX

;DY > DX
; ADD IN CORRECTION FACTOR FOR SMOOTHING
ADD BX, 1 ; CORR = DY/(DX+1)
C_CASE3:
MOV AX, [DELTA_Y]
MOV BX, 0
MOV [L_ERROR], AX ; INIT ERROR VAL. WITH CORR. FACT.
DR_C3:
MOV AX, [END_CNT] ; TEST INCREMENT COUNTER
SUB AX, [INC_CTR]
JNS DR_C3C
JMP EXIT_DL
DR_C3C:
TEST AX, [DELTA_Y], 8000H ; TEST FOR ERROR < 0
ADD [Y_POS], 1 ; ERROR < 0
PUSH [X_POS]
PUSH [Y_POS]
CALL LINE_SEG
MOV AX, [DELTA_X]
ADD AX, [DELTA_Y]
ADD AX, [INC_CTR], 1
JMP OR_C3
CASE_3C:
ADD [Y_POS], 1 ; ERROR >= 0
PUSH [X_POS]
PUSH [Y_POS]
CALL LINE_SEG
MOV AX, [DELTA_X]
ADD AX, [DELTA_Y]
ADD AX, [INC_CTR], 1
JMP OR_3
CASE_5:
ADD AX, [DELTA_X]
SUB [X_POS], 1
CALL LINE_SEG
SUB [DELTA_X], 1
JMP EXIT_DL
C_5_OR_6:
MOV AX, 0 ; CASE 3 OR 4. DX < 0 & DY > 0
SUB AX, [DELTA_X]
SUB AX, [DELTA_Y]
JNS CASE_5
JMP CASE_6
CASE_6:
MOV AX, 0 ; DY <= ABS(DX)
MOV BX, [DELTA_Y] ; ADD IN CORRECTION FACTOR FOR SMOOTHING
ADD BX, 1 ; CORR = DX/(DY+1)
C_CASES:
MOV AX, [DELTA_X]
SUB AX, [DELTA_X]
ADD AX, [INC_CTR]
JMP OR_C2
DR_C2:
MOV AX, [INC_CTR] ; TEST INCREMENT COUNTER
SUB AX, [INC_CTR]
JNS DR_C5C
JMP EXIT_DL

DR_C5C:
TEST [L_ERROR], 8000H ;TEST IF ERROR < 0
PUSH [X_ERROR] ;ERROR < 0
PUSH [X_ERROR]
CALL ADD
MOV AX, [L_ERROR]
MOV AX, [DELTA_X] ;ERROR >= 0
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
ADD
A
END PROC DRAW_LINE;

;PROCEDURE LINE_SEG ACCEPTS AS INPUT AN X AND Y POSITION AND CONVERTS THE POSITION INTO DISPLAYABLE PIXEL DATA

PROC LINE_SEG:
    MOV [SEGMENT_E], ES ;SAVE SEG E STATUS
    MOV ES, [COL_PL_ADDR] ;ENABLE SELECTED COLOR
    POP [RET-IS] ;SAVE RETURN ADDRESS
    CALL ADJ_3L ;CONVERT Y POS TO DISPLAYABLE S.L.
    CALL REL_VID_ADDR ;USE X & Y TO CALL REL. BYTE ADDR.
    POP BX ;EX REGISTER HOLDS REL. BYTE ADDR.
    MOV AX, [X_POS] ;MOV X POSITION TO AX REGISTER AND
    AND AX, 1 ;CALCULATE X BIT POSITION WITHIN A
    MOV BX, 00H ;BYTE FOR DISPLAY PURPOSES
    MOV DL, 80H
    JMP OUT_LINE

