LINGRAF: THE DESIGN OF AN INTERACTIVE TEACHING AID FOR LINEAR PROGRAMMING

A Thesis Presented to
The Faculty of the College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Stanley G. Revis
June, 1985

OHIO UNIVERSITY LIBRARY
CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.   PROBLEM DEFINITION</td>
<td>1</td>
</tr>
<tr>
<td>II.  EVALUATION AND DESIGN DEFINITION</td>
<td>3</td>
</tr>
<tr>
<td>Background</td>
<td>3</td>
</tr>
<tr>
<td>Applications</td>
<td>4</td>
</tr>
<tr>
<td>Criteria</td>
<td>5</td>
</tr>
<tr>
<td>III. SYSTEM DESCRIPTION</td>
<td>8</td>
</tr>
<tr>
<td>Hardware</td>
<td>8</td>
</tr>
<tr>
<td>Software</td>
<td>12</td>
</tr>
<tr>
<td>User Interface</td>
<td>12</td>
</tr>
<tr>
<td>Shading</td>
<td>18</td>
</tr>
<tr>
<td>Moving the Objective Function Line</td>
<td>24</td>
</tr>
<tr>
<td>IV.  SYSTEM INTEGRATION</td>
<td>28</td>
</tr>
<tr>
<td>Graphics</td>
<td>28</td>
</tr>
<tr>
<td>Memory Allocation</td>
<td>29</td>
</tr>
<tr>
<td>V.   LIMITATIONS AND SUGGESTIONS FOR FURTHER</td>
<td>34</td>
</tr>
<tr>
<td>RESEARCH</td>
<td></td>
</tr>
<tr>
<td>Limitations</td>
<td>34</td>
</tr>
<tr>
<td>Suggestions for Further Research</td>
<td>35</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>37</td>
</tr>
</tbody>
</table>

Appendix

<table>
<thead>
<tr>
<th>Appendix</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. LINGRAF USER'S MANUAL</td>
<td>38</td>
</tr>
<tr>
<td>B. BASIC PROGRAM LISTINGS</td>
<td>42</td>
</tr>
<tr>
<td>C. ASSEMBLY LANGUAGE PROGRAM LISTINGS</td>
<td>64</td>
</tr>
<tr>
<td>D. BASIC FLOW CHARTS</td>
<td>80</td>
</tr>
</tbody>
</table>
Chapter I
PROBLEM DEFINITION

Linear programming is a topic that is taught in several different curricula including, Industrial and Systems Engineering, Quantitative Methods and Mathematics. Linear programming is even being taught at the high school level. Problems with two decision variables may be solved graphically and are often used to introduce students to linear programming. The graphical solution technique is especially important in high school courses, as it helps students to visualize the structure and the process of solving the problems.

In a classroom setting, it is often difficult and time consuming to present the graphical solution to a linear programming problem. The professor must draw the solution on the chalk board, which is not only inaccurate, but also is time consuming. Students may become bored and lose track of the point that the professor is endeavoring to make. A better approach might be the use of transparencies and an overhead projector. This might reduce the time required to solve the problem, but any modification to the problem would probably require separate transparencies. This increases the preparation time required for the lecture and inhibits
interaction between the instructor and the students. Another disadvantage is that the process of constructing the graph can not be shown with transparencies. Thus, a fast, flexible, easy to use system for presenting the graphical solution to a linear programming problem would be beneficial.

Over the past several years, many new applications have been found for computer graphics. This has been due, in part, to the availability of fairly inexpensive microcomputer systems capable of displaying graphics. Many low-cost systems offer high-resolution color graphics. Such systems could be used effectively for displaying graphical solutions to linear programming problems with two decision variables.

This thesis will describe the design of a set of programs, named LINGRAP, that form a system that may be used to solve linear programming problems with two decision variables and up to nine constraints on an APPLE II-series microcomputer. The Apple II is extremely popular and available in schools at all levels.
Chapter II
EVALUATION AND DESIGN DEFINITION

This section will describe the criteria for the system, as well as identification of those who could be expected to use the system.

2.1 BACKGROUND

Although computers are commonly used to solve linear programming problems, very little attention has been paid to using computers to present the graphical solution. A search of several bibliographic databases failed to produce a single reference. The databases that were searched were ERIC, INSPEC, Mathfile and Microcomputer Index. However, two papers by Zimmerman et al., and Pruett and Bhagavan presenting graphical solutions on microcomputers were found, references 12 and 13. In a telephone interview, Dr. Zimmerman stated that their system was originally written for a Radio Shack TRS-80, Model I microcomputer. An updated version, for the IBM-PC, is a menu-driven program that allows the user to enter a linear programming problem and then presents the graphical solution using high-resolution graphics. It does not draw the objective function line. In a telephone interview, Dr. Pruett stated that their paper was based on Dr. Bhagavan's Master's Thesis. It consisted of three parts.
The first part guided the user through an explanation of linear programming. It required little input from the user; just pressing the return key to continue. In the next part, the user was given a problem statement for one of three pre-defined problems and was required to formulate the problem as a linear programming problem. He or she was not allowed to continue until the formulation was correct. After responding correctly, the graphical solution was presented. The user was able to move the constraint lines around on the plot. The third part of the system allowed the user to enter any problem of two variables. The problem was then solved and the graphical solution was presented. The system was written for a Radio Shack TRS-80, Model III microcomputer. It used character graphics, rather than high-resolution graphics, for the plots. The software presented here, LINGRAF, differs in that it plots the objective function and displays the plot in high-resolution graphics. It does not provide a tutorial on linear programming.

2.2 APPLICATIONS

An examination of the topic in almost any introductory linear programming text will reveal that the graphical solution technique is introduced early, usually before any mathematical solution techniques are introduced.

The package described in this thesis could be used in several ways, including:
1. As a visual aid during a lecture on the graphical solution technique.

2. By individual students, as part of an out-of-class assignment.

3. By any person who needs to solve or perform sensitivity analysis on a linear programming problem with two decision variables.

2.3 CRITERIA

A number of criteria were established for the system. They may be classified as general or specific.

The general criteria include:

1. The package must be easy to use, even for a user with no prior computer experience.

2. The programs must execute quickly enough that the user and/or viewers do not become bored and lose interest or lose their train of thought. This is particularly important when the system is being used as a visual aid during a lecture.

3. The system should be designed for a common computer system, in order that it may be transferable.

4. The package should require a minimum of peripheral devices that are not included with the basic computer system.
5. The package should require a minimum of additional software, over that included with the basic computer system.

6. All segments of software package should reside in memory concurrently. If this were not adhered to, overlaying techniques would have to be used. This would slow execution, due to the time the disk would be in use.

The specific requirements include the following:

1. The system should allow a user to solve a linear programming problem with two decision variables.

2. The graphical solution to the problem should be displayed on the screen.

3. The feasible region should be shaded.

4. The constraints should be plotted in a different color than the objective function, and a means of easily identifying each constraint should be provided.

5. The user should be able to change the problem after it has been solved. This includes the ability to add new constraint equations and modify or delete existing constraint equations. Also, the user should be able to change the objective function equation.

6. The user should be able to move the objective function line parallel to itself across the screen. The current value of the objective function, as well as the optimal value should be displayed with the graphical solution.
7. The user should be able to list the constraint equations and the objective function equation on the screen.

8. Finally, the programs should be user-friendly. They should always tell the user what he or she is expected to do and keep the user informed of what is happening. Input errors should be recognized where possible and the user should be told that an error has occurred and what corrective action to take.

The requirements and how each was met will be discussed in the next chapter.
Chapter III  
SYSTEM DESCRIPTION

3.1 HARDWARE

Before any programming could begin, a microcomputer system had to be chosen. Many microcomputers are available which offer the necessary graphics capabilities. Unfortunately, each system is different and programs written for one do not necessarily run on others. This is particularly true of programs involving graphics because of the wide variation in hardware and software capabilities. One system had to be chosen, and the design tailored to it. The Apple II is the system that was chosen. It was chosen for several reasons. It is capable of displaying high resolution graphics in six colors. Its features allow all of the design criteria to be met. Because it is one of the most popular microcomputers, it meets the portability criterion.

The Apple II is a microcomputer system based on the 6502 microprocessor. In its standard form, it runs the BASIC language. Two forms of BASIC are available, Integer BASIC and Applesoft BASIC. Integer BASIC, as its name implies, is only able to perform arithmetic on integer values. For this reason, it is not suitable for use with LINGRAF,
because many calculations require the use of floating-point arithmetic. Applesoft BASIC is a floating-point BASIC, and is thus suitable for use in LINGRAF. The Applesoft interpreter is available in several forms. The versions that reside in memory starting at hexadecimal location D000 are required for this system. Other versions reside in system RAM (random access memory), which reduces the memory available for user programs. The minimum amount of memory required for LINGRAF is 48K (49152) bytes. The Apple II-series computers that are compatible with LINGRAF are:

1. Any Apple IIC.
2. An Apple IIe with one or more disk drives.
3. An Apple II Plus with at least 48K of RAM and one or more disk drives.
4. An Apple II with at least 48K of RAM, one or more disk drives, and either the Applesoft ROM card (an Apple accessory) or Applesoft loaded into an Apple Language Card (an Apple accessory) or functional equivalent.

The only other required peripheral device is a video monitor. A color monitor is recommended, but a monochrome monitor would be adequate. The purpose of the disk is to permanently store the programs that comprise LINGRAF. The disk drive is a very common accessory for Apple II's, and is built into the Apple IIC. The only practical alternative is a cassette tape recorder, but this would be prohibitive
slow due to the size of the programs that comprise LINGGRAF. Thus, the requirement that additional peripheral devices be kept to a minimum has been met.

Applesoft does not provide a means of displaying text characters on the high-resolution graphics display. The ability to display text on the plot of the solution is important to annotate the display. For this reason, a program named the "High Resolution Character Generator" is used to provide this capability. This program is available from Apple Computer as part of the "DOS Toolkit" software package.

The Apple II's high resolution graphics offers a resolution of 280 (horizontal) by 192 (vertical) pixels. A pixel, or picture element, is the smallest point that may be plotted on the screen. Six colors are available, including black and white. Because of the method used to store the picture information, only four colors may be used in this application. They are green, blue, black and white. Alternately, violet and red could have been substituted for blue and green.

The Applesoft BASIC interpreter allows three high-resolution graphics operations. Predefined shapes may be drawn, but they are of little use in this system, as the shape definitions are difficult to create and to change dynamically. The other two operations, plotting points and drawing lines, are of more use. Individual points may be plotted by specifying the coordinates of the point. Lines may be drawn by
specifying the coordinates of the end points of the line. No other operations, such as shading, are included in Applesoft.

The coordinate system used in the high-resolution graphics is not user-definable. The x-coordinates range from 0 to 279 and the y-coordinates from 0 to 191. The origin is at the upper-left corner of the plot area.

Applesoft is an interpretive language, rather than a compiled language. It is inherently slower than compiled languages (Applesoft compilers are available, but are usually expensive). An alternative to the use of Applesoft would be the use of Apple Pascal or Apple FORTRAN. This was rejected because each required the Apple Language System, plus the compiler, and associated software, which are expensive. An alternative that requires no additional hardware or software would be the use of 6502 assembly language in place of Applesoft. This would increase speed, but it would also make the task of programming and maintenance much more difficult, due to the nature of the assembly language. A more practical alternative is the use of both Applesoft and assembly language. This is the approach that was chosen. Each is used where its strengths may be exploited.
3.2 SOFTWARE

3.2.1 User Interface

A very important aspect of the system design is the design of the user interface. This includes how all information is presented to the user and how the user passes information to the system. It also includes error detection and handling.

All options currently available to the user are displayed on the screen. There are two types of displays, graphics and text. The text display is used for entering data and when changes are being made to the problem. The graphics display is used to present the graphical solution to the problem. See LINGRAF Users' Manual, Appendix A.

When the graphics display is in use, little room on the screen is available for explaining available options. Therefore, only a list of which keys may be pressed is displayed. No description of the action of each element in the list is displayed. This is not seriously limiting as there are only three basic options while viewing graphics. They are:

1. Move objective function line (in either direction, in large or small steps).
2. Cause one of the constraints to flash (for identification).
3. Switch display to the main menu.
The keys that were chosen to perform these functions were selected so as to be as self-explanatory as possible. The left and right arrow keys move the objective function in small steps, the greater than and less than keys move it in larger steps, the escape (ESC) key switches to the main menu, and a number key causes the constraint with that number to flash. When a constraint is flashing, pressing any key causes it to stop flashing. A message to that effect replaces the list of valid keys while the constraint is flashing. An example of the graphics display is shown in Figure 1.

Figure 1
LINGGRAF Graphic Display
The text display is used to direct the user to enter all of the data necessary to define the problem. It is also used after the problem has been entered and solved to allow the user to make changes to the problem. Changes could be made to correct earlier data-entry errors, or to study how the changes affect the solution. Whenever the user leaves the graphics display, a menu is printed on the text screen. The menu approach was chosen because it clearly shows the user what options are available. The options available from the main menu are:

1. Add a new constraint, A(dd).
2. Change an existing constraint, C(hange).
3. Delete an existing constraint, D(lete).
4. List the constraints and the objective function, L(ist).
5. Return to the plot, R(eturn).
6. Quit; end execution of the program, Q(uit).

The user is asked to type A, C, D, L, R or Q: which correspond to the six options listed above. If any character other than those listed above is entered, the bell sounds, an error message is printed and the user is again asked to type A, C, D, L, R or Q. The main menu is shown in Figure 2.

When the add option is selected, the system checks to see if nine constraints have been entered already. If so, an error message is printed and the main menu is displayed.
The add option tells the user which constraint number is being added. The constraints are numbered in the order in which they were entered. The user is then asked to enter the two decision variable coefficients, the right side of the equation, and the type of constraint (less than, equal, or greater than). The only error checking that is done is to check that the response is <, = or >. After all data for the constraint have been added, it is displayed and the user is asked if it is correct. At this point, a menu containing three options is displayed. The options are:

1. Accept the new constraint.

2. Reject the new constraint, and immediately try to enter it again.

3. Reject the new constraint and return to the menu.

These three options allow data-entry errors to be detected and corrected by the user. This type of error is not detectable by the system. The third option allows an easy exit from the add routine when it is entered accidentally.

