PRINTED CIRCUIT BOARD COMPUTER AIDED DESIGN
ON THE APPLE II PLUS

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
Yi Chia Tang Vorhis
June, 1984
ACKNOWLEDGEMENTS

I would like to thank Dr. James C. Gilfert for his advice and help throughout the development of this thesis. I found this an enjoyable project, and am glad to have had this opportunity.

I am also indebted to Dr. Harold F. Klock for his informative lectures and personal assistance during my graduate study.

Last, I would like to thank my family. I am especially grateful to my aunt and uncle, Dr. and Dr. C. T. Byron Kao, for supporting my advanced study in the United States.

COPYRIGHT ACKNOWLEDGEMENT

Apple, Apple II Plus, Applesoft, DOS 3.3 are copyrighted by the Apple Computer Company.

VAX 11-730 is a registered trademark of the Digital Equipment Corporation.
## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>System Characteristics / Set Up</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Section 1 Memory</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Section 2 Solder Side and Component Side Array</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Section 3 Fundamentals of Applesoft BASIC</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>Section 4 Files</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>Program Organization</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Section 1 PART I</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Section 2 PART II</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>Section 3 PART III</td>
<td>36</td>
</tr>
<tr>
<td>4</td>
<td>Program Features</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>Section 1 Wire List</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>Section 2 Daisychaining</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>Section 3 Plated Through Holes</td>
<td>46</td>
</tr>
<tr>
<td>5</td>
<td>Special Subroutines</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Section 1 Matrix</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Section 2 Screen Show</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Section 3 Corners</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Section 4 Chain</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>Section 5 Sound</td>
<td>53</td>
</tr>
<tr>
<td>6</td>
<td>System Operation</td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion</td>
<td>64</td>
</tr>
<tr>
<td>A</td>
<td>Applesoft BASIC Commands and DOS Parameters</td>
<td>67</td>
</tr>
<tr>
<td>B</td>
<td>Flowcharts</td>
<td>76</td>
</tr>
</tbody>
</table>
Appendix C  Program Listing.................................92
Part I: Layout........................................93
Part II: Wire List....................................98
Part III: Trace Routing..............................102
Matrix and Clear Routines.........................109
Corners and Screen Show Routines..............112
Bibliography............................................118
Chapter 1. Introduction

The PC Board program is a software package which allows the user to quickly and easily create routings for printed circuit board design. User interactive prompting provides the flexibility necessary for a variety of needs. The program will run on a 48k APPLE computer, and offers menu driven selections.

Features include: flexible device layout alternatives, chip deletion, relocation, addition, wire list editing, comprehensive error checking, multiple routing algorithms, trace erasure, rerouting, daisychaining, single or double sided option, plated through holes, overlaying of solder and component side, repeated layout variations for the same wire list, archival storage of layout, wire list, and fully traced boards, single or dual disk drive setting, independent program and data disk option, saving of partially completed board for subsequent addition, etc.

The PC board program is intended for layout and design of small boards with 40 devices or less. The user can arrange the placement of each device in whatever order is desired. Discrete components like diodes, resistors, or capacitors are considered two terminal devices. Transistors or three terminal components would be socketed.
as part of a larger device.

After the devices have been arranged in the desired configuration, the user is expected to enter wire list data. It is easiest to compile the wire list on paper beforehand, and simply transfer the information when required. Although this may be a moderate chore, the wire list need not be re-entered each time a subsequent alteration is made to the layout file. This makes it much more convenient to erase devices, relocate them, or add new devices, and then reroute the traces. The procedure for creating a wire list is also very simple. It was perfected only after extensive trial and error. The wire list accommodates multiple pins on the same device, allows easy editing of entries, and is readily visualized. Much effort has been expended to assure flexibility and compatibility with other sections of the program.

The final task is to route the traces. The user is placed in an interactive role with the PC board program. Since it is physically impossible to foresee and handle all contingencies in the process of routing a trace from start to finish, a certain amount of user intervention is required. This is often apparent when the program successfully routes a trace, but the user is not satisfied with its asymmetrical form. The routing process is an
extremely complicated procedure, designed to handle many circumstances. It is also internally very flexible, because the same point in the program may be approached from several different directions. The many menu options during trace routing are proof of this.

Commercially available software which performs these functions exists almost exclusively for mainframe or minicomputers. Software houses like Intergraphics, Digital Control Systems, Micrografx, Enertronics Research, etc. have approached the microcomputer market with less specific goals. The Computer Aided Design (CAD) and graphics workstations available on small systems are suitable for drafting work, mechanical drawing, artistic expression etc. PC Board layout and routing is such a complex problem, that it is considered not approachable on microcomputers. Only by careful partitioning of user and microprocessor functions, is it possible to complete the task. In addition, commercially available systems are quite expensive, requiring special hardware, and high resolution graphics processing equipment.

The advantages of the PC Board program are thus clearly evident. Affordability and capability are primary characteristics. The PC board program brings automated design of small PC boards to the hobbyist or home design
engineer, where it was formerly available only to professionals with $100 K computer systems like Digital Equipment Corporation's PDP-11/70, VAX 11-730, etc.

The chapters which follow assume a basic acquaintance with microcomputers in general, and the Apple II+ in particular. Readers not familiar with these topics are referred to the bibliography for suitable preliminary materials. The Apple II Reference Manual and the Apple DOS Manual are particularly recommended.
Chapter 2. System Characteristics / Set Up

Section 1. Memory

The APPLE II PLUS microcomputer is considered to be a 64k machine. Most modern APPLE systems utilize 48k of dynamic RAM. The remaining 16k is generally dedicated to system I/O, and monitor and BASIC in ROM. Special peripheral cards are available to replace the ROMS in the memory map with RAM, by software controlled switches. Such peripheral cards are called RAM cards. In order to provide greater system compatibility and appeal, utilization of a RAM card in this project was avoided.

The decision to complete all tasks within 48k of machine memory, led to severe programming limitations, which were overcome through careful planning and memory management. Figure 2.1 describes the APPLE memory mapping. This illustration neglects some of the detailed options offered to the user, such as HI-RES page 2, DOS file buffer flexibility, etc.

It should be noted that the RAM area from $0800-$95FF is all available to the user if HI-RES graphics are not needed in the program. That is, a BASIC or assembly language program could stretch from $0800 all the way toward $95FF without any problem as long as HI RESOLUTION PAGE 1 (and PAGE 2) remained inactive. Hi Resolution
<table>
<thead>
<tr>
<th>Region</th>
<th>Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASIC &amp; MONITOR</td>
<td>0000-FFFF</td>
</tr>
<tr>
<td>ON BOARD I/O</td>
<td>0000-CFFF</td>
</tr>
<tr>
<td>DOS</td>
<td>C000-HIGHEST RAM ADDRESS</td>
</tr>
<tr>
<td>DOS FILE BUFFERS</td>
<td>9600-95FF</td>
</tr>
<tr>
<td>USER AREA</td>
<td>9CF8-95FF</td>
</tr>
<tr>
<td>(AVAILABLE FOR BASIC)</td>
<td></td>
</tr>
<tr>
<td>HI-RES PAGE 1</td>
<td>3FFF-4000</td>
</tr>
<tr>
<td>USER AREA</td>
<td>2000-1FFF</td>
</tr>
<tr>
<td>(AVAILABLE FOR BASIC)</td>
<td></td>
</tr>
<tr>
<td>TEXT / LO-RES</td>
<td>0000-07FF</td>
</tr>
<tr>
<td>DOS &amp; SYSTEM</td>
<td>03DO-03FF</td>
</tr>
<tr>
<td>USER AREA</td>
<td>03DO-03CF</td>
</tr>
<tr>
<td>KEYBOARD BUFFER</td>
<td>02FF-0300</td>
</tr>
<tr>
<td>STACK</td>
<td>0000-01FF</td>
</tr>
<tr>
<td>ZERO PAGE</td>
<td>0000-0100</td>
</tr>
</tbody>
</table>

**FIGURE 2.1**

Standard DOS 3.3 memory mapping.
graphics page 1 is, however, used to display the printed circuit board. This splits the effective user available RAM into two separate parts. The BASIC program or its variables, may not occupy any part of HI-RES page 1.

Figure 2.2 illustrates the default configuration of memory after loading a BASIC program. The program starts at $0800$ and extends toward higher addresses. Just beyond the end of the BASIC program is the LOMEM pointer. LOMEM marks the lower limit of real and array variable tables. Clearly, as the number of variables increases, the likelihood of overflowing into HI-RES page 1 increases. This appears as dots or lines scattered throughout HI-RES page 1.

A technique frequently used to avoid HI-RES page 1 overflow is shown in Fig 2.3. LOMEM is set by means of a BASIC instruction, to a memory location just beyond the end of HI-RES page 1. This allows vertical growth of variable tables as well as providing additional space for the BASIC program. Unfortunately, if the BASIC program becomes too large, it will overflow into the HI-RES screen, even though the variables are safely above HI-RES. Furthermore, if the Chain routine is used to link two BASIC programs together, LOMEM gets automatically reset to a position just beyond the end of the BASIC program. Therefore, even if the BASIC
FIGURE 2.2

Demonstration of over-write problems, when AppleSoft BASIC is used with standard mapping.
Modified mapping which alleviates the problems of Figure 2.2 for small programs.
is safely shy of the HI-RES memory, and LOMEM has been set to an address beyond $4000, as soon as any other program is "chained" to the first, LOMEM will be reset and the variable tables may overflow into HI-RES page 1. For the above reasons, memory was configured as shown in Fig. 2.4.

The BASIC program is made to begin at $4001 rather than $0800. This is accomplished by altering the Applesoft "Start of Basic" pointer located at zero page locations $0067 and $0068. In addition, a $00 is poked to location $4000 since the start of any BASIC program must be preceded by a $00. Relocating the BASIC program in this way, completely prevents any overflow of extraneous bits into the HI-RES memory. LOMEM is free to follow the BASIC program. The BASIC program and variables may occupy from $4000 to HIMEM. The area from $0800 to $2000 is now available for the PC board Solder and Component Side Arrays.

Section 2. Solder Side and Component Side Array

A mixed graphics / text screen is utilized so that instructions and information can be easily conveyed to the user. The HI-RES / Text screen of the APPLE II PLUS displays 44800 pixels in a 280 wide by 160 high array. Four lines of text are provided at the bottom of the screen.