XBIT1:   SUB DL, BX
            MOV DL, 0
            JMP OUT_LINE

XBIT2:   SUB DL, BX
            MOV DL, 40H
            JMP OUT_LINE

XBIT3:   SUB DL, BX
            MOV DL, 20H
            JMP OUT_LINE

XBIT4:   SUB DL, BX
            MOV DL, 10H
            JMP OUT_LINE

XBIT5:   SUB DL, BX
            MOV DL, 6
            JMP OUT_LINE

XBIT6:   SUB DL, BX
            MOV DL, 4
            JMP OUT_LINE

XBIT7:   SUB DL, BX
            MOV DL, 2
            JMP OUT_LINE

OUT_LINE: MOV DL, 0
            JNC OUT_LINE
            MOV DL, 3FH
            CALL APPLE
            MOV AX, [COLORwayne-1] ;TEST IF MIXING ENABLED
            XOR AX
            MOV AX, 0000H ;CLEAR SELECTED LOCATION IN ALL
            MOV [COLORwayne-0], AX
            MOV AX, 00FFH ;COLOR PLANES IF NO MIXING
            MOV [COLORwayne-0], AX
            MOV AX, [COLORwayne-1] ;SELECT B SEGMENT FOR ADDRESSING OF
            MOV [COLORwayne-1], AX
            MOV AX, 00FFH ;BIT 7 OUTPUT BIT
MOV AX, [COL_PL2]
MOV ES, AX
SEG ES VOD
OR [BX], DL
MOV AX, [COL_PL3]
MOV ES, AX
SEG ES VOD
PUSH [RET_LS] ; RESTORE RETURN ADDRESS
MOV ES, [SEGMENT_E] ; RESTORE SEG E
RET

END PROC LINE SEG;
****************************************************************************************************
; PROCEDURE CIRCLE INPUTS X & Y COORDINATES OF THE CIRCLE CENTER
; AND A RADIUS VALUE. IT THEN DRAWS A CIRCLE.
PROC CIRCLE;

MOV BP, SP
MOV AX, [BP+6] ; GET X CENTER COORDINATE OFF STACK
MOV AX, [BP+4] ; GET Y CENTER COORDINATE OFF STACK
MOV DX, [BP+2] ; GET RADIUS VALUE AND SET
MOV [X_REL], 0 ; INITIAL VALUES FOR X AND Y COORD.
MOV CL, 1
SHL DX, CL
MOV AX, 3
SUB AX, DX
MOV [P_VAL], AX

CIRC_LP:
MOV AX, [Y_REL] ; TEST IF FINISHED DRAWING CIRCLE
SUB AX, [X_REL]
JNS DR_CIRC
POP AX
POP BX
POP BX
POP BX
PUSH AX

; POINTS ON CIRCLE ARE PLOTTED IN GROUPS OF 8 WITH THE Y COORDINATE
; SCALED BY 1/2 TO COMPENSATE FOR X:Y RATIO IN THE DISPLAY
DR_CIRC:
MOV AX, [X_CTR] ; CALCULATE POINT 1
MOV DX, [X_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
MOV AX, [Y_REL]
MOV CL, 1
SHR AX, CL
ADD AX, [Y_CTR]
CALL ADJ_SL
POP [Y_POS]
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CIR_PIXEL ; PLOT (X_CENTER+X, Y_CENTER+Y)
MOV AX, [X_CTR] ; CALCULATE POINT 2
MOV DX, [X_REL]
SUB AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [Y_POS]
CALL CIR_PIXEL ; PLOT (X_CENTER-X, Y_CENTER+Y)
MOV BX, [Y_REL]

; PROCEDURE CIRCLE INPUTS X & Y COORDINATES OF THE CIRCLE CENTER
; AND A RADIUS VALUE. IT THEN DRAWS A CIRCLE.
PROC CIRCLE;

MOV BP, SP
MOV AX, [BP+6] ; GET X CENTER COORDINATE OFF STACK
MOV AX, [BP+4] ; GET Y CENTER COORDINATE OFF STACK
MOV DX, [BP+2] ; GET RADIUS VALUE AND SET
MOV [X_REL], 0 ; INITIAL VALUES FOR X AND Y COORD.
MOV CL, 1
SHL DX, CL
MOV AX, 3
SUB AX, DX
MOV [P_VAL], AX

CIRC_LP:
MOV AX, [Y_REL] ; TEST IF FINISHED DRAWING CIRCLE
SUB AX, [X_REL]
JNS DR_CIRC
POP AX
POP BX
POP BX
POP BX
PUSH AX