The change option allows one constraint to be replaced by another. It has two uses. If the user wants to delete one constraint and then add a new one, change is more direct than first deleting one and then adding the new one. The change option also allows the user to alter part of the data that defines a constraint; the right side, for example. Change also allows the objective function to be changed. When change is selected, it first lists all of the con-
straints and the objective function on the screen. The user is then asked which one is to be changed. The option of returning to the main menu is also available. The response is checked for validity, and if it is valid, the user is asked to enter the new data for the constraint. After the data has been entered, the revised constraint is displayed and the user has the same options as in the add option at the same point. The display, immediately after selecting the change option, is shown in Figure 3.

The delete option allows one constraint to be removed from the problem. LINGRAF first checks to see that more than one constraint exists. If only one exists, the user is informed that only one is left, and it may not be removed. The main menu is then restored, without any change to the problem. All of the constraints are displayed on the screen (the objective function is not displayed, because it can not be deleted). The user is then asked which constraint is to be deleted. The response is checked for validity. If it is valid, the chosen constraint is displayed on the screen and the user is asked if he or she really wants to delete it. The user may then either allow it to be removed and return to the menu or return to the menu without removing it.

The list option prints a list of the constraints and the objective function on the screen. The user is then asked to press any key (except control, shift or reset) to return to the main menu. The purpose of the list option is to allow the user to review the problem.
YOU MAY NOW:

A - ADD A NEW CONSTRAINT
C - CHANGE AN EXISTING CONSTRAINT
D - DELETE AN EXISTING CONSTRAINT
L - LIST THE CONSTRAINTS & THE O. F.
R - RETURN TO THE PLOT
Q - QUIT; END THE PROGRAM

PLEASE TYPE A,C,D,L,R OR Q ?

Figure 2
LINGRAF Main Menu

CHANGE A CONSTRAINT

THE CONSTRAINTS ARE:
1:  1 * X1 + 0 * X2 < 4
2:  0 * X1 + 2 * X2 < 12
3:  3 * X1 + 2 * X2 < 18

THE OBJECTIVE FUNCTION IS:

MAXIMIZE:  3 * X1 + 5 * X2

WHICH CONSTRAINT DO YOU WISH TO CHANGE?
-1 WILL RETURN YOU TO THE MENU
0 WILL CHANGE THE OBJECTIVE FUNCTION
A NUMBER WILL CHANGE THAT CONSTRAINT
 WHICH ONE?
PLEASE ANSWER -1, 0, 1, 2, OR 3: ?

Figure 3
Change Constraint or
Objective Function Display
The return option allows the user to return to the plot. If the problem has been modified (i.e. any constraints have been added, deleted or changed) since the last time the plot was displayed, the problem is solved again. This is done automatically. The user does not have to initiate it. If the objective function has been changed, the graphical solution is not done again, since the plot will not change. If no changes have been made to the problem, the old plot is restored.

The quit option is the means of exiting from LINGRAF. If it is selected, control is passed to the disk operating system and the computer waits for whatever commands the user may wish to type.

The user interface thus was designed to reduce errors in two ways. First, a menu driven system with extensive prompting was utilized to help reduce errors by keeping the user aware of exactly what is expected at any time. Second, all possible input is checked for validity. If it is found to be invalid, the user is so informed and is again given a prompt listing all valid responses.

3.2.2 **Shading**

Perhaps the most important feature of LINGRAF is the manner in which it derives the graphical solution. The user is able to watch the screen change from a blank screen to the final solution as constraints are added and as the fea-
The feasible region is updated, one constraint at a time. The method was chosen for its simplicity and speed. As mentioned previously, the Applesoft language does not have any shading capabilities. Thus, one of the more difficult tasks is the shading of the feasible region. In order to shade a region in Applesoft, a series of adjacent horizontal (or vertical) lines must be drawn. The only real problem is determining the end points of these lines. In the case of linear programming, this would be possible, but time consuming, thus violating the execution speed criterion.

The method developed to shade the feasible region uses both Applesoft and assembly language routines. If there is only one constraint, shading is fairly simple. For each horizontal line, a line is drawn from the constraint to either the left side of the plot, for less than constraints, or the right side of the plot, for greater than constraints. There are two exceptions to this but each is even simpler to handle. Equals constraints do not require shading. Constraints that are horizontal are handled by shading from the left side of the plot to the right side, either above the constraint, for greater than constraints, or below it, for less than.

The method by which one constraint's feasible region may be shaded forms the basis for the method used to shade the problem's feasible region. The feasible region resulting from all of the constraints may be thought of as the in-
tersection of the feasible regions resulting from each of the individual constraints. The Apple II has two separate high-resolution picture buffers. Either may be displayed at any time, and either may be altered at any time, regardless of whether or not it is being displayed. Each utilizes 8K (8192) bytes of RAM, and the data storage format is the same in each. LINGRAF draws each constraint and shades its feasible region in the first high-resolution screen buffer, known as high-resolution page one. After the shading is complete, the contents of this buffer are logically ANDed, one byte at a time, with the contents of the second buffer, known as high-resolution page two. The result is stored in page two, which is always displayed. To initialize the process, the first constraint's feasible region is copied to page two, instead of being ANDed. After all constraints have been processed, the result is the feasible region for the entire problem. Items such as axes and titles need to be drawn on both pages, so that they will remain after the AND operation. Figure 4 illustrates this process.

The method which LINGRAF uses to shade the feasible region is acceptably fast. It uses a combination of BASIC and assembly language routines. BASIC is used to control the loop that shades each horizontal line that needs to be shaded, and also to determine when to stop shading. Assembly language is used to find the x-coordinate of the constraint's intersection with the horizontal line to be
Start of loop for each constraint

Clear HIRES page 1 to black

Draw current constraint on both HIRES pages

Shade the feasible region for this constraint on HIRES page 1

Plot axes on HIRES page 1

First const?

Copy HIRES page 1 to HIRES page 2

View HIRES page 2

End of loop for each constraint

Plot previous constraints on HIRES page 1

AND HIRES pg 1 with pg 2; store result on pg 2

Figure 4
Determining the Graphical Solution
shaded. Each horizontal line on the high-resolution screen corresponds to forty contiguous bytes in the appropriate buffer. A complete description of the format in which the picture information is stored may be found in references 2, 3 and 5.

Each byte in the buffer corresponds to seven pixels on the screen. A non-zero bit in bits one through six corresponds to a point plotted in some color other than black. If the individual constraints that are plotted on page one are plotted in white on a black background, finding the x-coordinate of the first plotted point on a horizontal line is possible. The procedure is to check each of the forty bytes (in the corresponding buffer) that make up the horizontal line until one is found that contains a number other than zero. That byte may then be examined to determine which bit is set. Once the byte and the bit have been determined, the x-coordinate may be easily determined. This is done by the assembly language routine named LINE. Assembly language was used for speed and because Applesoft lacks bit-manipulation capabilities.

Before LINE can find the first non-black point on a horizontal line, the address of the first of the forty bytes in the buffer that correspond to that line must be determined. Adjacent horizontal lines on the high-resolution screens do not use adjacent areas of memory. Each screen buffer, as a whole, uses a contiguous area in memory, but
the lines are not contiguous within the buffer. For this reason, the address of the first byte must be determined for each horizontal line. The following equation gives this address, referred to as the base address.

\[ B = P + 1024(Y \text{mod} 8) + 128(\text{int}(Y/8) \text{mod} 8) + 40\text{int}(Y/64) \]

\(B\) is the base address, \(Y\) is the \(y\)-coordinate of the horizontal line (in plot coordinates) and \(P\) is \(8192\) for page 1 and \(16384\) for page 2. Because this equation must be evaluated for each horizontal line that is shaded, it is evaluated by an assembly language routine named BASECALC. Despite its apparent complexity, this equation may be quickly evaluated in assembly language, because all but one of the constants involved are powers of two. Thus, the multiplications and divisions are merely a set of shift operations.

The shading starts at the bottom line of the plot (the first horizontal line above the axis). It continues until either the top of the plot is reached, or a point is found adjacent to the \(y\)-axis. Note that the edges of the plot do not correspond to the edges of the high-resolution screen. Margins are provided to allow space for titles, prompts, axis labels, etc.
3.2.3 Moving the Objective Function Line

Another important feature of LINGRAF is its ability to move the objective function line parallel to itself across the screen. Applesoft allows a line to be drawn across the plot, but does not provide a simple means of erasing that line. A technique for removing the objective function line from the plot is necessary. After the problem has been solved, LINGRAF maintains two copies of the plot, one in each of the high-resolution buffers. One copy, the one in page two, is a permanent copy that is not altered. The objective function line is drawn on high-resolution page one. When the position of the objective function has to be changed, the display is temporarily changed from page one to page two. This gives the illusion that the line has been removed. Next, the contents of page two, the permanent copy, are copied to page one. This restores page one to a copy of the plot without an objective function line. At this point the new objective function line is drawn on page one. Finally, the display is switched back to page one, to show the plot with the new objective function line drawn on it. The entire process is done in BASIC, except for copying one buffer to the other. This is done by the assembly language routine MOVEHGR. Assembly language was chosen because the same process, the transfer of 8192 bytes, would take far longer to execute in BASIC than in assembly language. The process of moving the objective function line is illustrated in Figure 5.
Switch the display to HIRES page 2 (the permanent copy of the solution, without an objective function line) to "erase" the objective function line

Copy the HIRES page 2 buffer (the permanent copy of the solution) to the HIRES page 1 buffer

Draw the new objective function line on HIRES page 1

Switch the display to HIRES page 1 to display the new objective function line

Figure 5
Moving the Objective Function Line
The process of drawing the objective function line is complicated by the fact that Applesoft does not provide any clipping capabilities. Thus, LINGRAF must calculate the endpoints of the objective function line. This is complicated by the fact that the endpoints may lie on any of the edges of the plot. LINGRAF uses the following technique to draw the line. The line may be thought of as running from some point on a horizontal line at the top of the plot to some point on a horizontal line at the bottom of the plot. If these horizontal lines are considered to extend indefinitely in both directions, they may be used as a reference for plotting the line. The y-coordinates of the end points of the line are thus fixed, so only the x-coordinates need to be computed. The last objective function line's intersections with this line are stored, and when a move command is issued, the appropriate value is added to these values. They are then checked for validity since either or both may extend past the sides of the plot. If either is found to be out of bounds, its intersection with the appropriate side is then computed. If this is inside of the plot, the line is drawn. If not, the line is not drawn because it is entirely off the plot. In such a case, a tone sounds three times, to inform the user that the line is off the plot. The user may then issue the appropriate move command to move it back onto the plot.
The routine to animate the objective function was designed with several objectives. First, the objective function line must be easily discernible from the constraint lines. It is drawn in green, while the constraints are drawn in white. Also, the objective function line may be moved. Thus, to identify it on a crowded plot, the user only needs to move it slightly. A second objective was speed. By using the method described to move the line, and by using an assembly language routine to copy high-resolution page two's buffer to page one's buffer, the process takes about 0.5 seconds. If the user wishes to move the line a long distance across the plot, enough steps may be required that the process would take long enough to be annoying. For this reason, a second set of move commands were added. The standard move commands are the left and right arrow keys. The additional move commands are the greater-than and less-than symbols. These commands move the line in steps five times as large as the arrow keys. They may be used to move the line over long distances quickly. When the line approaches the point the user desires, the arrow keys must be used since the greater than and less than keys take such large steps that the line may miss the desired point.
Chapter IV
SYSTEM INTEGRATION

4.1 GRAPHICS

The Apple II is advertised as being able to use six colors in the high-resolution graphics mode. There are certain conditions under which only four colors are practical. Because of this, LINGRAF uses only four colors. A single byte in one of the high-resolution picture buffers contains information on seven pixels. Elementary information theory reveals that eight bits of information are not sufficient to contain information on seven pixels, each of which may be any of six colors. The following restrictions apply to the use of the colors:

1. Pixels in even columns must be black, violet or blue.
2. Pixels in odd columns must be black, green or red.
3. It is not possible to mix green and blue, green and red, violet and blue, or violet and red pixels in the same byte in the buffer.
4. Two adjacent colored (not black or white) pixels will appear to be white.

The four colors that were selected for LINGRAF are black, white, blue and green. They were selected for their compat-
ibility with each other. The background of the plot is black. White is used for all lettering, as well as the axes and the constraint lines. The axes are the same color as the constraint lines because the nonnegativity constraints coincide with the axes. The feasible region is shaded in blue, and the objective function line is drawn in green.

4.2 MEMORY ALLOCATION

All of LINGRAF needs to reside in memory concurrently. Allocation of memory was an important step in the design of LINGRAF. The 49152 bytes of memory need to contain the BASIC main program and subroutines, the assembly language subroutines, variable storage for BASIC and assembly language programs, the High Resolution Character Generator program, and the disk operating system (referred to as the DOS). The random access memory is addressed as locations 0 through 49151 (decimal). Locations 0 through 2047 are used by the operating system. BASIC programs are stored starting at location 2048 and build upward in memory. Simple variables (variables that are not strings or arrays) start just after the end of the BASIC program and build upward in memory. Arrays, except string arrays, start just after the end of the simple variables and also build upward. String variables and arrays start at HIMEM and build downward in memory. The manner in which string variables are allocated storage often allocates a new storage area to a string when
a its value is changed. This leaves the old storage area still in use, but flagged as inactive, i.e. the data is still stored, but Applesoft recognizes that it is no longer correct. If the end of string storage gets dangerously close to the end of simple array storage, a garbage collection process is initiated for string storage. This garbage collection packs the active strings together, removing the gaps where inactive strings were stored. The memory map for LINGRAF is shown in Figure 6.