In the low resolution graphics mode, it is possible to
<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
<th>Memory Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>BASIC &amp; MONITOR</td>
<td>BASIC &amp; MONITOR</td>
</tr>
<tr>
<td>0000</td>
<td>RCMS</td>
<td>RCMS</td>
</tr>
<tr>
<td>0000</td>
<td>ON BOARD I/O</td>
<td>ON BOARD I/O</td>
</tr>
<tr>
<td>0000</td>
<td>DOS</td>
<td>DOS</td>
</tr>
<tr>
<td>0000</td>
<td>DOS FILE BUFFERS</td>
<td>DOS FILE BUFFERS</td>
</tr>
<tr>
<td>0000</td>
<td>ASSEMBLY SUBROUTINES (MOST ROUTINES &amp; SHAPE TABLES)</td>
<td>ASSEMBLY SUBROUTINES (MOST ROUTINES &amp; SHAPE TABLES)</td>
</tr>
<tr>
<td>0000</td>
<td>BASIC VARIABLES</td>
<td>BASIC VARIABLES</td>
</tr>
<tr>
<td>0000</td>
<td>BASIC PROGRAM</td>
<td>BASIC PROGRAM</td>
</tr>
<tr>
<td>0000</td>
<td>HI-RES PAGE 1</td>
<td>HI-RES PAGE 1</td>
</tr>
<tr>
<td>0000</td>
<td>SOLDER SIDE ARRAY</td>
<td>SOLDER SIDE ARRAY</td>
</tr>
<tr>
<td>0000</td>
<td>COMPONENT SIDE ARRAY</td>
<td>COMPONENT SIDE ARRAY</td>
</tr>
<tr>
<td>0000</td>
<td>TEXT / LO-RES</td>
<td>TEXT / LO-RES</td>
</tr>
<tr>
<td>0000</td>
<td>DOS &amp; SYSTEM</td>
<td>DOS &amp; SYSTEM</td>
</tr>
<tr>
<td>0000</td>
<td>MATRIX SUBROUTINE &amp; ARRAY CLEAR</td>
<td>MATRIX SUBROUTINE &amp; ARRAY CLEAR</td>
</tr>
<tr>
<td>0000</td>
<td>KEYBOARD BUFFER</td>
<td>KEYBOARD BUFFER</td>
</tr>
<tr>
<td>0000</td>
<td>STACK</td>
<td>STACK</td>
</tr>
<tr>
<td>0000</td>
<td>ZERO PAGE</td>
<td>ZERO PAGE</td>
</tr>
</tbody>
</table>

**FIGURE 2.4**

Memory utilization optimized for PC layout problem.
determine whether the current cursor position coincides with a location on the screen where the pixels are not blank. In other words, it is easy to determine the color of a low resolution screen location. In this way, if a node is being constructed, it is simple to detect whether a trace has bumped into another trace. The intersection of 2 nodes might thus be avoided.

Unfortunately, the low resolution mode does not possess the resolution necessary for 40 pin IC's interconnected by intricate trace patterns. Clearly, high resolution graphics is the only real choice. High resolution mode, on the other hand, does not have the ability to return the color of the screen at the current cursor position. Furthermore, the high resolution screen, as it resides in memory, is scrambled in relation to the way it is displayed on the screen. The video memory is not displayed sequentially to the screen. This prevents easy determination of a particular high resolution screen location.

If only HIGH Resolution Video Memory were used in the determination and construction of a trace (as well as the subsequent display of the completed trace), numerous obstacles would have to be overcome. When a trace bumped into a particular pixel which was turned on, the only way
to determine whether it bumped into a pad, a corner, a plated-through hole, or another trace, would be by color.
Likewise, crosshairs or shapes to mark the appropriate pins for that node would need to be distinguished from the actual PC board components. In addition, it would be very difficult to manage a complex board, because the appropriate number of blank pixels would need to be maintained between adjacent traces, pads etc. For example, if there were 3 pixels separating two neighboring pads, only one trace would be allowed to pass between the pads (through the middle), even though blank pixels remain on either side of the trace. In other words, all blank pixels would not be alike.

A further complication of the High Resolution Video Memory Management scheme arises when an attempt is made to erase the last section of node just completed. Again, the only way to distinguish between a recent section of node and an old section of node would be by color. Since color in the APPLE II decreases the resolution and detail of the high resolution screen, any attempt to display color would not display certain vertical traces at all. Because of these and many other problems associated with direct Hi Resolution Video tracing, this approach is not suitable for microcomputer application.
Instead, a separate portion of memory is set aside as a scratchpad screen. This is in the form of an array. In fact, two such arrays exist, one for the solder side and one for the component. In theory, if enough memory existed in APPLE, multiple level PC boards could be designed nearly as easily as single sided boards.

The solder side array occupies $1310-\$1DFF$, and the component side array occupies $0810-\$12FF$. Each location in the array represents a specific 3 by 3 block of HI-RES screen pixels. Since the HI-RES screen is 280 by 160 pixels, each array was rounded to 92 by 52 matrix locations, or 4784 locations. Each solder or component side matrix location is represented by 1 nibble rather than a complete byte in memory, therefore both solder and component side arrays occupy only 4784 bytes rather than 9568 bytes in memory. Because memory utilization is at a premium, such a savings is significant. Each array location holds a hexadecimal value which represents a particular shape on the screen. Thus a $0$ is a blank 3 by 3 block on the screen. A hexadecimal $A$ represents an IC pad on the screen etc. Since a nibble can contain only 16 different binary combinations, only 16 different shapes may be distinguished on the screen. The memory space saved by specifying only 1 nibble per array location is therefore
offset by the limitation to 16 characteristics per location.

Figure 2.5 describes the assignment of characteristics to elements of the solder or component side array. Note that certain assignments appear identical to the user. For example, elements $1, $2, $3 all appear as a single dot on the video screen. As far as the user can distinguish, there is no difference between these elements. The program, however, distinguishes hidden attributes of these assignments. An array element holding $1 signifies a corner of a section of trace just completed. A $2 represents a corner of the same node, but not the latest section of this node. A $3 represents a section of an old node, not the currently active node. In this way, transparent to the user, the latest section of node or the entire currently active node may be erased and repeated. This avoids the complicated process of backtracking and erasing a whole trace. In order to erase the most recent section of trace, all $1, $4, and $7 values are erased from solder and component side arrays.

It is important to note the reasons for choosing each solder or component side array element as a 3 by 3 pixel array. Clearly, a 2 by 2 pixel would not suitable, as any horizontal or vertical trace element must have blank spaces
<table>
<thead>
<tr>
<th>NIBBLE</th>
<th>VIDEO ASSIGNMENT</th>
<th>characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>BLANK</td>
<td></td>
</tr>
<tr>
<td>$1</td>
<td>.</td>
<td>MOST RECENT CORNER</td>
</tr>
<tr>
<td>$2</td>
<td>.</td>
<td>LESS RECENT CORNER</td>
</tr>
<tr>
<td>$3</td>
<td>.</td>
<td>PERMANENT CORNER</td>
</tr>
<tr>
<td>$4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2.5**

Identification of the sixteen available shapes for any matrix location.
on either side for visual separation. In fact, only oddly dimensioned pixel arrays are suitable (3 by 3, 5 by 5, 7 by 7 etc). While a 5 by 5 array allows a wider variety of screen shapes, it does not allow as many pins on the screen. Only 32 5 by 5 array elements will stack end to end vertically on the 280 by 160 pixel HI-RES display. Since each pin of an IC must be separated by space enough for just one trace, each 5 by 5 pad is separated by one 5 by 5 blank space. Only 16 pads (a 32 pin chip) can therefore extend vertically down the screen, alternated by 16 blank spaces. 5 by 5 pixel arrays would not allow 40 pin chips on the screen.

By comparison, 3 by 3 pixel arrays stack up to 53 high, or 26 pads alternated by spaces. 40 pin chips easily fit on the screen.

A conceptual illustration of a portion of the solder side array, and a sample of how this section of solder side and component side would appear on the screen, are shown in Figure 2.6, 2.7, and 2.8 respectively. Note that this approach inherently centers the trace on the pad, prevents neighboring traces from intersecting, allows easy identification of array elements and is far superior to the direct video screen tracing approach. The penalty, of course, is the necessity of providing a mapping algorithm
FIGURE 2.6

Solder side array values as they would appear in RAM for a portion of a board.
FIGURE 2.7

Solder side traces corresponding to the matrix shown in Figure 2.6.
FIGURE 2.8

Component side traces corresponding to the matrix of Figure 2.6.
to convert the matrix to Hi Resolution screen RAM entries to permit its display.

Section 3. Fundamentals of Applesoft BASIC

The APPLE II PLUS computer can be programmed in several languages. Much of the PC Board program is written in Applesoft BASIC, in order to take advantage of the disk access and file routines as well as string manipulation and I/O offered by Applesoft.

There are certain limitations to using Applesoft BASIC. A major disadvantage is slow execution of code. The BASIC interpreter in the APPLE's ROM requires each BASIC statement to be interpreted before execution. This means that each time a statement is executed (even within a loop), the APPLE II PLUS decodes the statement before performing the function. This takes time. Therefore, certain functions of the PC Board program which require high speed were written in 6502 assembly language. These machine language routines will be discussed in Chapter 5.

Although BASIC is inherently slow, good programming practices can optimize its performance. Including multiple instructions in a single numbered line can vastly decrease execution time of a program. This is done by separating the statements with a colon. Other advantages of this technique are obvious. 1) A number of short, related
statements may be grouped together. 2) Remarks may be included on the same line as the BASIC statement. 3) Memory is saved by reducing the number of line numbers in the program.

It is also important to distinguish Applesoft BASIC commands from DOS commands. The distinction is not always obvious. For example, the Applesoft and DOS error messages are very similar. Likewise, it is possible to accidentally execute a DOS command from a purely Applesoft BASIC instruction, such as a REMARK statement. In general, DOS commands are concerned with file handling or disk procedures.

In order to execute a DOS command from BASIC, it is necessary to precede the command with a printed carriage return, and a control D (Hexadecimal $04). Thus PRINT: PRINT CHR$(4); "catalog" will cause the disk to reveal its file directory. Note that the first PRINT serves to generate a carriage return.

Most DOS commands allow the programmer to specify a number of options, such as the disk drive number to be accessed, the slot number of the disk controller card, and a "volume number" to identify the disk. The PC Board program defaults to slot 6 for the disk controller card, and only selects the appropriate drive number for data and
program.

As with any moderately complex program, the Applesoft BASIC programming language possesses certain flaws which only arise under special conditions. As the PC Board program increased in size, for example, the XDRAW command began behaving unpredictably. With the shape table in memory, shapes can be drawn on the screen by using the statements SCALE, ROT, DRAW and XDRAW. Ordinarily, the specified shape is drawn to the exact coordinate indicated by the XDRAW command. However, in limited memory, the XDRAW command does not locate the shape at the proper pixel position on the screen. The shape is shifted up on the screen by 1 pixel. By directly calling the position subroutine (followed by a call to the XDRAW routine), the problem is solved. Evidently, the exact screen position is not guaranteed by the BASIC XDRAW command.

The APPLE II PLUS makes use of special address locations which control the video display. The text, graphics, and mixed screen modes are determined by these "soft switches". By placing special addresses on the bus, the particular switches (flip flops) are set or cleared. Thus, through either BASIC or assembly routines, the software may set the desired video configuration. It does not matter whether the address is placed on the bus by a
READ, or WRITE, PEEK or POKE. The switch is thrown regardless. Machine language programs LOAD or STORE the hexadecimal addresses; BASIC programs PEEK or POKE the decimal equivalents.

In addition to the video switches, additional on-board I/O locations are worth mentioning. The keyboard latch and clear keyboard strobe are two heavily utilized switches which extend the capabilities of keyboard entry. With these switches it is possible to examine a keypress without waiting for the key to be pressed. Another useful switch controls the speaker. The PC Board program toggles the speaker at frequencies which produce an audible error message.

Section 4. Files

The PC Board program uses both random access and sequential text files. Both text files have advantages and disadvantages. The reason for using both text files is to accomplish different goals. The random access file has a length parameter. Each record is of a specified length, as shown in Figure 2.9. Every record in the random access file specifies a chip, where record N corresponds to chip N.

The pin layout file needs to specify the number of device pins, and the row and column of pin 1 (x, y
FIGURE 2.9

Several records in segment of Random Access Text File.
Record length is 4.