; POINTS ON CIRCLE ARE PLOTTED IN GROUPS OF 8 WITH THE Y COORDINATE
; SCALED BY 1/2 TO COMPENSATE FOR X:Y RATIO IN THE DISPLAY
DR_CIRC:
MOV AX, [X_CTR] ; CALCULATE POINT 1
MOV DX, [X_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
MOV AX, [Y_REL]
MOV CL, 1
SHR AX, CL
ADD AX, [Y_CTR]
CALL ADJ_SL
POP [Y_POS]
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CIR_PIXEL ; PLOT (X_CENTER+X, Y_CENTER+Y)
MOV AX, [X_CTR] ; CALCULATE POINT 2
MOV DX, [X_REL]
SUB AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [Y_POS]
CALL CIR_PIXEL ; PLOT (X_CENTER-X, Y_CENTER+Y)
MOV BX, [Y_REL]

; POINTS ON CIRCLE ARE PLOTTED IN GROUPS OF 8 WITH THE Y COORDINATE
; SCALED BY 1/2 TO COMPENSATE FOR X:Y RATIO IN THE DISPLAY
DR_CIRC:
MOV AX, [X_CTR] ; CALCULATE POINT 1
MOV DX, [X_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
MOV AX, [Y_REL]
MOV CL, 1
SHR AX, CL
ADD AX, [Y_CTR]
CALL ADJ_SL
POP [Y_POS]
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CIR_PIXEL ; PLOT (X_CENTER+X, Y_CENTER+Y)
MOV AX, [X_CTR] ; CALCULATE POINT 2
MOV DX, [X_REL]
SUB AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [Y_POS]
CALL CIR_PIXEL ; PLOT (X_CENTER-X, Y_CENTER+Y)
MOV BX, [Y_REL]
MOV CL, 1
SHR BX, CL
MOV AX, [Y_CTR]
SUB AX, BX
PUSH AX
CALL ADJ SL
POP [Y_POS]
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [Y_POS]
CALL CTR_PIXEL ;PLOT (X_CENTER-X, Y_CENTER-Y)
MOV AX, [X_CTR] ;CALCULATE POINT 4
MOV DX, [X_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CTR_PIXEL ;PLOT (X.CENTER+X, Y_CENTER-Y)
MOV AX, [X_CTR] ;CALCULATE POINT 5
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
MOV AX, [X_REL]
MOV CL, 1
SHR AX, CL
ADD AX, [Y_CTR]
PUSH AX
CALL ADJ SL
POP [Y_POS]
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CTR_PIXEL ;PLOT (X_CENTER+Y, Y_CENTER+X)
MOV AX, [X_CTR] ;CALCULATE POINT 6
MOV DX, [Y_REL]
SUB AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CTR_PIXEL ;PLOT (X.CENTER-Y, Y_CENTER+X)
MOV AX, [X_CTR] ;CALCULATE POINT 7
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CTR_PIXEL ;PLOT (X-CENTER-Y, YCENTER+X)
MOV AX, [X_CTR] ;CALCULATE POINT 8
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [Y_POS]
CALL CTR_PIXEL ;PLOT (X_CENTER+Y, Y_CENTER+X)
MOV AX, [X_CTR] ;CALCULATE POINT 9
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CTR_PIXEL ;PLOT (X_CENTER+Y, Y_CENTER+X)
MOV AX, [X_CTR] ;CALCULATE POINT 10
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [Y_POS]
CALL CTR_PIXEL ;PLOT (X.CENTER-Y, Y_CENTER-X)
MOV AX, [X_CTR] ;CALCULATE POINT 11
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CTR_PIXEL ;PLOT (X_CENTER-Y, Y_CENTER-X)
MOV AX, [X_CTR] ;CALCULATE POINT 12
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CTR_PIXEL ;PLOT (X_CENTER-Y, Y_CENTER-X)
MOV AX, [X_CTR] ;CALCULATE POINT 13
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CTR_PIXEL ;PLOT (X.CENTER-Y, Y_CENTER-X)
MOV AX, [X_CTR] ;CALCULATE POINT 14
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CTR_PIXEL ;PLOT (X_CENTER-Y, Y_CENTER-X)
MOV AX, [X_CTR] ;CALCULATE POINT 15
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CTR_PIXEL ;PLOT (X.CENTER-Y, Y_CENTER-X)
MOV AX, [X_CTR] ;CALCULATE POINT 16
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CTR_PIXEL ;PLOT (X_CENTER-Y, Y_CENTER-X)
MOV AX, [X_CTR] ;CALCULATE POINT 17
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CTR_PIXEL ;PLOT (X.CENTER-Y, Y_CENTER-X)
MOV AX, [X_CTR] ;CALCULATE POINT 18
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CTR_PIXEL ;PLOT (X_CENTER-Y, Y_CENTER-X)
MOV AX, [X_CTR] ;CALCULATE POINT 19
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CTR_PIXEL ;PLOT (X.CENTER-Y, Y_CENTER-X)
MOV AX, [X_CTR] ;CALCULATE POINT 20
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_POS]
CALL REL_VID_ADDR
PUSH [X_POS]
CALL CTR_PIXEL ;PLOT (X_CENTER-Y, Y_CENTER-X)
MOV AX, [X_CTR] ;CALCULATE POINT 21
MOV DX, [Y_REL]
ADD AX, DX
MOV [X_POS], AX
PUSH AX
PUSH [Y_VAL], 8000H ;TEST P FOR NEGATIVE VALUE
PROCEDURE CIR_PIXEL ACCEPTS AS INPUT A RELATIVE BYTE ADDRESS AND AN INDEX COORDINATE POSITION AND SETS THE CORRESPONDING PIXEL