One major difficulty that was encountered in the design process is that the two high-resolution picture buffers are located from address 8192 through address 24576. This limits the size of the BASIC program and its variables (except for strings and string arrays) to 6144 bytes if both high resolution pages are used. There is a large amount of free memory above the buffers, however. LINGRAF's BASIC program, not including variable storage, extended past location 8192. This was remedied by splitting the program into two parts, with a 16384 byte gap in the middle. The gap is the memory area used for the high-resolution buffers (locations 8192 through 24576). The procedure for splitting the program (which must be done after loading it, but before running it) may be found in reference 8. The BASIC programs listed in Appendix B include many REM (remark) statements. These annotate the program but are otherwise unnecessary. In fact, they increase the memory required to store the program to
<table>
<thead>
<tr>
<th>Decimal Address</th>
<th>Hexadecimal Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>53248 - 65535</td>
<td>$D000 - $FFFF</td>
<td>System ROM</td>
</tr>
<tr>
<td>49152 - 53247</td>
<td>$C000 - $CFFF</td>
<td>I/O addresses</td>
</tr>
<tr>
<td>38400 - 49151</td>
<td>$9600 - $BFFF</td>
<td>Disk Operating System (DOS 3.3)</td>
</tr>
<tr>
<td>* - 38399</td>
<td>* - $95FF</td>
<td>BASIC string variable storage (builds downward)</td>
</tr>
<tr>
<td>37632 - 38086</td>
<td>$9300 - $94C6</td>
<td>LINGRAF assembly language subroutines</td>
</tr>
<tr>
<td>34302 - 37631</td>
<td>$85FE - $92FF</td>
<td>High Resolution Character Generator</td>
</tr>
<tr>
<td>24576 - 34301</td>
<td>$6000 - $85FD</td>
<td>BASIC program (cont.) and variable storage</td>
</tr>
<tr>
<td>16384 - 24575</td>
<td>$4000 - $5FFF</td>
<td>HIRES page two buffer</td>
</tr>
<tr>
<td>8192 - 16383</td>
<td>$2000 - $3FFF</td>
<td>HIRES page one buffer</td>
</tr>
<tr>
<td>2048 - 8191</td>
<td>$8000 - $1FFF</td>
<td>BASIC program storage</td>
</tr>
<tr>
<td>1024 - 2047</td>
<td>$0400 - $07FF</td>
<td>Text display buffer</td>
</tr>
<tr>
<td>976 - 1023</td>
<td>$03D0 - $03FF</td>
<td>Monitor &amp; DOS vectors</td>
</tr>
<tr>
<td>768 - 975</td>
<td>$0300 - $03CF</td>
<td>Unused</td>
</tr>
<tr>
<td>512 - 767</td>
<td>$0200 - $02FF</td>
<td>System input buffer</td>
</tr>
<tr>
<td>256 - 511</td>
<td>$0100 - $01FF</td>
<td>System stack</td>
</tr>
<tr>
<td>252 - 255</td>
<td>$00F0 - $00FF</td>
<td>LINGRAF zero-page storage</td>
</tr>
<tr>
<td>0 - 251</td>
<td>$0000 - $00FB</td>
<td>System zero-page storage</td>
</tr>
</tbody>
</table>

* varies as program executes

Figure 6
LINGRAF Memory Map
beyond what is available for the PASiC portion of LINGRAF. For this reason, a program named the Applesoft Programmer's Assistant, which is included in the DOS Toolkit (available from Apple Computer), was used to remove all REM statements from the program. It is this shortened form of the program that actually executes.

One problem that was encountered during testing was that after running the program for an extended length of time, it would suddenly stop executing and the system would hang. Resetting the processor was the only way to regain control of the computer. Analysis revealed that the problem was caused by the string variable storage area extending too far and overwriting memory used for other uses. This was solved by forcing a string variable garbage collection to take place each time the main menu is displayed. The garbage collection is forced by a call to the Applesoft FREE function, which returns the amount of free memory. Before it calculates this, it calls the garbage collection routine. Further testing has shown that this solved the problem.

LINGRAF has been tested on many problems. Some of these tests were run for extended periods of time. Such tests involved moving the objective function back and forth and frequently calling all of the functions available in the main menu. The objective of the extended tests was to detect problems such as the one described in the previous paragraph. The only problems that are still known to exist
are due to limitations of the hardware. Problems where the feasible region is a line or a point do not show the feasible region in green, as it is not always possible to plot a given line or point in green. A second problem is that if the objective function line is vertical, it will appear only in alternate columns. It will be invisible in the other columns. This is also due to the limitations of plotting green points and lines. Plotting the objective function line in white would remedy this, but plotting it in green was preferred so that it would be easily discernible from the constraints. A vertical objective function line is not likely to be found in many problems.
5.1 LIMITATIONS

As with any system, LINGRAF has certain limitations. They include:

1. LINGRAF is not the most efficient means of solving linear programming problems. Packages that give only the numerical solution will usually be faster, thus they may be preferable if only the numerical solution is required. LINGRAF is intended to be a tool for illustrating the graphical solution technique, and not to be a general program for solving Linear Programming problems.

2. The maximum number of constraints that a problem may have is nine. The maximum of nine was chosen for two reasons. First, few problems used to illustrate the graphical solution technique are likely to have more than nine constraints. Second, with nine as the maximum, a simple means of causing one constraint to flash was possible; pressing one of the keys for the digits one through nine.

3. The feasible region will not be shaded when it is a line or a point. This is because it is not always possible to plot a given point or line in green.
5.2 SUGGESTIONS FOR FURTHER RESEARCH

There are several improvements which could be made to LINGRAF.

1. LINGRAF could be expanded to solve problems with three decision variables. The feasible region for a problem with three decision variables is a solid. Many algorithms exist for plotting three-dimensional shapes. The largest problem would be finding a means of clearly showing the intersection of the plane that would represent the objective function and the solid that would represent the feasible region. The ability to view the feasible region from any angle would be desirable. Such a change would probably require a computer with higher resolution graphics than the 280 by 192 pixels of the Apple II. Because of the increased complexity of three-dimensional problems, more of the program would need to be written in assembly language. This would allow the programs to execute with sufficient speed to be practical.

2. The constraints could be allowed to be moved in the same manner that the objective function currently may be moved. This would require either a faster shading technique, or the existing technique to be implemented completely in assembly language. This is due to the need to determine the new feasible region after each move.
3. More of LINGRAF could be written in assembly language. This would increase execution speed and decrease memory requirements allowing additional features to be added to LINGRAF. Programming the remainder of LINGRAF in assembly language would be a formidable task, however.

4. LINGRAF could be adapted to run on other popular microcomputer systems. This would probably require a new shading technique, as well as rewriting most, if not all, of the assembly language routines. Required modifications to the BASIC program would be numerous, but mostly of a minor nature.
BIBLIOGRAPHY


Appendix A
LINGGRAF USER'S MANUAL

LINGGRAF is a software package that solves linear programming problems with two decision variables and displays the graphical solution in high-resolution graphics.

HARDWARE REQUIREMENTS

The following Apple II-series systems are capable of running LINGGRAF.

1. Any Apple IIc.
2. An Apple IIe with one or more disk drives.
3. An Apple II Plus with at least 48K of RAM and one or more disk drives.
4. An Apple II with at least 48K of RAM, one or more disk drives, and either the Applesoft ROM card (an Apple accessory) or Applesoft loaded into an Apple Language Card (an Apple accessory) or functional equivalent.

RUNNING LINGGRAF

LINGGRAF is run by inserting the LINGGRAF disk in the disk drive and switching on the power to the computer. LINGGRAF gives descriptive prompts wherever possible, so reading through this manual once should be adequate preparation to use it. Whenever LINGGRAF asks you to press any key, press any key, except the shift key, the control (CTRL) key or the RESET key. LINGGRAF always explains what input it expects. Whenever LINGGRAF asks you to choose among several alternatives, all of the alternatives are listed. For example, after the menu is displayed you are asked to type A, C, D, L, R or Q. Thus, you should not have any trouble determining what information LINGGRAF desires.
ENTERING THE PROBLEM

Once LINGRAF has loaded, you will be asked how many constraints are in your problem. You must choose an integer between 1 and 9. You will then be asked to enter data for each constraint. The required data are:

1. The coefficient of decision variable X₁
2. The coefficient of decision variable X₂
3. The type of constraint: <, =, or >
4. The right side of the constraint equation

After the data for all of the constraints and the objective function have been entered, the problem will be solved. The display will show the problem being solved, as each constraint is added and the feasible region is updated to include that constraint.

When the solution has been found, it will be displayed and you will have several options. You may move the objective function line to the left or right by pressing the left- or right-arrow keys. The optimal solution, as well as the value of the objective function line that is currently being displayed will be shown. If you wish to move the line a long distance across the plot, the < and > keys may be used in place of the arrow keys. These keys move the line in large steps, so as you near the desired destination, switch back to the arrow keys. You may identify any constraint by pressing the key with that constraint's number. This will cause that constraint to flash until another key is pressed. To review the constraint numbers, use the list option described below.

While viewing the solution, you may switch to the main menu. To do this, press the escape (ESC) key. The menu gives you several options:

1. Add a new constraint
2. Change an existing constraint
3. Delete an existing constraint
4. List the constraints and the objective function
5. Return to the plot
6. Quit; exit from LINGRAF

To select one of these options, press A, C, D, L, R or Q.
Add

Add will ask you to enter all of the data for the new constraint equation, just as you did when entering the original problem. It will then display the new constraint equation for your approval. You may then accept it and return to the menu, reject it and retype it, or reject it and return to the menu. You will not be allowed to add a constraint if nine constraints already exist.

Change

Change allows you to change an existing constraint equation, or the objective function equation. Change first displays the constraint equations and the objective function equation and then asks which you wish to change. Typing "-1" returns you directly to the menu. If you select one of the constraints, you will be asked for all of the data for that constraint equation, just as in the add option. After you have entered the data, the revised constraint equation will be displayed for your approval. You may then accept it and return to the menu, reject and retype it, or reject it and return to the menu. If you select the objective function equation, you will be asked for the coefficients of the two decision variables, as well as whether the objective function is to be minimized or maximized. The equation will then be displayed for your approval, as above.

Delete

Delete allows you to remove one constraint equation from the problem. You may remove several constraints by calling delete once for each. Delete first displays the constraint equations and asks which you wish to delete. If you do not wish to remove any, answer "-1". After you have selected a constraint to delete, it will be displayed for your approval. You may then have it deleted and return to the menu, or return to the menu without removing it. Delete may not be used when there is only one constraint.

List

The list option lists the constraint equations and the objective function equation. It also shows if the objective function is to be minimized or maximized. The numbers to the left of the constraints are the constraint numbers that are used to refer to the constraints when making one flash, or using the add, change or delete options.
Return

Return is the means of exiting the menu and returning to the plot. LINGRAF automatically determines if the existing plot is still valid, or a new plot needs to be drawn. This would happen when any constraints were added, changed or deleted.

Quit

The quit option allows you to exit from LINGRAF. The computer will then be ready to accept any DOS or BASIC commands.
Appendix B