FIGURE 2.10

Example of Sequential Text File for wire list of 3 devices.
coordinates of pin 1). The length of each record in the file is fixed. Thus, OPEN Z$, L4 sets up a 4 field file record. This allows space for the 3 data items followed by a carriage return. The advantage of a fixed record length is that it allows immediate access of any record in the file. Applesoft can handle the file pointers of a random access file quite easily, since the record length and the record number are both known. This characteristic allows the user to reposition, delete, or add a chip simply by utilizing the appropriate record number. In other words, the user can randomly access the number of pins and row and column of any chip in the layout file. It does not need to search sequentially through the entire file for the proper record. Since most layout file manipulation occurs in Part I, this is where most random access text file activity occurs.

Part II of the PC Board program manipulates sequential text files. (See Figure 2.10.) This is because the number of pins and the number of chips in any given node is variable. Since the node length is not constant, a fixed record length is not desireable. Any attempt to establish a fixed length would need to accomodate both large and small node entries. The only way to handle this situation is to utilize sequential files. A sequential text file
does not require a uniform length for each record. The length of each record is only as long as it needs to be. This allows the file on the disk to hold as much information as possible. A disadvantage of such a scheme is that, finding a specific node is not as easy as for random access files. The node must be found by a sequential search. The format of data within each record in the sequential file was defined so that each layout file device is separated by a slash. This allows the proper pin to be associated with the appropriate device. Multiple pins on a single device are separated by a decimal point. Notice that each node describes the same number of devices in the layout file, but the record lengths may differ depending upon the number of pins in the node.

It is clear, nevertheless, that for the wire list file, the sequential technique offers advantages in disk space savings, as well as node capacity.
CHAPTER 3. Program Organization (Part I, Part I, Part III)

There are three separate parts in the PC Board program. Each part is substantially independent of other parts. The object of Part I is to generate and manipulate the IC layout file. Part II handles the wire list file. Part III uses the files of Part I and II, and traces nodes. Extensive Error Checking occurs in Part I and II to ensure that the layout and wire list files, once in place, are correct and need no further error handling precautions in Part III.

All parts use a menu driven structure which automatically establishes the sequence of operations. All menus are self explanatory. Improper entries are trapped out.

Section 1. Part I

At the beginning of Part I, several introductory questions are asked. The user can choose between one and two disk drive operation. The assumption has been made that there are separate program and data diskettes, but this is not absolutely necessary. Data files may be stored on the program disk.

The user next has the opportunity to view a traced printed circuit board which was completed previously. This allows fast access to work already done.
Action in Part I is controlled both by keyboard and joystick. A standard 150k joystick is recommended which provides a range of zero to 255 for the paddle value. Paddle 0 controls horizontal movement (movement from column to column). Column=276/255*PDL(0)/3. Paddle 1 controls vertical motion (movement from row to row). Row=156/255*PDL(1)/3. Button 0 selects the present location as the selected layout. Button 1 aborts the current attempt and sets up for another try.

Of course, integrated circuits do not have arbitrary numbers of pins. They also have different widths, depending upon the number of pins. Two distinct shapes have been defined in a shape table to account for this. The user must respond to a question in Part I which requests the number of pins for that IC. If the IC has 2 through 20 pins, then the width is set equal to 6, and uses shape 1. If the IC has 24 to 40 pins, the width is set to 12, and uses shape 2. If the number of pins entered is invalid (such as 7 pins or 4 pins), the error is detected and handled. Several types of errors are dealt with in Part I. Boundary check routines make sure all chips are located on the video screen. If two chips are too close together (less than 3 pixels apart horizontally) or one chip is on top of the other, the overlap error check routine will find
it. The chip occupies a certain space within the matrix. Since the chip's two lines of pins are oriented vertically, half of the pins are in one column of the matrix, and the other half are in another column. The spacing between the pins is determined by the matrix rows. There is one row position between each pin. This allows only one trace to pass between pins.

Chip overlap is determined by starting at a matrix position which is 3 columns beyond, and 1 row short of the desired pin 1 position. A short section of that row is then scanned to make certain no other pins occupy the space. The row is incremented and an equivalent section of row is scanned. In this way, the entire space (and a little beyond) is checked for overlap. The reason for scanning the chip beyond the boundaries of the chip itself, is not only to prevent 2 chips from being too close together. It also prevents a narrow IC of width 6, from being located inside a larger IC of width 12 (see Figure 3.1).

The user has 5 selections in order to establish the IC layout. Left or right arrows select the option, carriage return proceeds with the chosen action.

These are the options:
1). GENERATE TRACES FOR PC BOARD
2). EXAMINE THE PIN PLACEMENTS
Example of an overlap error which would be detected in Part I.
3). REPOSITION A COMPONENT ON PC BOARD

4). REMOVE A COMPONENT FROM PC BOARD

5). ADD A COMPONENT TO THE LAYOUT

Option 1) selects the tracing function. This implies that the layout file is complete. Before proceeding to the actual trace, however, the wire list must first be available. Another menu is displayed which determines whether to proceed to Part III for the trace, or to Part II for the wire list. Each part will be explained in later sections.

Option 2) displays the IC chips as they are distributed in the layout file. A display of each chip showing the number of pins and the location of pin 1 (X=ROW, Y=COLUMN) is issued. This is followed by the physical drawing of each chip to the screen. This gives a good illustration of where the chips are located. If the user does not like the layout, option 3, 4, or 5 may be selected to edit the existing file.

Option 3) allows joystick controlled repositioning of all chips in the layout file. Program execution repositions the selected chip and also executes all error checking routines to make sure the repositioned chip is in an acceptable location.

Option 4) erases a selected chip from the screen, but
leaves a void in the layout file in order to retire the IC number. In this way, subsequent chips added to the layout cannot occupy the file location of a previously deleted chip. This allows wire list compatibility (after the layout has been edited, there is no need to retype the wire list, since all other chips remain unchanged with the same IC number).

Option 5) adds a chip. If the previous wire list is used, events proceed as usual, but since the wire list does not include the added chip(s), nothing from the wire list appears on the screen for the new chip(s).

At the end of Part I, the next program is transferred to memory by a subroutine named "CHAIN". The CHAIN routine preserves variables. The program automatically checks for the presence of the appropriate wire list on diskette. If the wire list exists, execution goes directly to Part III which generates traces with the aid of the existing wire list. If no wire list is present, program execution goes to Part II which creates a new wire list. After the wire list is created in Part II, the logic proceeds to Part III for tracing.

Details regarding the CHAIN subroutine are presented in Chapter 5.

Section 2. Part II
Part II handles wire list formation. It relies heavily upon string manipulation. By definition, a node is a listing of the pins which are to be connected together. One device may have multiple pins on one node. Because the length of one node is a variable, the user cannot use fixed-length random access files. For this reason, the sequential text file is used in Part II. Wire list formation must be immune to changes in pin layout, and deletion and addition of chips. Consistent with the concept that the completed wire list should be free of errors, and should require little or no error checking in Part III, extensive techniques to trap out errors are applied in Part II.

A table-driven approach to wire list entry has been developed for Part II. The contents of the wire list file are converted to strings and displayed in a table. Manipulation of any table entry is possible. In the end, the table entries are written back into the file. There are two possible modes of entry: 1) Creating a new a wire list, and 2) Editing a previous wire list. 1) and 2) are distinguished simply by the initial assignment of strings to the table. For mode 1), the strings are cleared so that the table is blank. For mode 2), the old wire list file is read and converted to the table strings.
Each table displayed on the screen describes the wire list for a single node. The size of the table is determined by the number of devices (up to 40 devices). Selection of a device to be edited is made by left and right arrows. Error checking prevents improper entry.

To enter the edit mode, the carriage return is pressed. This allows free cursor movement and editing of characters within the field (1 FIELD = 1 DEVICE). Left arrow on the APPLE computer normally erases what was entered through string manipulation. This effect is eliminated so it is possible to backspace to a previous pin entry and make a change without requiring subsequent reentry of the field with the right arrow. Again, in the edit mode, error checking does not accept improper key closures. When the node is completed, an analysis is made of each string. Additional error checking assures no improper pin numbers are entered for any device. A new sequential wire list is generated. The new file is the "packed" wire list file, which generally occupies much less space than the sum total of all strings. The user may proceed to the next node if desired.

When the wire list is completed, control is transferred to Part III where the actual routing of traces is performed.
Section 3. Part III

Part III is the most difficult and also the most interesting part of the program. The goal in Part III is to generate trace routings. There are fundamentally 6 menu items which illustrate the major functions of Part III.

The options are:
1) CONTINUE WITH NODE "n"
2) PROCEED TO NODE "n"
3) REPEAT THIS LAST SECTION OF NODE "n"
4) ERASE ALL NEW SECTIONS OF NODE "n"
5) TRY OTHER PROCEDURE ON NODE "n"
6) DAISYCHAIN ENABLE/DISABLE
7) OVERLAY SOLDER AND COMPONENT SIDE

PRESS ESCAPE TO FLIP THE PC BOARD (a menu item only for double sided boards)

Option 1) allows the user to continue with the current node. Selection of this item means that the user is satisfied with the last section of node just completed. After this option is chosen, the only way to erase is to erase the entire node.

Option 2) indicates that the user is satisfied with the last section of node just completed, as well as with the entire node. Once the user goes on to another node, the previous node cannot be erased, only added to. After
this option has been selected, most of the corners which were dots, are changed to true corners which make the trace better in visual appearance.

Option 3) allows the user to redo the last section of the current node. Previous sections of node are not erased. This would be utilized when the user is not satisfied with the last section of node just completed.

Option 4) erases the entire node. However, if portions of the node were completed earlier and this is the second look at the node (The option is given to re-examine/edit nodes which were previously completed), then only the traces added during the second look will be erased.

Option 5) is a very convenient option for attempting different procedures on the node. This does not necessarily mean that subsequent procedures will succeed or yield different results than previous tries. If too many alternate procedures are attempted, the trace will fail and the user must start over. The trace can often be made to correspond more closely to what the user desires with this option.

Option 6) The daisychain function allows point to point tracing of the node. When the user proceeds to another node, this function is temporarily disabled to get
a new starting point for that node. Daisychaining is then resumed for later sections of that node. In this way, the daisychain allows easy "automatic" entry of node endpoints.

Option 7) allows the user to see both sides of the board simultaneously. When the overlay function is enabled, all trace routings are displayed from the solder side of the screen. This gives the user the opportunity to find vacant locations on both sides of the board where plated-through holes would be possible. The overlay function, when enabled, is effective on a single pass when adding a section to the current node.

It is helpful to know how the trace is determined. The beginning and ending points of a node can be thought of as occurring on the diagonal corner of a box (see Figure 3.2).

The trace can "only" stay within the box (except for certain special cases). This limits the trace so that it does not wander all over the screen. Occasionally, what appears to be an easy trace cannot be completed because of this principle. The trace almost never "reverses" direction in going from start to finish. The only time when it might is when it goes horizontally around a pin (see Figure 3.3), or vertically along a row of pins (see Figure 3.4). At all other times, the horizontal and
FIGURE 3.2
Example of trace with vectors inside imaginary box.

FIGURE 3.3
Example of trace with horizontal vector component outside imaginary box.

FIGURE 3.4
Example of trace with vertical vector component outside imaginary box.

COMPONENTS ALWAYS TOWARD ENDING PIN.

HORIZONTAL COMPONENT NOT DIRECTED TOWARD PIN.