PROC CIR_PIXEL:

    [SEGMENT_E]. ES   ;SAVE SEG E STATUS
    [RET_ADDR]. DS    ;SAVE RETURN ADDRESS
    DX, BX            ;GET X POSITION
    DL                ;GET RELATIVE BYTE ADDRESS

CPBIT1:    SUB     AL, 80H
            MOV     CX, 7FH
            SHR     AX, 1
            CPIX_ON

CPBIT2:    SUB     AL, 40H
            MOV     CX, 0FH
            SHR     AX, 1
            CPIX_ON

CPBIT3:    SUB     AL, 20H
            MOV     CX, 07H
            SHR     AX, 1
            CPIX_ON

CPBIT4:    SUB     AL, 10H
            MOV     CX, 0FH
            SHR     AX, 1
            CPIX_ON

CPBIT5:    SUB     AL, 0AH
            MOV     CX, 0FH
            SHR     AX, 1
            CPIX_ON

CPBIT6:    SUB     AL, 05H
            MOV     CX, 0FH
            SHR     AX, 1
            CPIX_ON

CPBIT7:    SUB     AL, 0FH
            MOV     CX, 0FH
            SHR     AX, 1
            CPIX_ON

CPIN_ON:   CMP     AL, 0FH
            JNZ     CPBIT1
            MOV     AL, 0FH
            CPIX_ON

... CLEAR ALL COLOR PLANES AT SELECTED PIXEL LOCATION BEFORE SETTING PIXEL TO DESIRED COLOR

ADD     DL, 3000H

END PROC CIR_PIXEL
MOV ES, DX
SEG ES
AND [BX], CL
MOV DX, OEOOH
MOV ES, DX
SEG ES
AND [BX], CL
MOV DX, [COL_PL1] ; SET PIXEL TO DESIRED COLOR
MOV ES, DX
SEG ES
OR [BX], AL
MOV DX, [COL_PL2]
MOV ES, DX
SEG ES
OR [BX], AL
MOV DX, [COL_PL3]
MOV ES, DX
SEG ES
OR [BX], AL
PUSH RET_CPE
MOV ES, [SEGMENT_E] ; RESTORE SEG E

END PROC CRT_PIXEL;

;**********************************************************************
;**********************************************************************
; PROCEDURE SET_PIXEL ACCEPTS AS INPUT AN (X,Y) COORDINATE AND SETS THE
; CORRESPONDING PIXEL TO THE SYSTEM COLOR
;**********************************************************************
;**********************************************************************

PROC SET_PIXEL;

MOV BP, SP
MOV [SEGMENT_E], ES ; SAVE ES STATUS
POP [RET_SRE] ; SAVE RETURN ADDRESS
MOV AX, [BP+4] ; GET X POSITION
MOV [X_POS], AX
CALL ADOSL
CALL REL_VID_ADDR
POP BX
MOV DX, [X_POS] ; GET RELATIVE BYTE ADDRESS
AND DX, 7 ; MAKE A BIT MASK