BASIC PROGRAM LISTINGS
10 REM *****************************************************************************
20 REM *
30 REM * LINGRAF - An Interactive Teaching Aid for Linear Programming *
40 REM *
50 REM *
60 REM * Copyright (C) 1985 By Stan Bevis *
70 REM *
80 REM * Note: Lines 9150 to 10670 adapted from Some Common Basic Programs, By Lon Poole and Mary Borchers. *
90 REM *
100 REM * Berkeley, California: Osborne/McGraw-Hill, *
110 REM *
120 REM *
130 REM *****************************************************************************
140 REM
150 CALL - 936: VTab (3): HTab (11): Print "":
160 Print "LINGRAF - AN INTERACTIVE TEACHING AID FOR LINEAR PROGRAMMING"
170 Print
180 REM *****************************************************************************
190 REM Input the initial problem
200 REM *****************************************************************************
210 N = 2: REM NO OF DEC VRBLS
220 Print "HOW MANY CONSTRAINTS ARE THERE?"
230 Input "PLEASE ENTER A NUMBER BETWEEN 1 AND 9 "; M
240 Print
250 IF M > 0 AND M < 10 THEN 300
260 Print
270 Print Chr$ (7); "***ERROR: INVALID RESPONSE;"
280 Goto 230
290 REM DIMENSION FOR 9 CONSTRAINTS
300 Dim A(9,11), RS(9), BV(9), SOLN(11), AS(11.22)
310 For Row = 1 To M
320 For Col = 1 To N
330 Print "ENTER COEFF OF X";Col;" FOR CONSTRAINT "; Row; " ";
340 Input A(Row,Col)
350 Next Col
360 Print "ENTER RIGHT SIDE OF EQN "; Row; " ";
370 Input RS(Row)
380 Print : Print "WHAT TYPE OF CONSTRAINT IS THIS; <, = OR >?"
390 Print
400 Input "PLEASE ENTER <, = OR > "; A$
410 IF A$ = "">" THEN A(Row,0) = 1: Goto 450
420 IF A$ = "<" THEN A(Row,0) = 255: Goto 450
430 IF A$ = "=" THEN A(Row,0) = 0: Goto 450
440 Print Chr$ (7); "***ERROR: INVALID RESPONSE"; Goto 390
450 Call - 936: VTab 5
460 Next Row
470 For Col = 1 To N
480 Print "ENTER COEFF OF X";Col;" OF OBJ FN ";
490 Input A(O,Col): A(O,Col) = -1 * A(O,Col)
500 Next Col
510 Print
520 Print "DO YOU WISH TO MINIMIZE OR MAXIMIZE"
530 PRINT "THE OBJECTIVE FUNCTION?"
540 PRINT
550 INPUT "PLEASE TYPE MIN OR MAX: ";Z$  
560 IF Z$ = "MIN" OR Z$ = "MAX" THEN 590
570 PRINT CHR$ (7);"***ERROR INVALID RESPONSE "
580 GOTO 550
590 Z$ = Z$ + "IMIZE"
600 REM ***********************************************
610 REM Assign memory addresses to constants
620 REM ***********************************************
630 A1 = 37633: REM MOVE
640 A2 = 37651: REM ANDHGR
650 A3 = 37671: REM BLACK
660 A4 = 37740: REM DECODE
670 A5 = 37816: REM KEYPR
680 A6 = 37828: REM LINESCAN
690 A7 = 252: REM SRCADR (2 BYTES)
700 A8 = 254: REM DSTADR (2 BYTES)
710 A9 = 37632: REM TEMPI
720 B1 = 37738: REM IN
730 B2 = 37739: REM IP
740 B3 = 37898: REM YPOSN
750 B4 = 38036: REM KEY2
760 B5 = 99999: REM UNUSED
770 B6 = 37826: REM XL (2 BYTES)
780 REM ***********************************************
790 REM Initialize some variables & set some constants
800 REM ***********************************************
810 F1 = 1: REM SIMPLEX REQ'D
820 HM = 20: REM HORIZ MARGIN
830 VM = 20: REM VERT MARGIN
840 XR = 279 - 2 * HM: REM # X PLOT POSNS
850 YR = 191 - 2 * VM: REM # Y PLOT POSNS
860 REM ***********************************************
870 REM Clear both HIRES screens to black and view page 2
880 REM ***********************************************
890 POKE A9,1: CALL A3: REM BLACK PG 1
900 POKE A9,2: CALL A3: REM BLACK PG 2
910 POKE - 16297,0: REM SELECT HIRES
920 POKE - 16304,0: REM DISPLAY GRAPHICS
930 POKE - 16299,0: REM SELECT PAGE 2
940 POKE - 16302,0: REM FULL-SCRN GRAPHICS
950 HCOLOR= 3: REM WHITE
960 F3 = 0
970 REM
980 REM Solve problem with simplex method, if necessary
990 REM
1000 IF F1 THEN GOSUB 6130: REM SIMPLEX
1010 REM
1020 REM Switch output driver to HRCG
1030 REM
1040 CALL 34305
1050 PRINT CHR$ (15); CHR$ (23); CHR$ (17); CHR$ (15); CHR$ (2)
1060 REM ****************************
1070 REM Determine the scaling for the plot
1080 REM ****************************
1090 XO(0) = 0:YO(0) = 0
1100 REM
1110 REM FIND MAX X- & Y-VALUES TO PLOT
1120 REM
1130 FOR I = 1 TO M
1140 XO(I) = 0:YO(I) = 0
1150 IF A(I,1) = 0 THEN 1180
1160 XO(I) = RS(I) / A(I,1)
1170 IF X0(I) > XO(0) THEN XO(0) = X0(I)
1180 IF A(I,2) = 0 THEN 1210
1190 YO(I) = RS(I) / A(I,2)
1200 IF YO(I) > YO(0) THEN YO(0) = YO(I)
1210 NEXT I
1220 REM
1230 REM Make X & Y scaling equal
1240 REM
1250 IF (YO(0) / XO(0)) > (YR / XR) THEN 1300
1260 REM X IS DECIDING AXIS
1270 YO(0) = YR * XO(0) / XR
1280 GOTO 1340
1290 REM Y IS DECIDING AXIS
1300 XO(0) = XR * YO(0) / YR
1310 REM ****************************
1320 REM Loop to plot each constr, shade its FR and update solution
1330 REM ****************************
1340 REM DRAW AXES ON GRAPHICS PAGE 2
1350 PG = 64
1360 GOSUB 5760
1370 REM LOOP FOR EACH CONSTRAINT
1380 FOR I = 1 TO M
1390 REM BLACK OUT HIRES PG 1
1400 POKE 19, I: REM TEMPI
1410 CALL A3: REM BLACK
1420 REM
1430 REM FIND WHERE TO PLOT THE NEXT CONSTRAINT
1440 REM
1450 X1 = HM
1460 Y1 = VM + YR * (YO(0) - YO(I)) / YO(0)
1470 X2 = HM + XR * XO(I) / XO(0)
1480 Y2 = 191 - VM
1490 REM
1500 REM TAKE CARE OF VERT. & HORIZ. CONSTS
1510 REM
1520 IF NOT XO(I) THEN X2 = 280 - HM:Y2 = Y1
1530 IF NOT YO(I) THEN X1 = X2:Y1 = VM
1540 REM
1550 REM PLOT THE NEW CONSTRAINT ON BOTH PAGES
1560 REM
1570 FOR PG = 32 TO 64 STEP 32
1580 POKE 230,PG
1590 HPlOT X1,Y1 TO X2,Y2
1600 NEXT PG
1610 POKE 230,32
1620 HOME
1630 PRINT
1640 PRINT SPC(10); CHR$(12);"ADDING CONSTRAINT "; CHR$(11);I
1650 POKE -16299,0: REM VIEW PG 2
1660 REM
1670 REM SAVE THIS CONSTRAINT'S END POINTS
1680 REM
1690 C(I,1) = X1:C(I,2) = Y1
1700 C(I,3) = X2:C(I,4) = Y2
1710 REM
1720 REM SHADE FEASIBLE REGION FOR THIS CONSTRAINT ONLY
1730 REM
1740 REM DON'T SHADE = CONSTRAINTS
1750 IF A(I,0) = 0 THEN 1970
1760 HCOLOR= 2: REM BLUE
1770 REM
1780 REM FIND WHERE TO DRAW TO
1790 REM
1800 IF A(I,0) = 255 THEN X2 = HM + 2: GOTO 1820
1810 X2 = 280 - HM: REM FOR --> CONSTRAINT
1820 FOR Y = 191 - VM TO VM STEP - 1
1830 POKE B3,Y
1840 CALL $6: REM LINESCAN
1850 X1 = PEEK (B6) + 256 * PEEK (B6 + 1): REM LINE OUTPUT
1860 IF X1 > 280 - HM THEN X1 = 280 - HM
1870 REM
1880 REM DONE SHADING?
1890 REM
1900 IF X1 = HM + 1 THEN 1970
1910 REM
1920 REM DRAW A HORIZ LINE
1930 REM
1940 HPlOT X1,Y TO X2,Y
1950 NEXT Y
1960 REM FINISH SHADING (> CONSTRAINTS)
1970 IF A(I,0) < > 1 THEN 2010
1980 FOR YY = Y TO VM STEP - 1
1990 HPlOT HM + 1,YY TO 280 - HM,YY
2000 NEXT YY
2010 HCOLOR= 3
2020 REM
2030 REM PLOT AXES ON GRAPHICS PAGE 1
2040 REM
2050 PG = 32
2060 GOSUB 5760
2070 IF I = 1 THEN 2160
2080 REM HPlOT HM,VM TO HM,191 - VM TO 279 - HM,191 - VM
2090 IF I = 1 THEN 2170
2100 FOR Y = 1 TO I - 1
2110 HPlOT C(Y,1),C(Y,2) TO C(Y,3),C(Y,4)
2120 NEXT Y
2130 REM
2140 REM IF 1ST CONSTR, COPY PLOT TO PG 2
2150 REM
2160 IF I < > 1 THEN 2230
2170 POKE A9,2: REM DEST = PG2
2180 CALL A1: REM COPY
2190 GOTO 2300
2200 REM
2210 REM DO THIS FOR 2ND & LATER CONSTRS
2220 REM
2230 POKE A9,2: REM DEST = PG2
2240 CALL A2: REM ANDHGR
2250 POKE 230,64
2260 FOR II = 1 TO I
2270 HPlot C(II,1),C(II,2) TO C(II,3),C(II,4)
2280 NEXT II
2290 POKE 230,32
2300 NEXT I
2310 HOME
2320 PRINT CHR$(12);"PLEASE PRESS <, >, ARROW, NUMBER OR ";
2330 PRINT CHR$(11);"ESC "; CHR$(12)
2340 REM ******************************************************
2350 REM The problem has now been solved, and the solution is being
2360 REM displayed on the screen. The following loop is controls
2370 REM the program from now until the user terminates the program.
2380 REM ******************************************************
2390 REM
2400 MZ = A(0,2) / A(0,1)
2410 ZM = 1 / MZ
2420 POKE 230,32: REM TO BE SAFE!!
2430 XS = HM + 1
2440 XF = 280 - HM
2450 YS = VM + 1
2460 YF = 190 - VM
2470 REM
2480 REM WHERE DOES Z-OPT CROSS X-AXIS?
2490 REM
2500 R1 = ((XR * SOLN(1)) / X0(0)) + HM + 1
2510 R2 = ((YR * (YO(0) - SOLN(2))) / Y0(0)) + VM + 1
2520 R3 = R1 + (190 - VM - R2) * MZ
2530 HOME : VTAB 24: PRINT SPAC(10);"OPTIMAL Z = ";SOLN(0);
2540 E8 = X0(0) / (277 - 2 * HM)
2550 E4 = SOLN(1) + SOLN(2) * MZ;E6 = E4
2560 REM
2570 REM DRAW THE FIRST LINE
2580 REM
2590 REM AX = YS + INT (((XF - XS) / 2)
2600 AX = R3
2610 AY = YF
2620 BX = AX - INT (((YF - YS) * MZ)
2630 BY = YS
2640 T1 = AX:T2 = BX
2650 REM
2660 REM AX IS OK, SO SKIP TO CHK BX
2670 REM
2680 GOTO 2980
2690 REM ************************************************
2700 REM Start of main z-line animation / command interpreter loop
2710 REM ************************************************
2720 REM
2730 REM Get next z-line move command
2740 REM
2750 CALL A4: REM DECODE
2760 DZ = PEEK (B2) - PEEK (B1)
2770 REM
2780 REM If the key was the ESCape key, display the main menu
2790 REM
2800 IF NOT DZ THEN CALL - 936: TEXT : GOTO 3120
2810 REM
2820 REM If a digit key (1..9) was pressed try to flash that constraint
2830 REM
2840 IF DZ < -10 THEN GOTO 3810
2850 REM ************************************************
2860 REM Move the z-line to a new position
2870 REM (The user pressed <, >, <-, or ->)
2880 REM ************************************************
2890 REM
2900 REM Find the x-intercept of the new z-line
2910 REM
2920 AX = T1 + DZ
2930 AY = YF
2940 T1 = AX: REM save it!
2950 REM
2960 REM does it need to be clipped?
2970 REM
2980 IF AX < XF THEN 3100
2990 REM
3000 REM clip to enter on right side
3010 REM
3020 BZ = AY - ZM * AX
3030 AX = XF
3040 AY = ZM * AX + BZ
3050 REM the next test would always fail, so skip it
3060 GOTO 3140
3070 REM
3080 REM Is the new line completly off the screen?
3090 REM
3100 IF AX < XS THEN 3710
3110 REM
3120 REM try this for top end point
3130 REM
3140 BY = T2 + DZ
3150 BY = YS
3160 T2 = BX: REM SAVE IT!
3170 REM
3180 REM Does it need to be clipped?
3190 REM
3200 IF BX = > XS THEN 3310
3210 REM adjust to leave on right side
3220 BX = XS
3230 B = AY - YM * AX
3240 BY = YM * BX + B
3250 BX = XS
3260 REM the next test would always fail, so skip it
3270 GOTO 3370
3280 REM
3290 REM Is the new line completely off the screen?
3300 REM
3310 IF BX > XF THEN 3710
3320 REM
3330 REM Get ready to draw the new z-line
3340 REM
3350 REM View HIRES page 2 (the permanent copy)
3360 REM
3370 POKE - 16299,0
3380 REM
3390 REM copy HIRES page 2 to HIRES page 1
3400 REM
3410 POKE A9,1: REM TEMP1
3420 CALL A1: REM MOVE
3430 REM
3440 REM Draw on HIRES page 1, in green
3450 REM
3460 POKE 230,32: REM to draw on HIRES page 1
3470 HCOLOR= 1: REM to draw in green
3480 REM
3490 REM Draw the new z-line (on HIRES page 1)
3500 REM
3510 HLOT AX,AY TO BX,BY
3520 REM
3530 REM View HIRES page 1 (so we can see our nifty new line)
3540 REM
3550 POKE - 16300,0
3560 REM
3570 REM Did the z-line just land on the optimal solution?
3580 REM
3590 E9 = ABS (T1 - INT (R3))
3600 E6 = E6 + DI * EB; E5 = - A(0,1) * E6
3610 HOME ; VTAB (2): PRINT "Z = ";E5;" (OPTIMAL VALUE=":SOLN(0);")": TAB(39)
3620 REM
3630 REM Go back to the start of the animation/command loop
3640 REM
3650 GOTO 2750
3660 REM
3670 REM -------Come here when z-line is completely off screen-------
3680 REM
3690 REM Beep three times & view HIRES pg 2 (it doesn't show z-line)
REM PRINT CHR$(7); CHR$(7); CHR$(7); REM 3 beeps (ASCII BEL)
POKE -16299,0: REM view HIRES page 2
REM Go back to the start of the animation/command loop
REM END OF ANIMATE MODULE
REM Subroutine to flash a constraint (digit key pressed)
POKE 230,32: REM draw on page 1
REM Determine which constraint to flash
REM C = -1*DZ - 10
REM Does constraint number C exist; if not, ignore the command
IF C > M THEN 2750: REM Reenter animate if constr C doesn't exist
REM Tell the user what is happening
HOME
CALL 34350: REM set HRCG as output driver
PRINT CHR$(17); CHR$(12); CHR$(19);
PRINT "CONSTRAINT ";C;" IS FLASHING; PRESS ANY KEY"
REM Draw the constraint in black
HCOLOR= 0: REM BLACK
HPLLOT C(C,1),C(C,2) TO C(C,3),C(C,4)
REM Wait a little while
FOR X = 1 TO 25: NEXT X
REM Draw the constraint in white
HCOLOR= 3: REM WHITE
HPLLOT C(C,1),C(C,2) TO C(C,3),C(C,4)
REM Should the flashing stop (i.e. has any key been pressed)?
CALL A5: REM KEYPR
KEY = PEEK (A9): REM TEMP1
HOME
IF KEY < 128 THEN 4000: REM branch to continue flashing
REM Quit flashing, restore prompt line, reenter animation loop
PRINT CHR$(12);"PLEASE PRESS <, >,ARROW, NUMBER OR ":
PRINT CHR$(11);"ESC "; CHR$(12)
GOTO 2750: REM START OF ANIMATE LOOP