VERTICAL COMPONENT NOT DIRECTED TOWARD PIN.
vertical vectors add up to point at the ending point.

The trace initially tries to go up or down at the halfway point of the imagined box. The trace usually proceeds horizontally first. If the routing cannot successfully be completed for a particular trace (for example, another trace, pad or plated-through hole was encountered), repeated attempts are made to successfully complete the routing. Each attempt tries a slightly different procedure. If all attempts fail on the solder side, and the PC board is double sided, the trace will automatically analyze the component side. In other words, when a trace cannot be completed on the solder side, the total trace from beginning to end, is attempted on the component side. All tracing variations will then be repeated on the component side.

As just described, the first attempt at successfully completing the routing is made by forcing the pathway to travel vertically at the horizontal midpoint between the starting and ending point of that section of node. In addition, the starting point is always lower on the video screen than the endpoint on the first attempt. This ensures a more symmetric routing path. Figure 3.5 a) illustrates this condition, and a typical obstacle in the path of the successful routing. If the first attempt
FIGURE 3.5 Showing alternate routing procedures attempted when standard routing fails.
FIGURE 3.5 (Continue). Alternate routing procedures.
cannot be completed, the starting and ending points are switched, as seen in Figure 3.5 b). Now with the starting point higher on the screen than the ending point, the routing can be finished. Clearly, switching the endpoints enables an obstacle to be circumvented which would otherwise block the routing.

It is not always possible or desirable to turn at the halfway point between start and finish. Figure 3.5 c) depicts a situation in which the first two routing attempts have failed. The next attempt is to not turn at the halfway point, but to continue traveling horizontally as far as possible. Figure 3.5 d) utilizes the same technique, but with the endpoints switched. As before, each successive procedure is performed only when the previous procedures have failed.

The next iteration attempts to turn only three columns from the starting point rather than at the halfway point or far from the starting point. This is shown in Figure 3.5 e), and allows the routing to reach the endpoint between traces which have obstructed the other techniques. Again, the endpoints are switched as shown in Figure 3.5 f) to provide even greater capability.

In all of the previous methods, the routing was forced to turn from the horizontal path and reach the endpoint row
at least three columns before reaching the endpoint column. This prevents a vertical trace from blocking an entire column of pins. There are times when this technique is undesirable, particularly when the user is trying to backtrack around an obstacle. In such cases it is better to turn at the endpoint column rather than before. Otherwise, that node would have a short section of trace traveling beside itself and meeting at the end. An inductive loop of this kind is best avoided. Figure 3.5 g) and 3.5 h) address this problem. It should be noted that this final technique is not an attempt to complete a routing where other methods have failed. In fact, most of the other techniques would probably succeed before this technique was applied. The user must select option 5, "TRY OTHER PROCEDURE ON NODE n" in order to take advantage of the method.

The available routing procedures were selected so as to handle the most common causes of failure to complete a trace. In addition, there are many special routing techniques that are employed to handle particular situations. These conditions are quite complicated, however.
Chapter 4. Program Features

Section 1. Wire List

It is very important that the user be able to create a new wire list or examine and edit the old one conveniently. The PC Board program allows the user to use the existing wire list without any changes or to create very easily a new wire list or to examine and edit the old one. The wire list is displayed in a 1- or 2-column table depending upon the number of chips. If one of the chips in the layout file has been deleted, that chip is not displayed in the wire list table and therefore does not contribute wires to the node.

Left or right arrows are pressed to select the chip, then RETURN is pressed to edit the pins of that chip. Pin numbers are entered separated by commas or spaces. The fields will ignore leading zeroes or spaces, as well as interspersed spaces or commas. The left and right arrows allow transparent cursor movement in the edit field. RETURN is pressed again to get out of edit mode. The user can then go on to other chips. The ESCAPE key is pressed to proceed to the next node.

Section 2. Daisychaining

The daisychaining function was added to speed and simplify the process of entering the starting and ending
points of the current trace. This allows the user to enter the starting point and draw a trace to the ending point, to another endpoint, to the next endpoint etc. This means that only one position must be located with the joystick for each additional section of node.

It is usually advisable to proceed in one general direction while daisychaining (ie left to right, right to left, bottom to top, or top to bottom). This avoids backtracking the routing and does not allow interference with previously completed sections of the same node. (see Figure 4.1)

The daisychain function provides an additional feature which simplifies operation. Daisychaining is temporarily disabled when a new node is started. This prevents the connection of 2 separate nodes. The daisychaining is automatically re-initiated once subsequent sections of the new node are begun.

Section 3. Plated-through Holes

The PC Board program can plate through holes from the solder to the component side. It will automatically punch a plated-through hole through the board if there is nothing to prevent it (endpoint too close to another pad - a trace already on the component side etc.). A plated-through hole is always necessary when the trace is completed on the
AVOID THIS BECAUSE IT FORCES TRACE TO BACKTRACK.

THIS DOES NOT CAUSE BACKTRACKING. ROUTING FLOWS FROM LEFT TO RIGHT.

FIGURE 4.1

General technique for routing a trace.
component side. However, due to lack of memory, the plated-through hole is not seen until the user decides to go on to the next node (Option 2 in Part III) or continue with the same node (Option 1 in Part III), indicating that the user is satisfied with the trace as is. It is not until after the user decides on Option 1 or 2 in Part III that the check is made to see whether the plated-through hole is too close to another pad. If not, the plated-through hole will appear in the next display of the PC Board.
Chapter 5. Special Machine Language Subroutines

Section 1. Matrix

The Matrix subroutine reads or writes the solder or component side array. The Matrix subroutine and Array Clear subroutine are located from $0300$ to $03CF$ in memory. Since each array location occupies one nibble in memory, one byte accounts for two neighboring array locations. When accessing a particular row and column in the array, it is necessary to locate the appropriate byte as well as the proper nibble of that byte. The high or low nibble are determined by odd or even column respectively. The desired byte is located by an equation which is based upon the number of bytes in a row. Once the location is pinpointed it is easily read or written.

Section 2. Screen Show

The Screen Show routine is used to convert the binary data of the solder or component side array into the appropriate shape at the correct screen location. For example, an IC pad is identified as a hexadecimal $A$, in the solder or component side array, but is identified as a small square shape on the screen. Screen Show scans each location of the array and compares its value to a list. When a match is found, a table look up technique is used to find the proper sequence of plotted points. The row and
column of that array element are converted to screen pixel locations. The shape is then plotted and the next array element is evaluated. In this way, each array location is examined and drawn to the screen individually. Although the routine is written in machine language, it takes perhaps 3 seconds to display the entire array.

The same routine is used to display either the solder side or the component side. Obviously, a few bytes must be preset to point at the proper array, before the routine is called. The solder side array is scanned from left to right and top to bottom, on the screen. Likewise, the component side array is scanned from right to left and top to bottom, referenced to the screen. This is because the component side view is from the top of the board, while the solder side view is from the bottom. The proper bytes must be set prior to executing the Screen Show routine to ensure a consistent display.

Section 3. Corners

The Corner subroutine plays an important part in the PC Board program. The assignment of characteristics to elements of the solder or component side array is described in Chapter 2. The Corner subroutine establishes a special table which is used to replace appropriate dots with corners in the solder or component side array. The process
of converting dots into corners is initiated by examining the locations surrounding the dots. The locations above, to the right, below, and to the left of the central dot are important. Once these values have been read, they are packed into two bytes. The first byte describes the top and right locations. The next byte describes the bottom and left. The bytes are then compared to values in the table. If a match is found, the dot is replaced with the appropriate corner.

There are many variations and permutations which yield the same result. Figure 5.1 illustrates this point. Note that as far as the table is concerned, a blank space which is next to the dot is totally different than a trace occupying the same spot. Because of the possibility for many varieties of traces in any given location, the number of combinations this would generate, would be enormous. Whenever possible, this effect is minimized.

Section 4. Chain

Due to limited space, it is not possible for all parts of the program to reside in memory simultaneously. One approach to such a problem would be easily implemented by splitting the program into a series of two or more parts which are LOADED and RUN sequentially. The second program may need to use the values of variables and arrays
TRACES SURROUNDING DOT ON SCREEN

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TRACES SURROUNDING DOT IN ARRAY

<table>
<thead>
<tr>
<th>05</th>
<th>05</th>
<th>05</th>
<th>05</th>
</tr>
</thead>
<tbody>
<tr>
<td>05</td>
<td>05</td>
<td>05</td>
<td>05</td>
</tr>
<tr>
<td>08</td>
<td>08</td>
<td>08</td>
<td>00</td>
</tr>
<tr>
<td>00</td>
<td>08</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

PACKED TABLE DATA

| 5885 | 5805 | 5880 | 5800 |

AFTER DOT CHANGED TO CORNER

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FIGURE 5.1
developed by the first program. However, the RUN command erases the first program's variables and arrays when it loads the second program. In order to LOAD and RUN a series of Applesoft programs without erasing earlier values of variables or arrays, Apple computer provides a machine language program on disk, called CHAIN.

To CHAIN from the program called Part I to the program called Part II, the CHAIN program must reside on diskette. The last lines of Part I to be executed would be:

```plaintext
: PRINT : PRINT D$: "BLOAD CHAIN,A520" : CALL 520 "PART II".
```

Section 5. Sound

There are two ways to produce a sound with the Apple speaker: 1) The bell character (CHR$(7) or CTRL-G), 2) Addressing locations -16336 and -16352, which causes the speaker output to emit a very short click. The latter alternative can be done in either Applesoft or machine language. Because of limited execution speed, no high frequency tones can be produced with Applesoft alone. A machine language routine was written, to produce an alarm when an error occurs. The subroutine is executed with a CALL 36815 ($8FCF).
Chapter 6. System Operation

It is useful to examine the details of a typical program run. Step by step description of each question-response pair will illustrate proper operation of the PC Board program.

The first question to be asked in Part I is, "CONFIGURE AS TWO DISK DRIVE SYSTEM ?" Suppose the reply is "N" for "NO". The immediate response is an instruction to the user, "INSERT DATA DISK, PRESS KEY." This informs the operator, on the supposition that only disk drive number 1 is operable, that if separate program and data disks are used, then the data disk (formatted) should be inserted. If the program and data disk are the same, then this instruction is not applicable.

The next query is, "VIEW OR MODIFY A PRINTED CIRCUIT BOARD PREVIOUSLY SAVED TO DISKETTE ?" Since it is assumed that this is the first run, the answer is "N" for "NO".

The next question is "HAVE THE LOCATIONS OF THE IC'S ALREADY BEEN STORED TO DISKETTE ?" Again, since this is the first run, the layout file has not been established, and the answer is "NO". The program then displays the catalog of the diskette and displays the words, "CREATE LAYOUT FILE, FILE NAME ?" This allows the user to view the file names already on the data diskette and avoid redundant file
nomenclature. The user enters an unused file name, and the program opens that file.

The High Resolution screen is cleared and switched on, and a question is displayed at the bottom of the screen, "THIS IC HAS HOW MANY PINS ?" The user may enter any valid number of pins from 2 to 40. At this point, the joystick must be approximately centered to prevent the error tone from sounding. The device is displayed to the screen, and may then be positioned under joystick control. Button zero of the joystick fixes the location of the device (Button one will erase the device). The program then performs error checking functions to prevent overlapping.