PBIT1:
SUB DL, 1
JNZ PBIT2
MOV AL, 80H
MOV CL, 7FH
JMP PIX_ON

PBIT2:
SUB DL, 1
JNZ PBIT3
MOV AL, 40H
MOV CL, OBFH
JMP PIX_ON

PBIT3:
SUB DL, 1
JNZ PBIT4
MOV AL, 20H
MOV CL, ODFH
JMP PIX_ON

PBIT4:
SUB DL, 1
JNZ PBIT5
MOV AL, 10H
MOV CL, OEFH
JMP PIX_ON

PBIT5:
SUB DL, 1
JNZ PBIT6
MOV AL, 3
MOV CL, O7FH
JMP PIX_ON

PBIT6:
SUB DL, 1
JNZ PBIT7
MOV AL, 4
MOV CL, OFBH
JMP PIX_ON

PBIT7:
SUB DL, 1
JNZ PBIT1
MOV AL, 5
MOV CL, OF7H
JMP PIX_ON

67
MOV AL, 2
MOV CL, 0FDH
JMP PIX_ON

PBIT7:
MOV AL, 1
MOV CL, OFDH
JMP PIX_ON

PIX_ON:
MOV DX, 0000H ;PIXEL LOCATION BEFORE SETTING
SEG ES, DX ;PIXEL TO NEW COLOR
AND [BX], CL
MOV DX, 0DO00H
MOV ES, DX
SEG ES, DX
AND [BX], CL
MOV DX, [COL_PL1] ;SET PIXEL TO DESIRED COLOR
MOV ES, DX
SEG ES, DX
OR [BX], AL
MOV DX, [COL_PL2]
MOV ES, DX
SEG ES, DX
OR [BX], AL
MOV DX, [COL_PL3]
MOV ES, DX
SEG ES, DX
OR [BX], AL
PUSH [RET_SPE] ;RESTORE RETURN ADDRESS
MOV ES, [SEGMENT_E] ;RESTORE SEGMENT E
RET

END PROC SET_PIXEL;

;--------------------------------------------------------------------------------
;PROCEDURE RESET_PIXEL ACCEPTS AS INPUT AN (X,Y) COORDINATE AND RESETS THE CORRESPONDING PIXEL BY WRITING "0" TO ALL COLOR PLANES
PROC RESET_PIXEL;
MOV BP, SP
MOV [SEGMENT_E], ES ;SAVE ES STATUS
POP [RET_SPE] ;SAVE RETURN ADDRESS
MOV AX, [BP+4] ;GET X POSITION
MOV [X_POS], AX
CALL ADJUST
CALL REL_VIDEO_ADDR
POP BX
MOV DX, [X_POS] ;GET RELATIVE BYTE ADDRESS
AND DX, 7 ;MAKE A BIT MASK