REM******************************************************************************
REM                      Display main menu (ESCape key was pressed)
REM******************************************************************************
CALL - 936: REM clear text screen
REM Force BASIC string garbage collection to keep string space as small as possible and avoid collisions that bomb LINGRAF
REM
ZZ = FRE (0)
REM Restore Apple monitor's Cout routine as output driver
POKE 43603,240: POKE 43604,253
REM Restore the cursor to the text page (it was changed by HRCG)
CALL 65171: REM SETVID (Apple monitor)
CALL 65161: REM SETKBD (Apple monitor)
REM Display the menu and read the user's command
PRINT "YOU MAY NOW:
PRINT "A - ADD A NEW CONSTRAINT"
PRINT "C - CHANGE AN EXISTING CONSTRAINT"
PRINT "D - DELETE AN EXISTING CONSTRAINT"
PRINT "L - LIST THE CONSTRAINTS & THE O. F."
PRINT "R - RETURN TO THE PLOT"
PRINT "Q - QUIT; END THE PROGRAM"
PRINT "PLEASE TYPE A,C,D,L,R OR Q ";
INPUT A$
REM Pass the ASCII code for the key that was pressed to KEY2
ONERR GOTO 4740: REM some control characters bomb line 4620
POKE A9, ASC (A$) + 128
POKE 216,0: REM disable ONERR
REM Call KEY2 to determine which key was pressed. KEY2 returns an integer between one and seven for use in the ON X GOTO
CALL B4 REM KEY2
X = PEEK (A9)
REM Go do whatever the user requested (if the choice was legal)
REM
ON X GOTO 5120,4810,3880,4440,5040,4870,4740
POKE 216,0: REM disable ONERR
REM
4760 REM Come here if any key other than A, C, D, L, R or Q was pressed
4770 REM
4780 PRINT CHR$ (7); "ERROR: INVALID RESPONSE."
4790 GOTO 4560: REM ask for A, C, D, L, R or Q again
4800 REM
4810 REM *******************************************************
4820 REM "Q" (Quit)
4830 REM *******************************************************
4840 REM
4850 REM Clear text screen, display text screen, home cursor
4860 REM
4870 CALL - 936
4880 TEXT
4890 HOME
4900 REM
4910 REM Restore I/O hooks to point to DOS (reconnect DOS)
4920 REM
4930 CALL 976: REM DOS warmstart vector
4940 REM
4950 REM End execution of LINGRAF; return to BASIC
4960 REM
4970 END
4980 REM *******************************************************
4990 REM "R" Return to the plot
5000 REM *******************************************************
5010 REM
5020 REM Change output driver to HRGC
5030 REM
5040 CALL 34305
5050 PRINT CHR$ (15); CHR$ (23); CHR$ (17); CHR$ (15); CHR$ (2)
5060 REM
5070 REM Return to the old plot, if the problem is still the same
5080 REM
5090 IF NOT F0 THEN POKE -16304,0: GOTO 2750
5100 REM
5110 REM The problem has been changed and must be solved again
5120 REM
5130 F0 = 0
5140 GOTO 860
5150 REM
5160 REM *******************************************************
5170 REM "A" Add a new constraint
5180 REM *******************************************************
5190 REM
5200 REM Clear the text screen and display it
5210 REM
5220 CALL - 936; TEXT : VTAB 5
5230 REM
5240 REM Determine if any more constraints may be added (max is 9)
5250 REM
5260 IF M < 9 THEN 5400: REM branch if more may be added
5270 PRINT "YOU HAVE ALREADY ENTERED THE MAXIMUM"
5280 PRINT "NUMBER OF CONSTRAINTS ALLOWED (9)."
PRINT "THUS, NO MORE MAY BE ENTERED."
PRINT
PRINT "PRESS THE ""ESC"" KEY TO END THE"
PRINT "PROGRAM, OR ANY OTHER KEY TO VIEW"
PRINT "THE MENU"
INPUT A#
IF A$ = CHR$ (27) THEN CALL - 936: TEXT : END : REM ESC
GOTO 3040: REM return to the main menu
REM Enter the new constraint
REM
PRINT "NOW ENTERING CONSTRAINT NUMBER ":M + 1
ROW = M + 1
REM Call a subroutine to enter the data for the new constraint
REM
GOSUB 5420
REM
REM Update the number of constraints (there is now one more)
REM
M = M + 1
REM
REM Set flags to indicate that the problem has changed and
REM and must be solved again (both graphically & with simplex)
FO = 1: REM must solve graphically
FI = 1: REM must solve with simplex
REM
REM Return to the main menu
REM
GOTO 3040
REM
REM "D" Delete a constraint
REM
REM Clear text screen, display text screen, home cursor
REM
CALL - 936
TEXT
HOME
PRINT "DELETE A CONSTRAINT"
PRINT
REM
REM Don't allow the last constraint to be removed
REM
IF M > 1 THEN 5840
REM
PRINT
PRINT "ERROR: THERE IS ONLY ONE CONSTRAINT;"
PRINT "IT MAY NOT BE REMOVED."
PRINT "PRESS ANY KEY TO VIEW MENU"
GET A#
GOTO 3040
REM
5820 REM List the constraints (call a subroutine to do it)
5830 REM
5840 F2 = 1: REM Set flag so O.F. won't be listed
5850 GOSUB 6610: REM list them
5860 REM
5870 REM Ask which one should be deleted
5880 REM
5890 PRINT "WHICH DO YOU WISH TO REMOVE?"
5900 PRINT "PLEASE ANSWER ";
5910 FOR I = 0 TO M - 1
5920 PRINT I; ", ";
5930 NEXT I
5940 PRINT " OR "; M
5950 PRINT "CHOOSE 0 TO RETURN TO THE MENU WITHOUT"
5960 PRINT " REMOVING ANY CONSTRAINTS"
5970 PRINT "WHICH ONE? ";ROW
5980 IF ROW = 0 THEN 3040
5990 IF ROW > = 1 AND ROW < M THEN 6050
6000 PRINT CHR$ (7); "***ERROR: INVALID RESPONSE";
6010 GOTO 5900
6020 REM
6030 REM Verify that the user really wants to delete this constr
6040 REM
6050 HOME
6060 PRINT
6070 PRINT "DO YOU REALLY WISH TO REMOVE:"
6080 PRINT
6090 BS = "+"
6100 IF A(ROW,2) < 0 THEN BS = "-
6110 PRINT A(ROW,1); "* X1 ";BS; " ";A(ROW,2); "* X2 ";A$; " ";RS(ROW)
6120 PRINT
6130 PRINT "PLEASE ANSWER Y OR N ";A$
6140 IF A$ = "Y" THEN 6210
6150 IF A$ = "N" THEN 3040
6160 PRINT CHR$ (7); "***ERROR: INVALID RESPONSE";
6170 GOTO 6130
6180 REM
6190 REM Delete it by moving all later constrs up one position
6200 REM
6210 FOR I = ROW TO M - 1
6220 A(I,0) = A(I + 1,0)
6230 A(I,1) = A(I + 1,1)
6240 A(I,2) = A(I + 1,2)
6250 RS(I) = RS(I + 1)
6260 NEXT
6270 REM
6280 REM Update number of constrs (there is now one less)
6290 REM
6300 M = M - 1
6310 REM
6320 REM Set flags to indicate that the problem has changed and
6330 REM and must be solved again (both graphically & with simplex)
6340 REM
F0 = 1: REM must solve graphically
F1 = 1: REM must solve with simplex
REM Return to main menu
GOTO 3040
REM List the constraint equations & the obj. fn.
REM Clear text screen, display text screen, home cursor
CALL - 936
TEXT
HOME
REM Call a subroutine to list them
GOSUB 6730
REM Display them until a key is pressed, then return to main menu
REM PRINT "PRESS ANY KEY TO VIEW MENU ";
GET A$
GOTO 3040: REM main menu
REM Subroutine to display the constraint equations and,
only the objective function equation. If
F1 is equal to one when this subroutine is called
the objective function is not displayed (for use
with the "D" (delete) command. F2 will always be
equal to zero after calling this subroutine.
REM Called by: "L" List constraints & O.F.
"D" Delete a constraint
"C" Change a constraint
REM THE CONSTRAINTS ARE:
FOR ROW = 1 TO M
B$ = "+
IF A(Row,2) < 0 THEN B$ = "−"
PRINT ROW; " ";
PRINT A(Row,1); " * \times 1 \cdot \cdot B$; ";
PRINT A(Row,2); " * \times 2 ";
IF NOT A(Row,0) THEN A$ = "=": GOTO 6830
IF A(Row,0) = 1 THEN A$ = ">": GOTO 6830
A$ = "<"
PRINT A$: ";RS(Row)
NEXT ROW
REM Return now if "D" (delete) called
IF F2 THEN F2 = 0: RETURN

REM Display the objective function equation
PRINT "THE OBJECTIVE FUNCTION IS:
PRINT I$; ";
X1 = - 1 * A(0,1)
X2 = - 1 * A(0,2)
PRINT X1; " * X1 + "*X2; " * X2
PRINT RETURN