Assuming all is well, the program poses its next question, "ADD ANOTHER IC ?" If the user replies "YES", then program control transfers back to the previous question, "... HOW MANY PINS ?" The cycle would generally continue in this way until the user replied "NO" to the question "ADD ANOTHER IC ?" the program at this time writes the layout file to diskette and displays the five option menu of Part I, which was described in Chapter 3 Section 1.

Choosing item 1 in the menu, "GENERATE TRACES FOR PC BOARD" causes a check for the wire list on diskette and if none is present, generates the prompt, "INSERT PROGRAM
"DISK, PRESS KEY." When the key is pressed, Part II of the program is loaded into RAM, and the message "INSERT DATA DISK, PRESS KEY." is issued. Part I is thus complete. An example layout file is shown in Figure 6.2. This illustrates one possible layout file configuration for the 8085 system shown in Figure 6.1.

Part II of the PC Board program is best generated from a wire list written on paper. At the top of the page are column headings, each with the device identification. At the left side of the page are rows, each with a particular node number. The pin numbers for any device in the node are entered in the appropriate column and row. An example of this is shown in Figure 6.3, describing address and data lines for the 8085 project of Figure 6.1.

When Part II is run, a table is displayed. An inverse video field is defined for each device, and numbered. The entire table corresponds to a particular node. The node number is displayed at the bottom of the table. Likewise, near the bottom of the video screen are the words, "ESCAPE = NEXT NODE ARROWS/RETURN = EDIT".

The arrow keys should be pressed until the desired device is selected. To edit that device field, the carriage return is pressed. Once in the edit mode, the line near the bottom of the video screen reads, "PUT ' ' OR
FIGURE 6.1

Example Trace Routing of Small 8085 System Address and Data lines.
FIGURE 6.2

Example Layout File for 8085 System Shown in Figure 6.1.
<table>
<thead>
<tr>
<th>NODES</th>
<th>DEVICE 1</th>
<th>DEVICE 2</th>
<th>DEVICE 3</th>
<th>DEVICE 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8085</td>
<td>LS 373</td>
<td>2716</td>
<td>6116</td>
</tr>
<tr>
<td>1</td>
<td>AD0</td>
<td>12</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>AD1</td>
<td>13</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>AD2</td>
<td>14</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>AD3</td>
<td>15</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>AD4</td>
<td>16</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>AD5</td>
<td>17</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>AD6</td>
<td>18</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>AD7</td>
<td>19</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>A0</td>
<td>19</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td>A1</td>
<td>16</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>15</td>
<td>A2</td>
<td>15</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>A3</td>
<td>12</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>17</td>
<td>A4</td>
<td>9</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>18</td>
<td>A5</td>
<td>6</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>19</td>
<td>A6</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>A7</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>A8</td>
<td>21</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>22</td>
<td>A9</td>
<td>22</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>23</td>
<td>A10</td>
<td>23</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>24</td>
<td>GND</td>
<td>20</td>
<td>1,10</td>
<td>12</td>
</tr>
<tr>
<td>25</td>
<td>Vcc</td>
<td>40</td>
<td>20</td>
<td>24</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 6.3**

Example Wire List for 8085 System indicating format for easy data entry in Part II.
', 'BTWN PIN ARROWS/RETURN = EDIT". At this time, the pins of that node belonging to the selected device may be entered from the keyboard. The arrow keys may be pressed to allow freedom of movement within the field. To exit the edit mode, the carriage return is again pressed.

Subsequent devices may be selected and edited. When the node is completed, the escape key is pressed. The question is then asked, "IS THE WIRE LIST COMPLETE ?" If the answer is "N" for "NO", then a new table with cleared device fields is displayed for the next node. If the answer is "Y" for "YES", then the user is instructed, "INSERT PROGRAM DISK, PRESS KEY". When a key has been pressed, the program loads Part III into RAM. Part II is complete.

Upon execution of Part III, the instruction, "INSERT DATA DISK, PRESS KEY" is viewed. After insertion, the question is asked, "IS THIS A DOUBLE SIDED PC BOARD ?" Assume the user answers "Y" for "YES".

At this moment, the HIRES screen is cleared and the chips are displayed to the screen. The wire list is automatically accessed, and whichever pins are involved in the node are displayed as X's on the device. The words, "BEGIN NODE n ?" are displayed at the bottom of the screen.

If the answer is "YES", a joystick controlled
crosshair appears, along with the words, "POSITION JOYSTICK AT START OF TRACE. PRESS BUTTON 0". The crosshair may now be located at the position where the user desires. Ordinarily, this would be at one of the pins marked with an X. Upon pressing button zero, words appear beneath the HIRES screen, "WAIT = CONTINUE BUTTON 1 = RE-ENTER". This provides the user with a brief delay and the option to reposition the crosshair in the event that it is not exactly in the desired location. This is very helpful for erratic joysticks.

After the delay is over, and assuming the starting point is satisfactory, a new crosshair is drawn to the screen. The words at the bottom of the screen are similar to the earlier instruction, but read, "POSITION JOYSTICK AT END OF TRACE. PRESS BUTTON 0". Again, after the endpoint is located and BUTTON 0 is pressed, the words, "WAIT = CONTINUE BUTTON 1 = RE-ENTER" are issued. If the delay loop is allowed to time out, the word "ANALYZING" appears, indicating that routing procedures are being calculated. If the routing is successful, the new trace is displayed (along with any other routings). The 7 item menu described in Section 3 Part III is printed in single line format at the bottom of the screen. An instruction above the menu describes proper access to all menu items. "ARROWS SELECT
OPTION THEN PRESS RETURN". The meaning of each item is fully detailed in Section 3 Part III.

Events will continue in this manner. Any time a new node is initiated, program execution loops back to the question, "BEGIN NODE n ?" At some point, the end of the wire list will be reached. The user is asked to respond to the question, "EXAMINE OR ADD TO PREVIOUS NODES ?" This allows the freedom to return to nodes which were previously skipped, or incompletely routed. If the user answers "NO", then the PC board is deemed completed. Of course, even a completed board may be modified at a later time. Figures 6.4 and 6.5 are samples of Solder and Component side trace routings for the small 8085 system shown in Figure 6.1.

The next question is, "SAVE THE PRINTED CIRCUIT BOARD TO DISKETTE ?" The answer is "Y", for "YES". The Solder and Component Side are both saved to diskette. When complete, the final instruction of the program is printed, "INSERT PROGRAM DISK. HIT ANY KEY TO RESTART PROGRAM".
FIGURE 6.4
Solder Side Trace Routing for 8085 System Address and Data Lines.

FIGURE 6.5
Component Side Trace Routing for 8085 System Address and Data Lines.
Chapter 7. Conclusion

The Printed Circuit Board program was developed for the APPLE II PLUS microcomputer, although the fundamental principles and algorithms would apply to many microcomputer systems. The user can easily draw any small PC board with up to 40 devices.

Because of limited memory space and screen resolution, the program cannot handle large boards. The number of features in the program are also limited by available memory. Despite any inherent limitations due to system size, the PC Board program does afford the capability of trace routing to users who in the past would not have had the option.

The program provides all of the features described in Chapter 1. While the options are numerous, a number of characteristics could be added or improved. For example, some of the slower executing portions could be converted to machine language. It is even possible to convert the entire program to machine language so as to re-allocate memory, eliminate multiple byte variables, and provide enough additional memory to handle expanded features such as 3 pin devices, edgeboard connectors, header pads, single in- line packages, etc. It would be desirable to expand the board size by viewing only a section of it at a time. It would
likewise be a great benefit to develop an algorithm to automatically locate the starting and ending points of each routing.

Other changes in the program would be very nice, but may not be possible because of limitations in system hardware. The color of adjacent pixels on the screen is a good example. It would be an invaluable aid to the board designer to be able to distinguish solder and component side traces by color. Unfortunately, this is not possible, because colored graphic images do not have the same resolution as monochrome images. Parts of the colored traces would not appear on the screen.

It has likewise become obvious that a program characteristic which benefits the user at one instant, may hinder the user at a later time. For example, the ability to route a trace both horizontally and vertically on one side of a printed circuit board is very useful for single sided boards. However, it is perhaps undesirable for double sided boards. Many commercially available double sided boards are designed with all vertical traces on one side of the board, and all horizontal traces on the other side. This simplifies the routing procedure, and allows for more densely packed boards. This would require a plated through hole every time the direction of the routing was
changed, and would involve extensive program modification.

The only other major improvement which comes to mind, is the provision for starting and ending point crosshair control from either side of the board. At present, the starting and ending points are entered only from the solder side of the board. This is because the node markers (X's) are referenced to the solder side of the screen because the pin 1 location of each chip is known from that point of view. The limitation this imposes, is that a trace can be routed on the component side only if there is space enough on both sides of the board for a plated-through hole. It would be nice to be able to draw a trace from the component side without the need for plating through.

Program enhancements are always easy to imagine. Nevertheless, it is important to recognize the large number of options already available which greatly simplify the design process. This is a very powerful design tool, and when properly utilized, provides the engineer with the ability to optimize, archive, and modify any small printed circuit board.
APPENDIX A

Applesoft BASIC Commands and DOS System Parameters Utilized by the PC Board Program.
Appendix A

a) Memory Management

HIMEM—The highest memory location available for the BASIC program, including variables. HIMEM is used to protect the area of memory above it from accidental BASIC variable erasure. The default value for HIMEM allows enough protected memory for 256 byte file buffers below DOS. The PC Board program sets HIMEM at $8C1A (35866). This allows for additional protected memory beneath the file buffers for assembly language routines data, etc.

LOMEM—sets the address of the lowest memory location available to BASIC program variables. This is the memory address for the first BASIC variable. The default value for LOWMEM is automatically adjusted to point to a location just beyond the end of the BASIC program.

b) I/O

INPUT—This instruction allows keyboard entry of more than a single character during execution of a program and handles the input of real, integer or string variables. Program execution waits for the user to enter any keypresses assigned to the variable.

GET—This instruction obtains a single character from the keyboard. The computer waits for a key to be pressed, as in an INPUT statement. The character is not displayed and
does not require that the RETURN key be pressed.

CONTROL D—The string variable D$ by convention, is assigned the ASCII code for CTRL-D (D$=CHR$(4)). This means that D$ is the character corresponding to the ASCII code $04. In order to execute a DOS command from within a BASIC program, the instruction must print a carriage return, followed by a D$ and finally the DOS command. PR±—Transfers output to slot. Control is transferred to the program residing in the appropriate peripheral card ROM.

c) Graphics

HGR—sets the high-resolution graphics mode, clears the high-resolution screen to black, and leaves 4 lines for text at the bottom of the screen. In this mode, the display is made up of points on a grid that is 280 horizontal positions wide by 160 vertical positions high. HCOLOR—sets high resolution graphics color to that specified by the value of HCOLOR. There are 7 different colors. (O=black 1, 1=green, 2=blue, 3=white 1, 4=black 2, 5=depends on TV, 6=depends on TV, 7=white 2). PC Board program uses HCOLOR set equal to 3 (white) because only with HCOLOR equal to 3, are all specified pixels visible. If a color other than white were specified only every other line would be seen.
HPlot—plots a high-resolution dot in the color set with HColor.

XDraw—This statement will draw a specified shape using the complement of the color that already exists at that location on screen. If the existing color is white, then XDraw will plot in black. The Draw statement behaves much like the XDraw, except that the Draw command simply draws in the last HColor.

Scale—sets the size of the shape to be drawn by XDraw or Draw to a factor from 1 to 255.