RBIT1:
JMP PIX_OFF
SUB DL, 1
JNZ RBIT2

RBIT2:
MOV AL, 0BFH
JMP PIX_OFF
SUB DL, 1
JNZ RBIT3

RBIT3:
MOV AL, 0DFH
JMP PIX_OFF
SUB DL, 1
JNZ RBIT4

RBIT4:
MOV AL, 0F7H
JMP PIX_OFF
SUB DL, 1
JNZ RBIT5

RBIT5:
MOV AL, 0FH
JMP PIX_OFF
SUB DL, 1
JNZ RBIT6

RBIT6:
```
RBIT6:  MOV  AL, 0FH
        JMP  PIX_OFF

JMP  DL, 1
JNZ  RBIT7

RBIT7:  MOV  AL, 0FH
        JMP  PIX_OFF

MOV  AL, 0FH
MOV  AL, 0FH
MOV  PL, 0"'

PIX_OFF:

MOV  DX, 0C000H
MOV  ES, DX
SEG
AND  [BX], AL
MOV  DX, 0D000H
MOV  ES, DX
SEG
AND  [BX], AL
MOV  DX, 0E000H
MOV  ES, DX
SEG
AND  [BX], AL
PUSH  [RET_RPE]   ;RESTORE RETURN ADDRESS
MOV  ES, [SEGMENT_E] ;RESTORE SEGMENT E
RET

E: ;PROC RESET_PIXEL:

;****************************************************************************
;PROCEDURE INQUIRE_COLOR IS USED TO INTERFACE WITH HIGHER LEVEL ADA
;PROGRAMS IN ORDER TO HANDLE I/O PARAMETER PASSING
PROC INQUIRE_COLOR:

MOV  BP, SP
PUSH  [BP+6]
PUSH  [BP+4]
CALL  INQ_COLOR
POP  AX   ;COLOR VAL RETURNED
POP  BP
MOV  [BP+2], AL
MOV  [DX], AL
POP  AX
MOV  [BP+4], AL
MOV  [BP+6], AX
RET

END PROC INQUIRE_COLOR;

;****************************************************************************
;PROCEDURE INQ_COLOR ACCEPTS X & Y COORDINATES AS AN INPUT AND
;RETURNS THE PIXEL COLOR CODE OF THAT LOCATION
PROC INQ_COLOR:

MOV  AX, [BP+4]
MOV  [X_POS], AX
POP  [RET_RIC] ;SAVE RETURN ADDRESS
CALL  ADJ_SLP
CALL  REL_VID_ADDR
POP  BX
PUSH  [X_POS]
CALL  X_MASK
POP  CX
PUSH  ES   ;SAVE ES STATUS
MOV  AX, 0C000H
MOV  [ES], AX
SEG
AND  DL, [BX]
JNZ  B1

B1:
AND  DL, 000H
JMP  ES, AX
SEG
AND  DL, [BX]
JNZ  B2
JMP  B_NO_R
```
B2: MOV AX, OEO00H ; RED IS SET, TEST GREEN
    MOV ES, AX
    MOV [BX], CL
    MOV DL, [BX]
    AND DL, CL
    JMP B3

B3: MOV AX, 0
    JMP EXIT IC ; COLOR IS WHITE

BR_NO_G: MOV AX, 0
    JMP EXIT IC ; COLOR IS MAGENTA

B_NO_R: MOV AX, OEO00H ; BLUE IS SET, RED IS NOT, TEST GREEN
    MOV ES, AX
    MOV DL, [BX]
    AND DL, CL
    JMP B4

B4: MOV AX, 1
    JMP EXIT IC ; COLOR IS CYAN

B_NO_RG: MOV AX, 3
    JMP EXIT IC ; COLOR IS BLUE

B5: MOV AX, OEO00H
    MOV ES, AX
    MOV DL, [BX]
    AND DL, CL
    JMP B6

B6: MOV AX, 4
    JMP EXIT IC ; COLOR IS YELLOW

R_NO_BG: MOV AX, 6
    JMP EXIT IC ; COLOR IS RED

NO_BR: MOV AX, OEO00H
    MOV ES, AX
    MOV DL, [BX]
    AND DL, CL
    JMP B7

B7: MOV AX, 5
    JMP EXIT IC ; COLOR IS GREEN

NO_RGB: MOV AX, 7
    JMP EXIT IC ; COLOR IS BLACK

END_PROC COLOR

;---------------------------------------------------------------------------------------------
; PROCEDURE X MASK ACCEPTS AN X COORDINATE POSITION AS INPUT AND RETURNS A BIT MASK BASED ON THE X COORDINATE
PROC X_MASK:
    POP DX ; SAVE RETURN ADDRESS
    POP AX ; GET X POSITION
    MOV AX, 7
    JNZ NBIT1

NBIT1:
    MOV AX, 80H
    JMP EXIT_XM!

EXIT_XM:
    MOV AX, 1
    JNZ NBIT2

NBIT2:
    MOV AX, 40H
    PUSH AX
    PUSH [RET IC]
    RET
MBIT2:  JMP EXIT_XM
        SUB AX, I
        JNZ MBIT3
        MOV AX, _2H
        JMP EXIT_XM

MBIT3:  SUB AX, I
        JNZ MBIT4
        MOV AX, _1CH
        JMP EXIT_XM

MBIT4:  SUB AX, I
        JNZ MBIT5
        MOV AX, _8H
        JMP EXIT_XM

MBIT5:  SUB AX, I
        JNZ MBIT6
        MOV AX, _4H
        JMP EXIT_XM

MBIT6:  SUB AX, I
        JNZ MBIT7
        MOV AX, _2H
        JMP EXIT_XM

MBIT7:  MOV AX, I
        JNZ MBIT8
        MOV AX, AX
        JMP EXIT_XM

EXIT_XM: PUSH AX
         ;RETURN X MASK
        PUSH DX
         ;RESTORE RETURN ADDRESS
        RET

END PROC X_MASK;