REM *******************************************************
REM "C" Change a constraint
REM *******************************************************
REM Clear text page, view text page, home cursor, print title
REM CALL - 936
TEX HOME
PRINT "CHANGE A CONSTRAINT"
PRINT REM
REM List the constraints (call a subroutine to do it)
REM GOSUB 6730
PRINT REM
REM Ask which one is to be changed
REM PRINT "WHICH CONSTRAINT DO YOU WISH TO CHANGE?"
PRINT " -1 WILL RETURN YOU TO THE MENU"
PRINT " 0 WILL CHANGE THE OBJECTIVE FUNCTION"
PRINT " A NUMBER WILL CHANGE THAT CONSTRAINT"
PRINT "WHICH ONE?"
PRINT "PLEASE ANSWER ";
FOR I = -1 TO M - 1
PRINT I; ", ";
NEXT
PRINT "OR ";M; ";
INPUT ROW
REM Go do the user asked for, if it was a valid choice
REM RETURN to main menu
REM chan se objective function
REM change a constr
REM trap errors
REM PRINT
7410 PRINT CHR$(7);"***ERROR: INVALID RESPONSE"
7420 GOTO 7200
7430 REM
7440 REM ------ Change the constraint ------
7450 REM
7460 CALL - 936: REM clear the text screen
7470 PRINT "CHANGE CONSTRAINT ";ROW
7480 REM
7490 REM Call a subroutine to ask for the data for the new constr
7500 REM
7510 GOSUB 8200
7520 REM
7530 REM Set flags to indicate that the problem has changed and
7540 REM and must be solved again (both graphically & with simplex)
7550 REM
7560 F0 = 1: REM must solve graphically
7570 F1 = 1: REM must solve with simplex
7580 REM
7590 REM Return to main menu
7600 REM
7610 GOTO 3040
7620 REM
7630 REM ------ Change the objective function ------
7640 REM
7650 CALL - 936: REM clear text screen
7660 PRINT "CHANGE OBJECTIVE FUNCTION"
7670 PRINT
7680 INPUT "ENTER COEFFICIENT OF X1 ";X1
7690 PRINT
7700 INPUT "ENTER COEFFICIENT OF X2 ";X2
7710 PRINT
7720 PRINT
7730 PRINT "DO YOU WISH TO MINIMIZE OR MAXIMIZE"
7740 PRINT "THE OBJECTIVE FUNCTION?"
7750 PRINT
7760 INPUT "PLEASE TYPE MIN OR MAX: ";Z$  
7770 IF Z$ = "MIN" OR Z$ = "MAX" THEN 7800
7780 PRINT CHR$(7);"***ERROR INVALID RESPONSE"
7790 GOTO 7760
7800 Z$ = Z$ + "IMIZE"
7810 HOME
7820 PRINT "THE NEW OBJECTIVE FUNCTION IS:"
7830 PRINT
7840 PRINT Z$: SPC( 2):
7850 PRINT SPC( 5);X1; " * X1 + ";X2; " * X2"
7860 PRINT
7870 PRINT "IS THIS CORRECT? PLEASE ANSWER:"
7880 PRINT
7890 PRINT "Y - TO ACCEPT IT,"
7900 PRINT "N - TO REENTER IT OR"
7910 PRINT "R - TO RETURN TO MENU WITHOUT"
7920 PRINT "ACCEPTING IT"
7930 PRINT
7940 INPUT "PLEASE ANSWER Y, N OR R ";A$
7950 REM
7960 IF A$ = "Y" THEN 8050
7970 IF A$ = "N" THEN CALL - 936: GOTO 7670
7980 IF A$ = "R" THEN 3040
7990 PRINT
8000 PRINT CHR$ (7):"**ERROR: INVALID RESPONSE"
8010 GOTO 7940
8020 REM
8030 REM Save the data for the revised objective function
8040 REM
8050 A(0,1) = - 1 * X1
8060 A(0,2) = - 1 * X2
8070 REM
8080 REM Set flags to indicate that the problem has changed and
8090 REM and must be solved again (but only with simplex)
8100 REM
8110 F0 = 1: REM graph will not change
8120 F1 = 1: REM must solve with simplex
8130 GOTO 676
8140 REM END OF CHANGE O.F.
8150 POKE - 16304,0
8160 GOTO 680
8170 REM *********************************************
8180 REM ENTER DATA FOR A CONSTRAINT
8190 REM *********************************************
8200 INPUT "ENTER COEFFICIENT OF X1: ";X1
8210 PRINT
8220 INPUT "ENTER COEFFICIENT OF X2: ";X2
8230 PRINT
8240 INPUT "ENTER RIGHT HAND SIDE: ";X4
8250 PRINT "IS THIS A <, = OR > CONSTRAINT?"
8260 INPUT "ENTER <, = OR > ";A$
8270 IF A$ = "<" THEN X3 = 255: GOTO 8340
8280 IF A$ = ">") THEN X3 = 1: GOTO 8340
8290 IF A$ = "=" THEN X3 = 0: GOTO 8340
8300 PRINT
8310 PRINT "***ERROR: INVALID RESPONSE"
8320 INPUT "ENTER <, = OR > ";A$
8330 GOTO 8270
8340 HOME: VTAB 5
8350 PRINT "CONSTRAINT NUMBER ";ROW
8360 PRINT X1;" * X1 + ";X2;" * X2 ";A$;" ";X4
8370 PRINT "IS THIS CORRECT? " : PRINT
8380 PRINT "TYPE: Y-TO ACCEPT THE NEW CONSTRAINT,
8390 PRINT " N-TO REENTER THIS CONSTRAINT, OR"
8400 PRINT " R-TO REJECT THE NEW CONSTRAINT"
8410 PRINT SPC( 11);"AND RETURN TO THE MENU"
8420 INPUT A$
8430 IF A$ = "Y" THEN 8520
8440 IF A$ = "N" THEN HOME: VTAB 5: GOTO 8170
8450 IF A$ = "R" THEN POP : GOTO 3040: REM MENU
8460 REM TRAP ERRORS
8470 INPUT "INVALID ANSWER; PLEASE TYPE Y, N OR ESC "; A$
8480 GOTO 8430: REM OK NOW?
8490 REM
8500 REM Accept the new constraint
8510 REM
8520 A(ROW,1) = X1
8530 A(ROW,2) = X2
8540 A(ROW,0) = X3
8550 RS(ROW) = X4
8560 RETURN
8570 REM ****************************************************************************
8580 REM DRAW TIC MARKS ON AXES
8590 REM ****************************************************************************
8600 REM
8610 REM Save contents of memory location 230 for later restoration
8620 REM
8630 TP = PEEK (230)
8640 POKE 230,P6
8650 IF P6 = 32 THEN PRINT CHR$ (15); CHR$ (1):
8660 H PLOT HM,VM TO HM,191 - VM TO 279 - HM,191 - VM
8670 SP = LOG (X0(0)) / LOG (10)
8680 SQ = SP - (INT (SP))
8690 IF SQ < 0.6 THEN SP = SP - 1
8700 SP = 10 ^ (INT (SP))
8710 REM DO Y-AXIS
8720 H PLOT HM - 5,191 - VM TO HM,191 - VM
8730 TIC = 0
8740 TIC = TIC + SP
8750 IF TIC > Y0(0) THEN 8810
8760 Y = VM + YR * (Y0(0) - TIC) / Y0(0)
8770 D1 = TIC
8780 H PLOT HM - 5,Y TO HM,Y
8790 GOTO 8740: REM do next tic
8800 REM Place tic at top
8810 H PLOT HM - 5,VM TO HM,VM
8820 REM
8830 REM Now do X-axis
8840 REM
8850 H PLOT HM,191 - VM TO HM,191 - VM
8860 Y = VM + YR * (Y0(0) - D1) / Y0(0)
8870 H PLOT HM - 5,Y TO HM - 5,Y
8880 PRINT CHR$ (17); 
8890 VTAB 3
8900 PRINT D1
8910 VTAB 22
8920 PRINT "0";
8930 TIC = 0
8940 TIC = TIC + SP
8950 IF TIC > X0(0) THEN 9020
8960 X = HM + XR * TIC / X0(0)
8970 H PLOT X,196 - VM TO X,191 - VM
8980 GOTO 8940
8990 REM
9000 REM Plot tic at right end
9010 REM
9020 HPLOT 279 - HM,196 - VM TO 279 - HM,191 - VM
9030 POKE 230,TP
9040 IF PG = 32 THEN PRINT CHR$(15); CHR$(2);
9050 RETURN
9060 REM ************************************************************
9070 REM SIMPLEX MODULE
9080 REM
9090 REM Note: Lines 9150 to 10670 adapted from Some Common
9100 REM Basic Programs, By Lon Poole and Mary Borchers.
9120 REM
9130 REM
9140 REM ************************************************************
9150 F4 = 1: REM ASSUME SOLN EXISTS
9160 L = 0: REM NUMBER OF <'S
9170 E = 0: REM NUMBER OF ='S
9180 G = 0: REM NUMBER OF >'S
9190 REM COUNT <'S, ='S AND >'S
9200 FOR ROW = 1 TO M
9210 IF A(ROW,0) = 255 THEN L = L + 1: GOTO 9240
9220 IF NOT A(ROW,0) THEN E = E + 1: GOTO 9240
9230 G = G + 1
9240 NEXT ROW
9250 CT = N * M + G
9260 CS = CT + 1
9270 CU = N * L + G
9280 M1 = M + 1
9290 M2 = M + 2
9300 REM ZERO OUT THE ARRAYS
9310 FOR I = 0 TO 2
9320 SOLN(I) = 0
9330 NEXT I
9340 FOR I = 1 TO M2
9350 FOR J = 1 TO CS
9360 AS(I,J) = 0
9370 NEXT J
9380 NEXT I
9390 FOR I = 1 TO M
9400 B(I) = 0
9410 NEXT I
9420 I = 1: REM INDEX FOR AS()
9430 REM COPY <'S TO AS()
9440 FOR ROW = 1 TO M
9450 IF A(ROW,0) < 255 THEN 9500
9460 AS(I,1) = A(ROW,1)
9470 AS(I,2) = A(ROW,2)
9480 AS(I,CS) = RS(ROW)
9490 I = I + 1
9500 NEXT ROW
9510 REM COPY ='S TO AS()
9520 FOR ROW = 1 TO M
IF A(ROW,0) > 0 THEN 9580
9540 AS(I,1) = A(ROW,1)
9550 AS(I,2) = A(ROW,2)
9560 AS(I,CS) = RS(ROW)
9570 I = I + 1
9580 NEXT ROW
9590 REM COPY >'S TO AS()
9600 FOR ROW = 1 TO M
9610 IF A(ROW,0) > 1 THEN 9660
9620 AS(I,1) = A(ROW,1)
9630 AS(I,2) = A(ROW,2)
9640 AS(I,CS) = RS(ROW)
9650 I = I + 1
9660 NEXT ROW
9670 REM CONVERT MIN/MAX TO NUMBER
9680 ZS = 1
9690 IF I$ = "MAXIMIZE" THEN ZS = 1
9700 CT = N + M + G
9710 CS = CT + I
9720 CU = N + L + G
9730 FOR I = 1 TO M
9740 FOR J = 1 TO N
9750 IF I < L THEN 9770
9760 AS(M1,J) = AS(M1,J) - AS(I,J)
9770 NEXT J
9780 IF I > L THEN 9820
9790 B(I) = N + I
9800 AS(I,N + I) = 1
9810 GOTO 9880
9820 B(I) = N + G + I
9830 AS(I,N + G + I) = 1
9840 IF I > L + E THEN 9860
9850 GOTO 9880
9860 AS(I,N + I - E) = -1
9870 AS(M1,N + I - E) = 1
9880 NEXT I
9890 REM COPY Q.F. TO AS()
9900 AS(M2,1) = A(0,1)
9910 AS(M2,2) = A(0,2)
9920 IF L = M THEN 10090
9930 M3 = M1
9940 GOSUB 10220
9950 FOR II = 1 TO M
9960 IF B(II) = CU THEN 10080
9970 IF AS(II,CS) = 0.00001 THEN 10010
9980 F4 = 0: REM NO FEAS SOLN
9990 TEXT : PRINT "NO FEAS SOLN": STOP
10000 GOTO 10070
10010 FOR J1 = 1 TO CU
10020 IF ABS(AS(II,J1)) < 0.00001 THEN 10070
10030 R = II
10040 S = J1
10050 GOSUB 10470
10060 J1 = CU
10070 NEXT J1
10080 NEXT I1
10090 M3 = M2
10100 GOSUB 10220
10110 FOR J = 1 TO CU
10120 FOR I = 1 TO M
10130 IF B(I) < J THEN 10160
10140 SOLN(I) = AS(1,CS)
10150 I = M
10160 NEXT I
10170 NEXT J
10180 SOLN(0) = ZS * AS(M2,CS)
10190 RETURN : REM FROM SIMPLEX
10200 REM - OPT ROUTINE
10210 REM - FIRST PRICE OUT COLUMNS
10220 P = -0.00001
10230 FOR J = 1 TO CU
10240 IF AS(M3,J) = P THEN 10270
10250 S = J
10260 P = AS(M3,J)
10270 NEXT J
10280 IF P = -0.00001 THEN 10650
10290 GOSUB 10330
10300 GOSUB 10410
10310 GOTO 10220
10320 REM - FIND LEAVE BASIS
10330 Q = 1.E + 38
10340 FOR I = 1 TO M
10350 IF AS(I,S) < = 0.00001 THEN 10390
10360 IF AS(I,CS) / AS(I,S) = > Q THEN 10390
10370 R = I
10380 Q = AS(I,CS) / AS(I,S)
10390 NEXT I
10400 RETURN
10410 IF Q = 1.E + 38 THEN 10440
10420 GOSUB 10470
10430 RETURN
10440 F4 = 0: REM UNBOUNDED
10450 RETURN : REM FROM SIMPLEX
10460 REM - PIVOTING
10470 P = AS(R,S)
10480 FOR I = 1 TO M2
10490 IF I = R THEN 10560
10500 FOR J = 1 TO CS
10510 IF J = S THEN 10550
10520 AS(I,J) = AS(I,J) - AS(I,S) * AS(R,J) / P
10530 IF ABS(AS(I,J)) > = 0.00001 THEN 10550
10540 AS(I,J) = 0
10550 NEXT J
10560 NEXT I
10570 FOR J = 1 TO CS
10580 AS(R,J) = AS(R,J) / P
10590 NEXT J
10600 FOR I = 1 TO M2
10610 AS(I,S) = 0
10620 NEXT I
10630 AS(R,S) = 1
10640 B(R) = S
10650 RETURN
10660 REM RETURN TO M/PROG
10670 RETURN
Appendix C

ASSEMBLY LANGUAGE PROGRAM LISTINGS
-- NEXT OBJECT FILE NAME IS LINGRAF.ASSY.OBJ0
3  ORG $9300
4  **********************
5  LINGRAF ASSEMBLY LANGUAGE ROUTINES
6  (C) 1984 by Stanley G. Bevis
7  *
8  **********************
9
10  I/O addresses
11
12  KBD EQU $C000 ; keyboard register
13  KBDSTRB EQU $C010 ; keyboard strobe
14
15  **********************
16  *
17  HIERs Util ity Routines
18  *
19  *
20  *
21  *
22  *
23  *
24  *
25  *
26  *
27  *
28  *
29  *
30  *
31  *
32  *
33  *
34  *
35  *
36  *
37  *
38  *
39  *
40  *
41  *
42  *
43  *
44  *
45  *
46  *
47  *
48  *
49  *
50  *
51  *
52  *
53  *

9300: 3 ORG $9300
9300: 4 **********************
9300: 5 *
9300: 6 LINGRAF ASSEMBLY LANGUAGE ROUTINES
9300: 7 *
9300: 8 (C) 1984 by Stanley G. Bevis
9300: 9 *
9300: 10 **********************
9300: 11 ;
9300: 12 ;
9300: 13 : I/O addresses
9300: 14 :
9300: 15 KBD EQU $C000 ; keyboard register
9300: 16 KBDSTRB EQU $C010 ; keyboard strobe
9300: 17 :
9300: 18 **********************
9300: 19 *
9300: 20 HIERs UTILITy ROUTINES
9300: 21 1) MOVE: Copy one HIERs page to the other page.
9300: 22 *
9300: 23 2) ANDHGR: AND one HIERs page with the other page.
9300: 24 *
9300: 25 3) BLACK: Fill a HIERs page with black.
9300: 26 *
9300: 27 ON ENTRY:
9300: 28 TEMPl contains the HIERs "destination" page number. The other page is *
9300: 29 the "source". BLACK uses only the "destination" page.
9300: 30 *
9300: 31 *
9300: 32 *
9300: 33 Called by: BASIC program
9300: 34 *
9300: 35 (C) by Stan Bevis 4/10/83
9300: 36 *
9300: 37 **********************
9300: 38 :
9300: 39 : Zero-page equates
9300: 40 :
9300: 41 SRCADR EQU $FC ; 2-byte base addr (source)
9300: 42 DSTADR EQU $FE ; 2-byte base addr (dest)
9300: 43 :
9300: 44 ; General variable storage
9300: 45 :
9300: 46 TEMPl DFB 0 ; several uses
9300: 47 :
9300: 48 :
9300: 49 ; MOVE: copies source HIERs page to destination page
9300: 50 :
9300: 51 :
9300: 52 MOVE JSR SETUP
9300: 53 LDY #0 ; use as offset
LINGRAF.ASSY  LINGRAF Assembly Language Routines

9308:B1 FC 54 NEXT1 LDA (SRCADR),Y ;move a byte
9308:91 FE 55 STA (DSTADR),Y
930A:C8 56 INY
930B:D0 F9 57 BNE NEXT1 ;if not end of mem page
930D:20 5C 93 58 JSR ENCHK
9310:90 F4 59 BCC NEXT1
9312:60 60 RTS
9313: 61 ;
9313: 62 ;
9313: 63 ;AND: AND's the 2 HIRES pages - puts the result
9313: 64 ;in the "destination" page
9313: 65 ;
9313: 66 ;
9313:20 39 93 67 ANDHR JSR SETUP
9316:A0 00 68 LDY #0 ;use as offset
9318:B1 FC 69 NEXT2 LDA (SRCADR),Y ;AND 2 bytes
931A:91 FE 70 AND (DSTADR),Y
931C:91 FE 71 STA (DSTADR),Y ;update dest page byte
931E:C8 72 INY
931F:D0 F7 73 BNE NEXT2 ;if not end of mem page
9321:20 5C 93 74 JSR ENCHK ;done yet?
9324:90 F2 75 BCC NEXT2
9326:60 76 RTS
9327: 77 ;
9327: 78 ;
9327: 79 ;BLACK: Fills "destination" HIRES page with black
9327: 80 ;
9327: 81 ;
9327:20 39 93 82 BLACK JSR SETUP ;get addresses
932A:A0 00 83 LDY #0 ;use as offset
932C:A9 00 84 NEXT3 LDA #0 ;for 7 black dots
932E:91 FE 85 NEXT4 STA (DSTADR),Y ;plot 7 black dots
9330:C8 86 INY
9331:D0 FB 87 BNE NEXT4
9333:20 5C 93 88 JSR ENCHK ;done yet?
9336:90 F4 89 BCC NEXT3
9338:60 90 RTS
LINGRAF.ASSY  LINGRAF Assembly Language Routines