Rot—sets the angular rotation for the shape to be drawn by XDraw or Draw. The amount of rotation is specified by a number between 0 to 255.

Pdl(0) or Pdl(1)—reads the current value of one of the game controls. (A number from 0 through 255). The value of the paddle(joystick) may be scaled by setting a ratio: full scale/255 = screen location/paddle reading.

d) Error Handling

Onerr Go To—When an error occurs, Onerr Go To may be used to avoid having an error message printed and execution halted. The command sets a flag that causes an unconditional jump (When a subsequent error occurs) to the program line indicated by line number.

e) Memory Access
PEEK—obtains the decimal value of the byte at a given memory location.

POKE—stores the given 8 bit value into the memory location whose address is specified.

f) Text Command

TEXT—This statement is used to return the screen to the standard alphanumeric display mode following the use of high or low resolution graphics.

VTAB and HTAB—are cursor moving commands, used to locate a predetermined position on the text screen. VTAB moves vertically. HTAB moves horizontally.

SPC( )—can be used in a PRINT statement to move the cursor X spaces to the right.

INVERSE—sets text display to the reverse video mode, so that characters on the screen are displayed as black letters on a white background.

NORMAL—reverts text display back to standard mode (white letters on a black background), with no flashing.

g) String Manipulation

LEFT$(A$)—returns (or supplies) the specified number of leftmost characters in the string enclosed in parenthesis. If no number is specified, it returns only the leftmost character in the string.

MID$(A$,N)—returns a substring at the Nth position of A$
to the end (last character) of A$.

RIGHT$(A$)—returns the string which contains the rightmost character of the string A$.

LEN(A$)—is equal to the length of the string A$.

STR$(A)—is the string equivalent of the numerical value A.

Strings can be added together (concatenated) to form longer strings.

VAL(A$)—is equal to the numerical equivalent of the string A$.

NULL STRING—PRINTing a NULL string on the terminal will cause no characters to be printed, and the cursor will not be advanced to the next column. The NULL string character is a string which has been defined and therefore resides within the string variable table. Although the pointers in the table have been allocated, the actual characters have not yet been assigned.

h) File Handling (DOS Commands)

1.) Sequential Text File Commands

OPEN—allocates a memory buffer of 595 bytes to the text file, and prepares the system to write or read from the beginning of the file. This command is used with WRITE and READ commands to create and retrieve sequential text files.

CLOSE—causes all remaining characters in the output part of the file buffer to be sent to the diskette specified
when that file was OPENed. CLOSE also deallocates the buffer associated with the sequential text file.

WRITE—After this command, PRINT statements send their output to the specified file instead of to the APPLE's TV screen. In other words, all APPLE output characters that would normally be displayed on the screen are sent to the diskette instead.

READ—After this command, INPUT and GET statements obtain their response characters from the specified sequential text file instead of from the APPLE's keyboard.

2.) Random Access Text File Commands

OPEN—allocates a 595 byte file buffer to the random access text file, and sets the record length to the number of bytes specified. The number must be in the range 1 to 32767. OPEN is used with the READ and WRITE commands to create and retrieve random access text files.

CLOSE—causes all remaining characters in the output part of the file buffer to be sent to the diskette in the drive that was specified when the file was OPENed. CLOSE de-allocates the buffer associated with the random access text file.

WRITE—After this statement, PRINT statements send their output to specified file instead of to the APPLE's TV screen. All APPLE output characters that would normally be
displayed on the screen are sent to the diskette.

READ—After this command, INPUT and GET statements obtain their response characters from the specified random access text file instead of from the APPLE's keyboard. INPUT causes characters to be READ from the random access file's current record, one field at a time.

3.) Assembly Language File Commands

BSAVE—creates a file and stores the contents of a segment of the APPLE's memory. The segment is specified by the starting address and the number of bytes to be stored.

BLOAD—returns the contents of a Binary file from disk to the APPLE's memory. BLOAD does not erase a BASIC program in memory, unless the data is BLOADed into the particular portion of memory containing the program.

BRUN—First BRUN does a BLOAD at the indicated location. The file is BLOADed starting at the same location from which it was BSAVEed. Once BLOADed, the file (which is an assembly language program) is started by a machine language jump (JMP) to the starting location.

i) Decisions

ON e GO TO L1, L2, L3 —Where e can be any BASIC expression and L1 L2 L3 — are line numbers. Valid values for e are integers 1, 2, 3 — n. If e=1, the computer goes to L1. If e=2, the computer goes to L2, and so on.
NOTE: An ON -- GO TO can replace several IF statements.

IF -- THEN -- if the condition stated between the words IF and THEN is false, then the instruction following the word THEN is ignored, and execution continues at the next line number.

j) Subroutine Commands

CALL—causes the computer to execute the machine language program beginning at a specified address. Parameters may be passed to the machine language program via a POKE to a common address location.

GOSUB—This causes the program to branch to a BASIC subroutine at the specified line number. When the subroutine is completed, a RETURN statement in the subroutine will return to execution of the program at the instruction following the most recently used GOSUB statement.
APPENDIX B

FLOWCHARTS
PART I

START

INITIALIZEATION
 SET HIMEM & DIMENSION VARIABLES

LOAD ASSEMBLY ROUTINES

CLEAR BOARD MATRIX

TWO DISK DRIVES?

YES
 SET NUMBER OF DISKS = 2

NO
 SET NUMBER OF DISKS = 1

VIEW PREVIOUSLY COMPLETED BOARD?

YES
 LOAD BOARD & VIEW IT

NO

a1
1. **CHECK WHETHER LAYOUT FILE EXISTS?**
   - **YES**
   - **NO**

2. **SET UP LAYOUT FILE**

3. **POSITION CHIP(S)**

4. **BOUNDARY CHECK**
   - **OVERLAP CHECK**

5. **CREATE RANDOM ACCESS FILE FOR LAYOUT FILE**

6. **DISPLAY CHIPS TO SCREEN**

7. **PRINT MENU FOR PART I**
CLEAR HI-RES & DRAW EPLOT
CLEAR LAYOUT MEMO
READ LAYOUT & DISPLAY
FILE CONTENTS TO TEXT SCREEN
WRITE LAYOUT TO MEMO
WRITE LAYOUT TO HI-RES SCREEN

IS FLAG = 2 ?

YES

f1

NO

IS FLAG = 3 ?

YES

d1

FIND THE IC WHICH USER WANT TO REPOSITION. & ERASE THAT IC.

NO

IS FLAG = 4 ?

YES

f1

PRESS Q TO QUIT DELETE FIND THE IC WHICH USER WANTS TO REMOVE. & WRITE FILE TO SCREEN. SHOW DELETED CHIP AS -----

NO

FLAG = 5

TURN ON HI-RES

c1
IS WIRE LIST FILE ON DISKETTE?  

ON WIRE LIST MENU SELECTION  
1. USE EXISTING WIRE LIST  
2. EDIT / EXAMINE WIRE LIST  
3. CREATE A NEW WIRE LIST

YES  

IF MENU SELECTION = 1  
GO TO PART III

NO

SET FLAG = 1

IF MENU SELECTION = 2

SET FLAG = 2

NO

SET FLAG = 1

MENU SELECTION = 3

NO

GO TO PART II

YES

IF MENU SELECTION = 1

NO

SET FLAG = 1
START

LOAD ASSEMBLY ROUTINE (MATRIX)  
CREATE SEQUENTIAL WIRE LIST

IS FLAG = 1 ?

YES

CREATE A NEW WIRE LIST
FILL STRINGS WITH BLANK SPACES

NO

FLAG = 2

USE EXISTING WIRE LIST.  
FILL STRINGS WITH PREVIOUS WIRE LIST PINS

EDIT / EXAMINE THIS NODE

PARSE EACH STRING ADD PINS TO NEW WIRE LIST

IS THE WIRE LIST DONE ?

YES

LOAD PART III

NO
PART III

START

LOAD ASSEMBLY ROUTINE (MATRIX)

IS NUMBER OF DISK DRIVE = 1 ?

YES → INSERT DATA DISK

NO → IS IT DOUBLE SIDED PC BOARD ?

YES → SIDE = 2

NO → SIDE = 1

SET UP FLAGS, VARIABLES ETC.

READ LAYOUT FILE. OBTAIN NUMBER OF PINS, & ROW & COLUMN OF EACH PIN 1.
READ 1 NODE OF WIRE LIST FROM WIRE LIST FILE

MANIPULATE STRINGS.
MARK EACH PIN OF NODE ON VIDEO SCREEN.

READ PADDLE DISPLAY CROSSHAIR OF STARTING POINT

CHECK BUTTON 0 PRESSED?

IF BUTTON 1 PRESSED, REDO

READ PADDLE DISPLAY CROSSHAIR OF ENDING POINT

CHECK BUTTON 0 PRESSED?

IF BUTTON 1 PRESSED, REDO
CLEAR FLAGS

EXECUTE COMPLEX ROUTING LOGIC

ERASE PREVIOUS ROUTING ATTEMPT FROM ARRAY CLEAR FLAG 1, BUT NOT OTHERS

WAS ROUTING COMPLETED?

SELECT MENU OPTIONS EXECUTE FUNCTION

SET FLAG 1
DID ROUTING GO AS FAR AS ENDPOINT COLUMN?

TRY TO GO AROUND

IS FLAG 1 SET?

TRY TO GO AROUND
YES

IS FLAG 2 SET?

NO

SET FLAG 2
SWITCH STARTING AND ENDING POINTS

YES

IS FLAG 3 SET?

NO

SET FLAG 3
CHANGE VARIABLE VALUE SO THAT NEXT ROUTING WILL NOT CHANGE DIRECTION AT MIDWAY (STEP= 90)

YES

IS FLAG 4 SET?

NO

SET FLAG 4
SWITCH STARTING AND ENDING POINTS
YES IS FLAG 5 SET?

NO

SET FLAG 5 CHANGE VARIABLE SO THAT NEXT ROUTING WILL CHANGE DIRECTION VERY EARLY (STEP = 3)

YES IS FLAG 6 SET?

NO

SET FLAG 6 SWITCH STARTING AND ENDING POINTS

YES IS FLAG 7 SET?

NO

SET FLAG 7 CHANGE VARIABLE SO THAT NEXT ROUTING WILL NOT START HORIZONTALLY
YES

IS FLAG 8 SET?

NO

SET UP FLAG 8
SWITCH STARTING AND
ENDING POINTS

YES

IS FLAG 9 SET?

NO

SET THIS DOUBLE SIDED BOARD?

YES

SET FLAG 9
SWITCH TO COMPONENT SIDE

PRINT ERROR MESSAGE MENU
AND TRACE IS NOT ROUTEABLE

a3
START OF MATRIX WRITE

START OF MATRIX READ

SET UP FLAG

SET UP FLAG

GET COORDINATES ETC. OF DESIRED ARRAY LOCATION

DETERMINE WHICH BYTE TO ACCESS

IS IT EVEN COLUMN (LOW NIBBLE)?

IF YES

IS FLAG SET FOR WRITE?

WRITE TO LOW NIBBLE

WRITE TO HIGH NIBBLE

If YES

IS FLAG SET FOR WRITE?