;PROCEDURE BOUNDARY_FILL ACCEPTS AS INPUT X AND Y COORDINATES, A FILL COLOR, AND A BOUNDARY COLOR AND PERFORMS A SCREEN FILL WITH THE FILL COLOR UP TO THE SPECIFIED BOUNDARY
PROC BOUNDARY_FILL;

POP [RET_FILL]
POP [B_COLOR]
MOV [SEGMENT_FE] ES
MOV AX, [COL_PL1]
MOV [SAV_COL1_STAT], AX
MOV AX, [COL_PL2]
MOV [SAV_COL2_STAT], AX
MOV AX, [COL_PL3]
MOV [SAV_COL3_STAT], AX
CALL B_FILL
POP [RET_FILL]
MOV ES, [SEGMENT_FE]
MOV AX, [SAV_COL1_STAT]
MOV [COL_PL1], AX
MOV AX, [SAV_COL2_STAT]
MOV [COL_PL2], AX
MOV AX, [SAV_COL3_STAT]
MOV [COL_PL3], AX
RET

END PROC BOUNDARY_FILL;

;PROCEDURE R_FILL IS THE RECURSIVE PART OF THE BOUNDARY FILL PROCEDURE. IT IS USED TO AVOID UNNECESSARY PASSING OF THE COLOR PARAMETERS WHICH DO NOT CHANGE ON SUCCESSIVE RECURSIVE CALLS.
PROC B_FILL;

POP DX
        ;SAVE RETURN ADDRESS
MOV CX, [X_POS]
MOV BX, [Y_POS]
POP [Y_POS]
        ;GET NEXT Y POSITION OFF STACK
POP [X_POS]
        ;GET NEXT X POSITION OFF STACK
PUSH DX
PUSH CX
PUSH BX
PUSH [X_POS]
        ;SAVE PRESENT X POSITION
PUSH [Y_POS]
        ;SAVE PRESENT Y POSITION

61
CALL INO_COLOR
POP AX
MOV AX, [P_COLOR], AX ;COMPARE COLOR AT PRESENT LOCATION
CMP AX, [B_COLOR] ;WITH BOUNDARY COLOR
JNE TEST FC
JIP EXIT_F
TEST.FC:
MOV AX, [P_COLOR];COMPILE COLOR AT PRESENT LOCATION
CMP AX, [F_COLOR];WITH FILL COLOR
JNZ F_PIX
JNE EXIT_F
F_PIX:
PUSH [X_POS]
PUSH [Y_POS]
cALL ADJ SL
CALL REL VID ADDR
PUSH [X_POS]
cALL COLOR
PUSH [Y_POS]
cALL COLOR
PUSH AX
PUSH [X_POS]
cALL CIR PIXEL ;SET LOCATION (X,Y) TO FILL COLOR
MOV AX, [X_POS]
ADD AX, 1
PUSH AX
PUSH [Y_POS]
cALL B_FILL ;CALL B_FILL PASSING (X+1,Y)
MOV AX, [X_POS]
SUB AX, 1
PUSH AX
PUSH [Y_POS]
cALL B_FILL ;CALL B_FILL PASSING (X-1,Y)
ADD AX, [X_POS]
ADD AX, 1
PUSH AX
PUSH [Y_POS]
cALL B_FILL ;CALL B_FILL PASSING (X,Y+1)
MOV AX, [Y_POS]
SUB AX, 1
PUSH AX
PUSH [X_POS]
cALL B_FILL ;CALL B_FILL PASSING (X,Y-1)
MOV AX, [Y_POS]
PUSH AX
PUSH [X_POS]
EXIT_F:
PUSH [Y_POS]
;RESTORE POSITION (X,Y)
PUSH [X_POS]
RET

END PROC B_FILL;

PROC AREA_FILL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

PROC AREA_FILL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

END
PROCEDURE A_FILL IS THE RECURSIVE PART OF THE AREA FILL PROCEDURE.

IT IS USED TO AVOID UNNECESSARY PASSING OF THE COLOR PARAMETERS WHICH
DO NOT CHANGE ON SUCCESSIVE RECURSIVE CALLS.