9339:  92 ;  SETUP & ENDCMK
9339:  93 ;
9339:  94 ; Subroutines called by COPY, ANDHGR & BLACK
9339:  95 ;
9339:  96 ;
9339:  97 ; SETUP - Determines the starting addresses of the
9339:  98 ; source & destination HIRES pages. The
9339:  99 ; addresses are stores in SRCADR,
9339: 100 ; SRCADR+1, and DSTADR, DSTADR+1
9339: 101 ;
9339: 102 ;
9339:A9 00 103 SETUP LDA #0 ; LOB (both pages)
9339:B5 FC 104 STA SRCADR
9339:B5 FE 105 STA DSTADR
9339:A9 20 106 LDA ##20 ; for HIRES page 1
9341:CE 00 93 107 DEC TEMP1 ; dest HIRES page no
9344:DO 0B 108 BNE PG2 ; HIRES page 2 is dest
9346:BF FF 109 STA DSTADR+1 ; HOB of dest
9348:0A 110 ASL A ; HOB of source
9349:BF FD 111 STA SRCADR+1
934B:04 FD 112 LDA #40 ; where to stop
9350:BD 00 93 113 STA TEMP1
9350:60 114 RTS
9351:BD 00 93 115 PG2 STA SRCADR+1 ; HOB of source
9353:0A 116 ASL A ; HOB of dest
9354:BF FF 117 STA DSTADR+1
9356:A9 60 118 LDA ##60 ; where to stop
9358:BD 00 93 119 STA TEMP1
935B:60 120 RTS
935C:  121 ;
935C:  122 ;
935C:  123 ;
935C:  124 ; ENDCMK - Checks to see if caller is done; if not,
935C:  125 ; base addr's are updated.
935C:  126 ;
935C:  127 ; If caller is done, carry flag is set.
935C:  128 ;
935C:E6 FF 129 ENDCMK INC DSTADR+1
935E:AD 00 93 130 LDA TEMP1
9361:CF FF 131 CMP DSTADR+1
9363:1B 132 CLC
9364:DO 01 133 BNE MORE ; caller is not done
9366:3B 134 SEC ; caller is done
9367:E6 FD 135 MORE INC SRCADR+1
9369:60 136 RTS
LINGRAF.ASSY

Lingraf Assembly Language Routines

936A: 138 ****************************
936A: 139 *
936A: 140 * KEYBOARD SCAN & DECODE *
936A: 141 *
936A: 142 * 1) Waits for a keypress *
936A: 143 * 2) Ignores all keys other than left & right *
936A: 144 * arrows, greater & less than signs, *
936A: 145 * ESCape and the digits (0-9), *
936A: 146 * 3) Two values are returned after a valid *
936A: 147 * key has been pressed. They identify *
936A: 148 * the key and are stored in ZN & ZP: *
936A: 149 *
936A: 150 * Key ZN ZP *
936A: 151 * left arrow 1 0 *
936A: 152 * right arrow 0 1 *
936A: 153 * less than 5 0 *
936A: 154 * greater than 0 5 *
936A: 155 * ESCape 0 0 *
936A: 156 * digit N N+10 0 *
936A: 157 *
936A: 158 * Called by: BASIC animation routine *
936A: 159 *
936A: 160 ****************************
936A: 161 ;
936B: 00 162 ZN DFB 0 ;move ZN pixels left
936B: 00 163 ZP DFB 0 ;move ZP pixels right
936C: 164 ;
936C: 165 ;
936C: AD 00 C0 166 DECODE LDA KBD ;wait for keypress
936F: 10 FB 167 BPL DECODE
9371: 2C 10 C0 168 BIT KBDSTRB ;clear strobe
9374: 169 ;
9374: 170 ;Initialize variables
9374: 171 ;
9374: A2 00 172 LDX #0
9376: 8E 6B 93 173 STX ZP
9379: 8E 6A 93 174 STX ZN
937C: 175 ;
937C: 176 ;Decode the key
937C: 177 ;
937C: C9 8B 178 CMP #$8B ;left arrow?
937E: D0 06 179 BNE RTARW
9380: A9 01 180 LDA #1
9382: BD 6A 93 181 STA ZN
9385: 60 182 RTS
9386: 183 ;
9386: C9 95 184 RTARW CMP #$95 ;right arrow?
9388: D0 06 185 BNE LTKEY
938A: A9 01 186 LDA #1
938C: BD 6B 93 187 STA ZP
938F: 60 188 RTS
9390: 189 ;
LINGRAF.ASSY
LINGRAF Assembly Language Routines

9390: C9 BC 190 LTKEY CMP #$BC ; "<"?
9392: D0 06 191 BNE GTKEY
9394: A9 05 192 LDA #$5
9396: BD 6A 93 193 STA ZN
9399: 60 194 RTS
939A: 195 ;
939A: C9 BE 196 GTKEY CMP #$BE ; "\">
939C: D0 06 197 BNE ESC
939E: A9 05 198 LDA #$5
93A0: BD 6B 93 199 STA ZP
93A3: 60 200 RTS
93A4: 201 ;
93A4: C9 9B 202 ESC CMP #$9B ; Escape key?
93A6: D0 01 203 BNE NUMS ; no - try digits
93A8: 60 204 RTS ; leave ZN & ZP = 0
93A9: 205 ;
93A9: 206 ; Is it a number (0-9)?
93A9: 207 ;
93A9: C9 B1 208 NUMS CMP #$B1 ; < "1"?
93AB: 30 BF 209 BMI DECODE ; invalid key
93AD: C9 BA 210 CMP #$BA ; > "9"?
93AF: 10 BB 211 BPL DECODE ; invalid key
93B1: 212 ; It is a number; now find out which one
93B1: 213 ; Put N+10 in ZN (N is the digit)
93B1: 214 ;
93B1: 38 215 SEC
93B2: E9 A6 216 SBC #$A6
93B4: BD 6A 93 217 STA ZN
93B7: 60 218 RTS
LINGRAF.ASSY  LINGRAF Assembly Language Routines

93BB:  220 **********************************************************************
93BB:  221 *
93BB:  222 * KEYPR - Unconditionally puts the contents of *
93BB:  223 * $C000 in IP and clears the keyboard *
93BB:  224 * strobe. It does not check to see if *
93BB:  225 * any key was pressed. *
93BB:  226 *
93BB:  227 * Called by: BASIC program *
93BB:  228 *
93BB:  229 **********************************************************************
93BB:  230 ;
93BB:  231 ;
93BB:AD 00 C0 232 KEYPR LDA KBD ;read keyboard
93BB:BD 00 93 233 STA TEMFL ;pass it to caller
93BB:2C 10 C0 234 BIT KBDSTRB ;clear strobe
93C1:60 235 RTS
**LINE SCAN ROUTINE**

Finds the first point plotted to the right of X=21 on a given horizontal line on HIRES.

**On entry:** Y-cord of line in YPOSN

**On exit:** X-coord of 1st point in XL, XH

---

```
255 LINE JSR BASCALC  ;get base addr
257 ;Find the first plotted point (right of X=21)
259 ;start @ X=21
260 STA XL  ;X-coord LOB
261 LDA #0  ;for X=21
262 STA XH  ;X-coord HOB
263 LDY #3  ;for next byte
264 ;Routine to find 1st non-zero byte
266 ;
267 CHECK LDA (BASE),Y  ;get a byte
268 BNE FOUND  ;a point is plotted
269 ;
270 ;Get set for next byte
271 ;
272 INY  ;point to next byte
273 CPY #40  ;gone too far?
274 BEQ NODOT  ;no points on this line
275 LDA #7  ;update X-position
276 CLC
277 ADC XL
278 STA XL
279 LDA #0
280 ADC XH
281 STA XH
282 JMP CHECK  ;go do next byte
```

---

```
283 ;
284 ;Come here when non-zero byte has been found.
285 ;Determine which point it is.
286 ;
287 FOUND LDX #0  ;count bits
288 CHKBIT ROR A
```
LINGGRAF.ASY      LINGGRAF Assembly Language Routines

93F3:BO  04  289  BCS  BITSET ;found 1st point
93F5:EB   290  INX
93F6:4C F2 93  291  JMP  CHKBIT ;look @ next bit
93F9:     292 ;
93F9:8A   293  BITSET  TXA           ;adjust X-posn counter
93FA:18   294  CLC
93FB:6D C2 93  295  ADC  XL
93FE:BD C2 93  296  STA  XL
9401:A9 00  297  LDA  #0
9403:6D C3 93  298  ADC  XH
9406:BD C3 93  299  STA  XH
9409:60   300  NODOT  RTS
**LINGRAF. Assy**  
**LINGRAF Assembly Language Routines**

940A: 302 ;
940A: 303 ******************************
940A: 304 *
940A: 305 * BASCALC: Finds the address of the first *
940A: 306 * of 40 bytes that represent a *
940A: 307 * horizontal line in hires page 2. *
940A: 308 *
940A: 309 * The following equation is used (Y = y-coord): *
940A: 310 *
940A: 311 * B = 1024(Y mod 8) + 128(int(Y/8)mod 8) *
940A: 312 * + 40(int(Y/64) + B192 *
940A: 313 *
940A: 314 * where Y = y-coordinate of the line *
940A: 315 * B = The base address for line Y *
940A: 316 *
940A: 317 * Called by: LINE (Assembly) *
940A: 318 *
940A: 319 * On entry: Y-coordinate of line in TEMPI *
940A: 320 *
940A: 321 * On exit: Base address for line in SUML,H *
940A: 322 *
940A: 323 ******************************
940A: 324 ;
940A:00 325 YPOSN DFB 0
940B:00 326 TEMPL DFB 0
940C:00 327 TEMPH DFB 0
00FE: 328 SUML EQU $FE
00FF: 329 SUMH EQU $FF
940D: 330 ;
940D: 331 ;Note: In the following comments, Y refers
940D: 332 ; to the line's y-coordinate, and
940D: 333 ; not the Y index register.
940D: 334 ;
940D:A9 00 335 BASCALC LDA #0
940F:85 FE 336 STA SUML
9411:85 FF 337 STA SUMH
9413: 338 ;
9413: 339 ;Put Y MOD 8 in TEMP and INT(Y/8) in TEMPI
9413: 340 ;
9413:AD 0A 94 341 LDA YPOSN
9416:20 B5 94 342 JSR DIV8
9419:8E 00 93 343 STX TEMPI ;save INT(Y/8)
941C:BD 0B 94 344 STA TEMPL ;save Y MOD 8
941F:A9 00 345 LDA #0
9421:BD 0C 94 346 STA TEMPH
9424: 347 ;
9424: 348 ;Multiply TEMPL,TEMPH by 1024
9424: 349 ;
9424:A2 0A 350 LDX #10 ;for 2^10
9426:20 7A 94 351 JSR EXP2
9429:AD 0B 94 352 LDA TEMPL ;save result in SUM
942C:85 FE 353 STA SUML
LINGRAF.ASSY  LINGRAF Assembly Language Routines

442E: AD 0C 94 354 LDA TEMPL
4431: 85 FF 355 STA SUMH
4433: 356 
4433: AD 00 93 357 ;Put Y/8 MOD 8 in TEMPL,TEMPH and Y/64 in TEMP
4433: 358 
4433: AD 00 93 359 LDA TEMP1 ;get Y/8
4436: 20 85 94 360 JSR DIVB
4439: BE 00 93 361 STX TEMPI ;save Y/64
443C: BD 0B 94 362 STA TEMPL ;save Y/8 MOD 8 in TEMP
443F: A9 00 363 LDA #0
4441: BD 0C 94 364 STA TEMPL
4444: 365 
4444: 366 ;Multiply TEMPL,TEMPH by 128
4444: 367 
4444: A2 07 368 LDY #7 ;for 128=2^7
4446: 20 7A 94 369 JSR EXF
4449: 370 
4449: 371 ;Add TEMPL,TEMPH to SUML,SUMH
4449: 372 
4449: 18 373 CLC
444A: AD 0E 94 374 LDA TEMPL
444D: 65 FE 375 ADC SUML
444F: 85 FE 376 STA SUML
4451: AD 0C 94 377 LDA TEMPL
4454: 65 FF 378 ADC SUMH
4456: 85 FF 379 STA SUMH
4458: 380 
4458: 381 ;Put Y/64 in A (always 8 bits)
4458: 382 
445B: AD 00 93 383 LDA TEMP1 ;get Y/8
445E: 0A 384 ASL A
445F: 0A 385 ASL A
4460: 0A 386 ASL A
446E: BD 00 93 387 STA TEMPI ;save INT(Y/64) * 8
446F: 0A 388 ASL A
4471: 0A 389 ASL A ;now have INT(Y/64) * 32
4473: 1B 390 CLC
4474: 6D 00 93 391 ADC TEMPI ;gives INT(Y/64) * 40
4476: 392 
4477: 393 ;Add (A) to SUML,SUMH
4477: 394 
4477: 18 395 CLC
447B: 65 FE 396 ADC SUML
447A: 85 FE 397 STA SUML
447C: A9 00 398 LDA #0
447E: 65 FF 399 ADC SUMH
447F: 85 FF 400 STA SUMH
447E: 20 392 
447E: 402 ;Add 8192 to SUML, SUMH
447E: 403 
447E: 1B 404 CLC
447F: A9 20 405 LDA #$20 ;HOB of 8192
LINGRAF.ASSY  LINGRAF Assembly Language Routines

9475:65 FF  406  ADC  SUMH
9477:85 FF  407  STA  SUMH
9479:   408   ;
9479:   409  ;Base addr for Y-posn is now in SUML, SUMH
9479:   410   ;
9479:60  411  RTS
LINGRAF.ASSY

LINGRAF Assembly Language Routines

947A: 413 Subroutines called by BASCALC
947A: 414
947A: 415
947A: 416 Routine to raise (A) to 2^(X)
947A: 417
947A: 418
947A: 18 419 EXP2 CLC
947B: 2E 0B 94 420 ROL TEMPL
947E: 2E 0C 94 421 ROL TEMPH
9481: CA 422 DEX
9482: D0 F6 423 BNE EXP2
9484: 60 424 RTS
9485: 425
9485: 426
9485: 427 Routine to divide (A) by 8
9485: 428 leaves quotient in X and remainder in A
9485: A2 00 429 DIVB LDX #0
9487: 38 430 SUBT SEC ; subtract out 8's
9488: E9 08 431 SBC #8
948A: 90 04 432 BCC MOD ; went one too many
948C: EB 433 INX ; count 8's
948D: 4C 87 94 434 JMP SUBT
9490: 18 435 MOD CLC ; now do remainder
9491: 69 08 436 ADC #8 ; go back one 8
9493: 60 437 RTS
439 ************ LINGRAF Assembly Language Routines

440 * KEY2: Keyboard decoder for main menu

442 *

443 * On entry: ASCII code for key that was

444 * pressed is in TEMP1

445 *

446 * On exit: Code (1..7) for key is in TEMP1.