READ LOW NIBBLE

READ HIGH NIBBLE

RETURN TO CALLING PROGRAM (BASIC OR ASSEMBLY LANGUAGE)
ALL MOST RECENT TRACES ARE CONVERT TO LESS RECENT TRACES

READ SOLDER & COMPONENT SIDE ARRAY STARTING AT ROW 2 & LOOKING FOR LESS RECENT CORNER • ($2)

READ VALUE OF MATRIX AROUND THE $2 IN FOLLOWING ORDER : TOP, RIGHT, BOTTOM, LEFT.

ANY OF THESE LOCATIONS WHICH ARE "PERMANENT" ARE TREATED LIKE BLANKS (CONVERTED TO $0)
THUS $3, $6, $9 ARE READ AS $0
(ONLY $3, $6, $9 ARE DONE THIS WAY)

THE 4 CONVERTED VALUES ARE THEN CONCATENATED INTO 2 BYTES

C
LOOK UP TABLE & REPLACE THE $2 WITH APPROPRIATE CORNER

ALL LESS RECENT TRACES ARE THEN CHANGED PERMANENT TRACES

END
APPENDIX C

Program Listing
PART I LAYOUT

10 REM BASIC MATRIX FROM 30810-41300 & 1210-16000. MATRIX WHICH REA
DS/Writes the BASIC MATRIX AT 30300-40390. CLEAR WHICH CLEARS THE 'B
ASIC MATRIX' TO IERDES AT 30391-403AE.
25 REM ERROR RECOVERY ROUTINE AT 303AF-403BB. SCREEN SHOW EXAMINES E
ACH MATRIX LOCATION AND DRAWS IT ON SCREEN AT $-$. 
30 IF D2 < 0 THEN GOTO 25
40 HIMEM: 33866: REM 9801A
50 DIN CH(50), XH(50), YH(50), M$ (40)
70 CLEAR = 915; READ = 500; WRITE = 510; DIR = 515
90 D$ = "": REM CONTROL 3
90 SIDE = 0
100 GOTO 475
110 EC = PEEK (222)
120 PRINT DS$; CALL 943
120 PRINT DS$; "CLOSE"; Z$ 
180 IF EC < 9 THEN PRINT: PRINT DS$; "CLOSE"; M$
190 ON ER GOTO 1290, 2235, 1773, 1775, 1780, 1400, 1650, 2150, 2040
202 GOTO 10
208 EC = PEEK (222): POKE 215, 0
210 IF EC = 6 THEN GOTO 2045
211 IF EC = 10 THEN PRINT: PRINT DS$; "UNLOCK"; M$: DT; D2: PRINT DS$; GOTO 2015
212 PRINT "ERROR NUMBER"; EC: GOTO 212
220 POKE 27, X: POKE 28, Y: CALL 768: A = PEEK (26): RETURN
27, X; POKE 28, Y: CALL 775: GOSUB 225: RETURN
230 POKE 36437, 22: POKE 36440, 3: POKE 36528, 38: POKE 36529, 143: POKE 78
7, 3: POKE 916, 8: POKE 923, 19, POKE 932, 19: RETURN: REM SWITCH TO
COMPONENT SIDE
CH TO SOLDER SIDE
260 A = PEEK (49152)
267 IF SIDE = 2 THEN SIDE = SIDE: IF A = 155 THEN AA = PEEK (787): IF
36536: RETURN
268 IF SIDE = 2 THEN SIDE = SIDE: IF A = 155 THEN GOSUB 220: GOSUB 270
241 CALL 62450: HSR: GOSUB 445: CALL 36536
270 RETURN
270 A = PEEK (787): IF A = 8 THEN AS = "COMPONENT SIDE": RETURN
271 AS = "SOLDER SIDE": RETURN
280 HOME: PRINT "THE CATALOG LISTING IS": PRINT D$: "CATLOGS.D": DT: D2: RETURN
281 Wy = 1: PRINT "Y": INVERSE: PRINT "N": NORMAL: PRINT CHR$ (8): 
CHR$ (8) & CHR$ (8): GOTO 285
284 Wy = 0: INVERSE: PRINT "Y": NORMAL: PRINT "N": CHR$ (8) & CHR$ (8)
CHR$ (8) & CHR$ (8): GOTO 285
288 F$ = PEEK (49152): IF F$ < 127 THEN GOTO 286
287 F$ = CHR$ (F)
288 IF F$ = CHR$ (217) THEN F$ = "Y": INVERSE: PRINT "Y": NORMAL: PRINT
"N": POKE 49168, 0: RETURN
289 IF F$ = CHR$ (204) THEN F$ = "N": INVERSE: PRINT "N": NORMAL: POKE
49168, 0: RETURN
290 IF F$ = CHR$ (141) THEN F$ = F$: IF Wy = 0 THEN F$ = CHR$ (217): GOTO
286
291 IF F$ = CHR$ (141) THEN F$ = F$: IF Wy = 1 THEN F$ = CHR$ (204): GOTO
289
292 IF F$ = CHR$ (136) THEN F$ = CHR$ (217): GOTO 286.
IF FS = CHR$(149) THEN FS = CHR$(206): GOTO 289
294 FS = "\": POKE 49168,0: PRINT: RETURN
295 FOR DY = 1 TO 400: NEXT: RETURN
300 SS = 0: SQ = 0: FOR I = 0 TO PN - 1: XDRAW S AT Y * 5,3 * (X + 2 * I) &
            IF SQ = 1 THEN SQ = 1: GOTO 1
310 IF SS > 128 THEN SQ = 1
320 NEXT: IF SS = 1 THEN SQ = 1: GOTO 400
370 IF SQ = 1 THEN GOTO 255
320 GOTO 910
400 FOR L = 0 TO PN - 1: XDRAW S AT Y * 5,3 * (X + 2 * L) &
            SW = PEEK (49250): IF SW > 128 THEN SQ = 1
410 NEXT: IF SQ = 1 THEN GOTO 1180
420 GOTO 1060
430 HOME: VTAB 22: F = 1: PRINT "ARROWS SELECT OPTION, THEN PRESS ": INVERSE
            PRINT "RETURN": NORMAL
            PRINT M$(F): NORMAL: POKE 49168,0
434 IF DL = 0 THEN VTAB 8: FOR I = 1 TO MAX: HTAB 1: PRINT M$(I): NEXT
            : VTAB 7 + F: HTAB 1: INVERSE: PRINT M$(F): NORMAL: POKE 49168,0
436 IC = PEEK (49152): IF IC < 127 THEN GOTO 434
438 FS = CHR$(IC): IF FS = CHR$(141) THEN POKE 49168,0: RETURN
440 IF FS = CHR$(136) THEN F = F - 1: IF F < 0 THEN F = MAX
442 IF FS = CHR$(149) THEN F = F - 1: IF F > MAX THEN F = 1
450 GOTO 422
444 HPLCT 0.0 TO 274.0 TO 274.159 TO 0.159 TO 0.0: RETURN
460 CALL CLEAR: GOSUB 250: CALL CLEAR: GOSUB 265: RETURN
470 PRINT D$: "LOAD MOST ROUTINES.OBJ,A$8C18,D1": POKE 232,97: POKE 232,142
472 IF D2 = 0 THEN TEXT: HOME: VTAB 10: PRINT "CONFIGURE AS TWO DISK
            DRIVE SYSTEM? ": GOSUB 284: IF FS < "Y" THEN HOME: VTAB 10:
            PRINT "INSERT DATA DISK IN DRIVE 2, PRESS KEY": SET V$,
474 IF D2 = 0 THEN D2 = D2: IF FS < "N" THEN PRINT "INVALID ENTRY":
            CALL 36815: GOSUB 295: GOTO 476
476 IF D2 = 0 THEN D2 = 1
480 IF D2 = 1 THEN TEXT: HOME: VTAB 10: PRINT "INSERT DATA DISK, PRE
            VIOUSLY SAVEd TO DISKETTE? ": GOSUB 281
500 GOSUB 265: POKE 230,22: M$COLOR 3: IF FS = "Y" THEN HOME: GOSUB 2
            500: INVERSE: PRINT "VIEW/MODIFY": NORMAL: INPUT "FILE NAME"; V$;
            HOME
510 IF FS = "Y" THEN PRINT: PRINT D$: "LOAD".; V$:.,A$8010.D$; D2: PRINT
            SIDE = 2 THEN HOME: VTAB 22: PRINT "PRESS ESCAPE TO FLIP PC BOARD
            TO CONTINUE "; GET FS; GOTO 511
511 POKE 49168,0
512 IF SIDE = 2 THEN SIDE = SIDE: IF FS = "Y" THEN F = PEEK (49152): IF
            F < 127 THEN GOTO 512
515 IF SIDE = 2 THEN SIDE = SIDE: IF FS = "Y" THEN X = FRE (0): IF F <
            128 THEN GOTO 517
518 IF SIDE = 2 THEN SIDE = SIDE: IF FS = "Y" THEN GOSUB 266: GOTO 511
515 IF FS = "Y" THEN HOME: VTAB 22: PRINT "PRESS ANY KEY
            TO CONTINUE ": SET FS: GOTO 517
520 IF FS < "N" THEN PRINT "INVALID ENTRY": CALL 36815: GOSUB 295: GOTO
            490
520 GOTO 520
527 TEXT: HOME: VTAB 20: PRINT SPC(16): FLASH: PRINT "CAUTION": NORMAL
            PRINT "ANSWER YES ONLY IF LAYOUT FILE DEVICES": PRINT "WE'RE NOT R
            EPOSITIONED AFTER SAVING BOARD": VTAB 10: PRINT "ADD TRACES TO THE
            BOARD JUST SEEN? ": GOSUB 281: IF FS = "Y" THEN I = 0: GOTO 540
530 IF FS < "N" THEN PRINT "INVALID ENTRY": CALL 36815: GOSUB 295: GOTO
            537
537 GOTO 490
540 I = I + 1: IF I < = LEN (VS) THEN I = I: IF MID$(VS,1,1) < > ".
"THEN GOTO 540  
541 IF I > LEN (V$) THEN HOME: VTAB 10: PRINT V$: "FILE NAME ERROR":  
CALL 36815: GOSUB 295: GOTO 490  
542 I$ = LEFT$ (V$, I - 1) : W$ = "", "WIRE":"; "; Z$: "D": D$: PRINT D$: PRINT D$: "; "W$: "D"; D$: PRINT D$: ROT = 0: SCALE = 1: HCOLOR = 3:  
GOSUB 245: GOTO 2080  
544 CALL 945: VTAB 10: PRINT "OUTPUT OR WIRE LIST NOT ON DATA DISK": PRINT "CANNOT CONTINUE": CALL 36815: GOSUB 295: GOTO 490  
550 TEXT : HOME: VTAB 10: PRINT "HAVE THE LOCATIONS OF THE IC'S ALREADY Y SEEN STORED TO DISKETTE?":: GOSUB 284  
559 IF F$ < > "N" THEN PRINT "INVALID ENTRY": CALL 36815: GOSUB 295: GOTO 550  
564 HOME : GOSUB 22: INPUT "THIS IC HAS HOW MANY PINS?"; P$: P$ = VAL (P$): P$ = P / 2  
565 IF P = 2 THEN W = 4$ = 1: GOTO 805  
566 IF P = 4 THEN W = 4$ = 1: GOTO 805  
567 IF P = 1 THEN W = 4$ = 1: GOTO 805  
568 IF P = 2 THEN W = 4$ = 1: GOTO 805  
569 IF P = 3 THEN W = 4$ = 1: GOTO 805  
570 IF P = 4 THEN W = 4$ = 1: GOTO 805  
571 IF P = 2 THEN W = 4$ = 1: GOTO 805  
572 IF P = 3 THEN W = 4$ = 1: GOTO 805  
573 IF P = 4 THEN W = 4$ = 1: GOTO 805  
574 IF P = 2 THEN W = 4$ = 1: GOTO 805  
575 IF P = 3 THEN W = 4$ = 1: GOTO 805  
576 IF P = 4 THEN W = 4$ = 1: GOTO 805  
577 IF P = 2 THEN W = 4$ = 1: GOTO 805  
578 IF P = 3 THEN W = 4$ = 1: GOTO 805  
579 PRINT "INVALID ENTRY": CALL 36815: GOSUB 295: GOTO 640  
305 HOME  
320 VTAB = 1; PRINT "POSITION THE JOYSTICK LOCATING PIN 1": VTAB = 2; PRINT  
"BUTTON 0 = ENTER / BUTTON 1 = QUIT": Y = 276 / 255 * PDL (0) / J1X  
= 134 / 255 * PDL (1) / J1Y = INT (Y) J1X = INT (X)  
833 IF X < 2 THEN CALL 36815: GOTO 805  
850 IF X > 52 - 2 * PN THEN CALL 36815: GOTO 805  
863 IF Y > 88 THEN CALL 36815: GOTO 805  
880 IF Y < W + 4 THEN CALL 36815: GOTO 805  
995 ROT = 0: SCALE = 1: GOTO 310  
910 SX = 0: FOR I = 0 TO PN - 1: IDRAW S AT Y = J1Y + 2 * I: SA = PEEK (49250): IF SA > 128 THEN SX = 1  
926 NEXT : IF SX = 1 THEN GOTO 1180  
940 GOTO 280  
953 X1 = X: Y1 = Y  
970 HOME : VTAB = 22: PRINT SPC (13); "ANALYZING":  
9769 FOR I = 1 TO (PN - 1) * 2 + 3  
1000 FOR J = 1 TO W + 7  
1015 S = (XI + 1) - 2*Y = (Y1 - J) + 4: GOSUB 220.  
1030 IF A = 10 THEN HOME : VTAB = 22: PRINT SPC (13); "OVERLAP ERROR": CALL 36815: GOSUB 295Y = Y1: X = X1: SQ = 0: IF PEEK (49250) > 128 THEN  
SQ = 1  
1043 IF A = 10 GOTO 400  
1060 IF A = 10 THEN GOTO 820  
1073 NEXT : NEXT  
1090 FOR I = 1 TO PN  
1103 X = (I - 1) * 2*Y = Y1:A = 10: GOSUB 223: Y = Y1 - W$: GOSUB 233  
1 NEXT  
1120 IF ER = 5 THEN GOTO 1975  
1123 PRINT D$: "WRITE": $$; R$: R$: PRINT P$: R$ = R$ + 1  
1126 PRINT D$: "WRITE": $$; R$: R$: PRINT Y$: R$ = R$ + 1: PRINT D$:  
1180 HOME : VTAB = 22: PRINT SPC (10); PRINT "ADD ANOTHER IC?": GOSUB
IF FS = "Y" THEN GOTO 640