PROCEDURE A_FILL:

```
PROCEDURE A-FILL
IS THE RECURSIVE PART OF THE AREA FILL PROCEDURE.

IT IS USED TO AVOID UNNECESSARY PASSING OF THE COLOR PARAMETERS WHICH
DO NOT CHANGE ON SUCCESSIVE RECURSIVE CALLS.

PROCEDURE A_FILL:

POP DX ;SAVE RETURN ADDRESS
MOV CX, [X_POS]
MOV BX, [Y_POS]
POP [X_POS]
POP [Y_POS]
PUSH DX
PUSH CX
PUSH [X_POS]
PUSH [Y_POS]
CALL TIO_COLOR
POP AX
MOV [P_COLOR], AX ;COMPARE COLOR AT PRESENT LOCATION
CMP AX, [B_COLOR] ;WITH AREA COLOR
JNZ EXIT_AF
PUSH [X_POS]
PUSH [Y_POS]
CALL ADJ_SL
CALL REL_VID_ADDR
PUSH [X_POS]
PUSH [P_COLOR]
CALL COLOR
CALL CIR_PIXEL ;SET LOCATION (X,Y) TO FILL COLOR
MOV AX, [X_POS]
ADD AX, 1
PUSH AX
PUSH [Y_POS]
CALL A_FILL
MOV AX, [X_POS]
SUB AX, 1
PUSH AX
PUSH [Y_POS]
CALL A_FILL ;CALL A_FILL PASSING (X+1,Y)
MOV AX, [X_POS]
ADD AX, 1
PUSH AX
PUSH [Y_POS]
CALL A_FILL ;CALL A_FILL PASSING (X-1,Y)
MOV AX, [X_POS]
PUSH AX
CALL A_FILL ;CALL A_FILL PASSING (X,Y+1)
MOV AX, [X_POS]
PUSH AX
CALL A_FILL ;CALL A_FILL PASSING (X,Y-1)
EXIT_AF:
PUSH [Y_POS]
PUSH [X_POS]
RET
```

END PROC A_FILL;

Pragma directives
LIST OF REFERENCES


BIBLIOGRAPHY


# Initial Distribution List

<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
<th>Name and Address</th>
</tr>
</thead>
</table>
| 1.  | 2      | Defense Technical Information Center  
|     |        | Cameron Station  
|     |        | Alexandria, Virginia 22302-6145 |
| 2.  | 2      | Library Code 0142  
|     |        | Naval Postgraduate School  
|     |        | Monterey, California 93943-5002 |
| 3.  | 1      | Department Chairman, Code 52  
|     |        | Department of Computer Science  
|     |        | Naval Postgraduate School  
|     |        | Monterey, California 93943 |
| 4.  | 3      | Dr. Uno R. Kodres, Code 52Kr  
|     |        | Department of Computer Science  
|     |        | Naval Postgraduate School  
|     |        | Monterey, California 93943 |
| 5.  | 1      | Dr. Michael J. Zyda, Code 52Zk  
|     |        | Department of Computer Science  
|     |        | Naval Postgraduate School  
|     |        | Monterey, California 93943 |
| 6.  | 1      | Daniel Green, Code 20F  
|     |        | Naval Surface Weapons Center  
|     |        | Dahlgren, Virginia 22449 |
| 7.  | 1      | CAPT. J. Hood, USN  
|     |        | PMS 400B5  
|     |        | Naval Sea Systems Command  
|     |        | Washington, D.C. 20362 |
| 8.  | 1      | RCA AEGIS Repository  
|     |        | RCA Corporation  
|     |        | Government Systems Division  
|     |        | Mail Shop 127-327  
|     |        | Moorestown, New Jersey 08057 |
| 9.  | 1      | Library (Code E33-05)  
|     |        | Naval Surface Weapons Center  
|     |        | Dahlgren, Virginia 22449 |
| 10. | 1      | Dr. M. J. Graia  
|     |        | Applied Physics Laboratory  
|     |        | John Hopkins Road  
|     |        | Laurel, Maryland 20707 |
| 11. | 1      | Dana Small, Code 8242  
|     |        | NOS  
|     |        | San Diego, California 92152 |
| 12. | 2      | Thomas R. Brown, Jr.  
|     |        | 340 Essex Dr.  
|     |        | Lexington Park, Maryland 20653 |
END

3-87

DTI C