447 * The codes are as follows:

448 * 1 - A

449 * 2 - C

450 * 3 - D

451 * 4 - L

452 * 5 - R

453 * 6 - O

454 * 7 = Any other key

455 *

456 * Called by: BASIC program

457 *

458 ************

459 :

460 KEY2 LDA TEMP1 ; the character

461 LDX #1

462 ;

463 ; Which key was pressed?

464 ;

465 CMP #C1 ; "A"?

466 BNE CKEY

467 BEQ EXIT

468 ;

469 CKEY INX

470 CMP #C3 ; "C"?

471 BNE DKEY

472 BEQ EXIT

473 ;

474 DKEY INX

475 CMP #C4 ; "D"?

476 BNE LKEY

477 BEQ EXIT

478 ;

479 LKEY INX

480 CMP #CC ; "L"?

481 BNE RKEY

482 BEQ EXIT

483 ;

484 RKEY INX

485 CMP #D2 ; "R"?

486 BNE QKEY

487 BEQ EXIT

488 ;

489 QKEY INX

490 CMP #D1 ; "Q"?
LINGRAF.ASSY  LINGRAF Assembly Language Routines

94BE:00 02  491    BNE  ERR1
94C0:FO 01  492    BEQ  EXIT
94C2:    493    :
94C2:    494 ;Key pressed was not a valid choice; report
94C2:    495 ;the error to the calling program, then fall
94C2:    496 ;to EXIT.
94C2:    497    :
94C2:EB  498  ERR1  INX
94C3:    499    :
94C3:    500 ;Come here when the key has been identified
94C3:    501    :
94C3:BE 00 93  502  EXIT  STX  TEMP1
94C6:60  503    RTS

*** SUCCESSFUL ASSEMBLY: NO ERRORS
### SYMBOL TABLE SORTED BY SYMBOL

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>9313</td>
<td>ANDHGR</td>
</tr>
<tr>
<td>9327</td>
<td>BLACK</td>
</tr>
<tr>
<td>936C</td>
<td>DECODE</td>
</tr>
<tr>
<td>935C</td>
<td>ENDCHEK</td>
</tr>
<tr>
<td>947A</td>
<td>EXP2</td>
</tr>
<tr>
<td>C000</td>
<td>KBD</td>
</tr>
<tr>
<td>949D</td>
<td>LKEY</td>
</tr>
<tr>
<td>9301</td>
<td>MOVE</td>
</tr>
<tr>
<td>932E</td>
<td>NEXT4</td>
</tr>
<tr>
<td>94BB</td>
<td>QKEY</td>
</tr>
<tr>
<td>FC</td>
<td>SRCADR</td>
</tr>
<tr>
<td>9300</td>
<td>TEMP1</td>
</tr>
<tr>
<td>93C2</td>
<td>XL</td>
</tr>
<tr>
<td>940D</td>
<td>BASECALC</td>
</tr>
<tr>
<td>93F2</td>
<td>CHKBIT</td>
</tr>
<tr>
<td>94A6</td>
<td>DKEY</td>
</tr>
<tr>
<td>93A4</td>
<td>ESC</td>
</tr>
<tr>
<td>93F0</td>
<td>FOUND</td>
</tr>
<tr>
<td>93B8</td>
<td>RKEY</td>
</tr>
<tr>
<td>9487</td>
<td>SUBT</td>
</tr>
<tr>
<td>93C3</td>
<td>EXIT</td>
</tr>
</tbody>
</table>

### SYMBOL TABLE SORTED BY ADDRESS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC</td>
<td>SRCADR</td>
</tr>
<tr>
<td>FF</td>
<td>SUMH</td>
</tr>
<tr>
<td>9313</td>
<td>ANDHGR</td>
</tr>
<tr>
<td>932E</td>
<td>NEXT4</td>
</tr>
<tr>
<td>9367</td>
<td>MORE</td>
</tr>
<tr>
<td>93B6</td>
<td>RTARW</td>
</tr>
<tr>
<td>93A9</td>
<td>NUMS</td>
</tr>
<tr>
<td>93C4</td>
<td>LINE</td>
</tr>
<tr>
<td>93F9</td>
<td>BITSET</td>
</tr>
<tr>
<td>940C</td>
<td>TEMPH</td>
</tr>
<tr>
<td>94B7</td>
<td>SUBT</td>
</tr>
<tr>
<td>94A6</td>
<td>DKEY</td>
</tr>
<tr>
<td>94C2</td>
<td>ERR1</td>
</tr>
</tbody>
</table>

### SYMBOL TABLE SORTED BY ADDRESS (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF</td>
<td>SUMH</td>
</tr>
<tr>
<td>9313</td>
<td>ANDHGR</td>
</tr>
<tr>
<td>932E</td>
<td>NEXT4</td>
</tr>
<tr>
<td>9367</td>
<td>MORE</td>
</tr>
<tr>
<td>93B6</td>
<td>RTARW</td>
</tr>
<tr>
<td>93A9</td>
<td>NUMS</td>
</tr>
<tr>
<td>93C4</td>
<td>LINE</td>
</tr>
<tr>
<td>93F9</td>
<td>BITSET</td>
</tr>
<tr>
<td>940C</td>
<td>TEMPH</td>
</tr>
<tr>
<td>94B7</td>
<td>SUBT</td>
</tr>
<tr>
<td>94A6</td>
<td>DKEY</td>
</tr>
<tr>
<td>94C2</td>
<td>ERR1</td>
</tr>
</tbody>
</table>
Appendix D

BASIC FLOW CHARTS
START

CLEAR SCREEN, DISPLAY TITLES, ETC.

ASK HOW MANY CONSTRAINTS AND READ

PRINT ERROR MESSAGE

VALID ANSWER?

START OF LOOP FOR EACH CONSTRAINT

READ EACH DEC. VRBL (x1, x2) COEFF.

READ TYPE OF CONSTR (<, =, OR >)

A
READ RHS OF CONSTR

END OF LOOP FOR EACH CONSTR

INIT ADDRESS CONSTS AND DISPLAY HIPER PAGE 2

NEED SIMPLEX SOLU Y

SOLVE PROBLEM WITH SIMPLEX METHOD

N

SELECT H.R.C.G. AS OUTPUT DRIVER

DRAW AXES

B
B

Determine Horiz & Vert Scaling

Start of Loop for Each Constr.

Clear Hires Page 1 to Black (Call Black)

Draw Axes, Tics & Labels on Pg 1

Find Where the Coord Crosses Each Axis

Draw the Constr. On Both Pgs.

C
C

TELL USER WHICH CONSTR IS BEING DONE

VIEW HIRES PAGE 2

SAVE THIS CONSTR LINE'S END POINTS

SHADE THIS CONSTR'S FEAS REGION ON PAGE 1

DRAW IN ALL OLD CONSTR LINES ON PAGE 1

FIRST CONSTR?

Y N

D E
D

COPY PAGE ONE TO PAGE TWO

END OF LOOP FOR EACH CONST.

PRINT COMMAND PROMPT AT TOP OF SCR

FIND SLOPE OF OBJ. FN. LINE

FIND WHERE OPTIMAL OBJ. FN. CROSSES X-AXIS

PRINT OPTIMAL OBJ. FN. VALUE AND DRAW THE LINE

E

AND PG 1 WITH PG 2, SAVE RESULT ON PAGE 2

F
CALL ASSY
SUBROUTINE
DECODE TO
GET NEXT CMD

ESC KEY

NUMBER KEY

FIND WHERE
TO PLOT
NEW OBJ,
FN. LINE

VIEW HIRES
PAGE 2

BEAR 3
TIMES
AS ERROR
MESSAGE

VIEW HIRES
PAGE 2

OFF SCREEN?

COPY PG 2
TO PG 1
(CALL ASSY
SUBR. MOVE)

F

G
G

DRAW NEW OBJ. FN. LINE ON HIRES PAGE 1

VIEW HIRES PAGE 1

F
1. **DETERMINE WHICH CONSTR. TO FLASH**

2. **TELL USER WHICH CONSTR. IS FLASHING (GIVE NUMBER)**

3. **DRAW THE CONSTR IN BLACK AND DELAY**

4. **DRAW THE CONSTR IN WHITE AND DELAY**

5. **CALL ASS'Y KEY PR SUBROUTINE**

6. **ANY KEY PRESSED?**

   - **N**
     - **RESTORE PROMPT AT TOP OF SCR**
   - **Y**
I N C R E M E N T C O N S T R A I N C O U N T E R

H A V E U S E R E N T E R D A T A F O R T H I S C O N S T R


H
That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That

---

Use

That
CLEAR TEXT SCREEN

IS THERE >1 CONSTR

DISPLAY ALL OF THE CONSTRS.

PRINT ERROR MESSAGE

WAIT FOR KEYPRESS

H

READ NO. OF THE CONSTR. TO REMOVE

VERIFY THAT THE USER WANTS TO DO IT

MOVE ALL LATER CONSTRS. UP ONE

SET FLAGS TO SOLVE REVISED PROBLEM

UPDATE CONSTR COUNTER

H
P

CLEAR TEXT DISPLAY

LIST THE CONSTRS.

WAIT FOR KEYPRESS

H

Q

SET HRCE AS OUTPUT DRIVER

Y

NEED TO SOLVE PROB.

B

N

F
R

CLEAR TEXT SCREEN

SET I/O HOOKS FOR DOS

END
Appendix E

ASSEMBLY LANGUAGE FLOW CHARTS
MOVE

CALL SETUP

Y-REG=0
INIT. INDEX

LOAD ACC. WITH BYTE FROM SOURCE

SAVE ACC. IN DEST. HIRES PG.

INCREMENT INDEX (Y-REG)

(Y)=0?

CALL ENDCHECK

C-FLAG CLEAR?

RETURN
ANDHGR

CALL SETUP

Y-REG = 0 (INIT. INDEX)

B

CALL ENDCHK

C-FLAG CLEAR?

C

LOAD ACC. WITH SOURCE BYTE

AND ACC. WITH DEST BYTE

SAVE RESULT IN DEST. BYTE

INCREMENT BYTE INDEX (Y-REG.)

RETURN

Y

N

Y

N
BLACK

CALL SETUP

ACC. = 0 (FOR SEVEN BLACK PIXELS)

INIT. BYTE INDEX TO Ø (YREG)

STORE ACC. IN DEST. BYTE

INCREMENT BYTE INDEX

Y = 0?

CALL ENDCHECK

C-FLAG CLEAR?

RETURN
SETUP

SOURCE LOB = 0
DEST LOB = 0

1. SOURCE PAGE?
   1. SOURCE H.O.B. = $20
      - PAGE TO STOP AT = $40
      - RETURN
   2. SOURCE H.O.B. = $40
      - PAGE TO STOP AT = $60
      - RETURN

2.
ENDCHK

INCREMENT DEST. MEM. PG. NUMBER.

COMPARE ACC. WITH PG. TO STOP AT

CLEAR C-FLAG

DO MORE PGS?

N

SET C-FLAG

Y

INCREMENT SOURCE MEM. PAGE NO.

RETURN
KEYPR

PUT ASCII CD. OF LAST KEY PRESS IN ACC.

SAVE ACC. IN TEMPI

CLEAR KEYBOARD STROBE

RETURN
DECIDE

WAIT FOR KEYPRESS AND CLEAR STROBE

INIT, ZN & ZP TO ZERO

KEY?

Y

ZN = 1
RETURN

N

KEY?

Y

ZP = 1
RETURN

N

KEY?

Y

ZN = 5
RETURN

N

KEY?

Y

ZP = 5
RETURN

F
F

YES

ESC KEY?

RETURN

NO

ACC. < $31?

E

ACC. ≥ $8A?

E

NO

KEY WAS A DIGIT (0-9)

STORE ACC. IN EP

RETURN
KEY2

PUT CHARACTER (CHAR) IN ACC.

INIT X-REG TO ONE

IS CHAR "A"?

IS CHAR "C"?

IS CHAR "D"?

INCREMENT X-REG.

INCREMENT X-REG.

INCREMENT X-REG.

INCREMENT X-REG.

H
To INDICATE ERROR

INCREMENTS X-REG TO INCREASE ERROR

SAVE X-REG IN TEMPI

RETURN
BASCALC

STORE YPOSN MOD 8
IN TEMPL, H

STORE INT(YPOSN/8)
IN TEMPI

STORE TEMPL, H * 1024
IN SUML, H

STORE TEMPI MOD 8
IN TEMPL, H

STORE YPOSN/64 (= TEMPI/8)
IN TEMPI

STORE TEMPL, H * 128
IN TEMPL, H

STORE SUML, H + TEMPL, H
IN SUML, H

STORE YPOSN/64 * 40
IN TEMPI

STORE ACC. + SUML, H
IN SUML, H

STORE SUML, H + 8192
IN SUML, H

RETURN
DIV8

INIT X-REG TO ZERO

SUBTRACT EIGHT FROM ACC.

INCREMENT X-REG (COUNT EIGHTS)
EXP2

CLEAR C-FLAG

ROTATE TEMPL, H ONE LEFT

DECREMENT X-REG

\( x = 0 \) ?

\( N \)

RETURN

\( Y \)