1210 IF FS < "X" THEN PRINT "INVALID RESPONSE": CALL 36815: GOSUB 2

1225 PRINT C$:"CLOSE":Z

1240 TEXT :0L = 0:MAX = 5:M$(1) = "1.") GENERATE TRACES FOR PC BOARD":M$(2) = "2.") EXAMINE THE PIN PLACEMENTS":M$(3) = "3.") REPOSITION A CD COMPONENT ON PC BOARD":M$(4) = "4.") REMOVE A COMPONENT FROM PC BOARD"

1250 M$(5) = "5.") ADD A COMPONENT TO THE LAYOUT":SIDE = 0: GOSUB 431.

1300 ON F GOTO 2005,1215,1750,1345,1540

1315 ER = 4: ONERR GOTO 115

1320 CALL 42450; GOSUB 445; ROT = 0; SCALE = 1; HCOLOR = 3; GOTO 1155

1345 CALL 642450; GOSUB 445; ROT = 0; SCALE = 1; HCOLOR = 3; ER = 1: ONERR GOTO 115

1360 GOTO 1355

1375 PRINT "PRESS A KEY TO VIEW LAYOUT": GET FS; HOME; POKE 49222,1; POKE 49229,1; VTAB 221; PRINT "PRESS A KEY TO RETURN TO MENU"; GET FS; GOTO 1240

1390 PRINT "PRESS A KEY TO QUIT DELETING A D. IF FS = "G" THEN GOTO 1240

1405 ER = 2: ONERR GOTO 115

1420 INPUT "REMOVE WHICH IC NUMBER? ":M$;M = VAL (M$)

1435 IF M < 1 THEN PRINT "INVALID ENTRY": CALL 36815; GOTO 1405

1450 IF M > I / 3 THEN PRINT "INVALID ENTRY": CALL 36815; GOTO 1405

1465 M$ = "J" M = 1; PRINT D$:"OPEN":Z$;L4,D;L2:GOSUB 445; PRINT D$:"OPEN":Z$;L4,D;L2:PRINT D$:"READ":Z$;R;R1: INPUT P$;P = VAL (P$);PN = P / 2: I = I + 12; I = 6:1 = 1

1460 PRINT D$:"READ":Z$;"",R;R1: INPUT R$;X = VAL (R$);I = I + 11:RN = I

1480 IF ER = 7 THEN GOTO 1660

1565 PRINT D$:"READ":Z$;"",R;R1: INPUT R$;X = VAL (R$);I = I + 11:RN = I

1585 PRINT D$:"READ":Z$;"",R;R1: INPUT R$;X = VAL (R$);I = I + 11:RN = I

1620 IF ER = 1 THEN GOTO 1585

1675 IF CN = "---" THEN GOTO 1585

1705 X = X1;Y = Y1;IF P > 22 THEN W = 12;S = 2

1720 FOR J = 1 TO PN:X = X + 1;Y = Y + 1;I = I + 1: X = X + 1;Y = Y + 1;PRINT D$:"OPEN":Z$;L4,D;D2; POKE 49222,1; POKE 49229,1; HCOLOR = 3; GOTO 640

1750 ER = 5: ONERR GOTO 115

1765 GOTO 1320

1770 INPUT "REPOSITION WHICH IC NUMBER? ":M$;M = VAL (M$)

1795 IF M < 1 THEN PRINT "INVALID ENTRY": CALL 36815; GOTO 1780

1810 IF M > I / 3 THEN PRINT "INVALID ENTRY": CALL 36815; GOTO 1780

1825 MM = 3:MrAN = MM - 3:PRINT D$:"OPEN":Z$;L4,D;D2:PRINT D$:"READ":Z$;R;R1:INPUT T$;X = VAL (T$);PN = P / 2:RN = RN + 1

1840 PRINT D$:"READ":Z$;"",R;R1: INPUT T$;X = VAL (T$);RN = RN + 1

1865 PRINT D$:"CLOSE":Z$; PRINT CHR$ (13)

1880 IF Q$ = "---" THEN HOME; VTab 10; PRINT "A DELETED IC CANNOT BE REPOSITIONED": GOSUB 295; GOTO 1765

1900 IF Q$ = "---" THEN HOME; VTab 10; PRINT "A DELETED IC CANNOT BE
REPOSITIONED": GOSUB 295; GOTO 1745
1930 IF US = "~~" THEN HOME; VTAB 10; PRINT "A DELETED FC CANNOT BE
REPOSITIONED": GOSUB 295; GOTO 1745
1940 W = LiS: 1:XI = (J - 1) I X2; Y Y = Y; IF P > 22 THEN W = 12; S = 2
1960 FOR J = 1 TO PX: X = (J - 1) I X2: Y = Y; I = 2: GOSUB 2231:Y = Y
I - W: GOSUB 2231: NEXT: POKE 49222,1; POKE 49231,1; POKE 49239,1: GOSUB
295; FOR I = 0 TO RN - 1: XDRAW S AT Y1 # J, S # XI # J I: NEXT
1970 ; GOTO 305
1975 TEXT : PRINT D$:"OPEN";Z$;","D$:"WRITE";Z$;","S$;","MM =
2: PRINT XI
1990 PRINT D$:"WRITE";Z$;","R$;","MM - I: PRINT YI: PRINT D$:"CLOSE";Z$;
S$;"GOTO 115
2000 SGRALL = 1: ONERR GOTO 208
2010 N$ = "$ = ",WIRE$: PRINT : PRINT D$:"RENAME ";N$: ",W$";Z$;",D$; PRINT
DS
2025 SL = 0:MAX = J:MS(1) = ",I.: USE EXISTING WIRE LIST";MS(2) = ",I.: ED
IT/EXAMINE WIRE LIST";MS(3) = ",I.: CREATE A NEW WIRE LIST"; GOSUB 4
J: ON F GOTO 2040,2050,2065
2040 IF D$ = 1 THEN HOME; VTAB 10: PRINT "INSERT PROGRAM DISK, PRESS
KEY": GOTO 1
2045 PRINT : PRINT D$:"LOAD CHAIN. AS$20,01": CALL 550"PART III"
2050 SGRALL = FLAG = 1
2065 IF D$ = 1 THEN HOME; VTAB 10: PRINT "INSERT PROGRAM DISK, PRESS
KEY": GOTO 1
2070 PRINT : PRINT D$:"LOAD CHAIN. AS$20,01": CALL 550"PART II"
2080 ; HOME : VTAB 10: PRINT SPEC ":15:";"ANALYZING:"ER = 8:RN = 0: ONERR GOTO
115
2090 PRINT D$:"OPEN";Z$;","L$";
2100 PRINT D$:"READ";Z$;","R$;";RN: INPUT G$;I = RN / 3 + 1:RN = RN + 1:CH
(1) = VAL (G$)
2110 PRINT D$:"READ";Z$;","R$;";RN: INPUT G$;RN = RN + 1:XH(I) = VAL (G$)
2120 PRINT D$:"READ";Z$;","R$;";INPUT G$;RN = RN + 1:YH(I) = VAL (G$)
2130 : GOTO 2100
2140 ER = 9: PRINT : PRINT D$:"OPEN";N$: ONERR GOTO 115
2140 PRINT : PRINT D$:"READ";N$: INPUT SL$: PRINT D$:;J = 1:KI = 0
2150 LS = **"BLS" = **
2160 KI = XI + 1: IF KI > LEN (SL$) THEN GOSUB 2220: GOTO 2140
2170 LS = MID$ (SL$,KI,1): IF LS = CHR$ (46) THEN GOSUB 2220: GOTO 21
30
2180 IF LS = CHR$ (47) THEN J = J + 1: GOSUB 2220: GOTO 2150
2190 IF LS = CHR$ (45) THEN J = J + 1:KI = KI + 1: GOTO 2150
2200 BLS = BLS + LS: GOTO 2160
2220 JJ = J: IF LS = CHR$ (47) THEN JJ = J - 1
2230 IF CH(JJ) = 0 THEN RETURN
2240 RN = CH(JJ) / 2: P = VAL (BLS$): IF P > RN THEN GOTO 2270
2250 X = XH(JJ) + (CH(JJ) - P) # Z; Y = YH(JJ) - 12
2260 IF CH(JJ) < 22 THEN Y = YH(JJ) - 6
2290 GOSUB 220: IF A = 10 THEN RETURN
2300 GOTO 2260
PART II WIRE LIST

10 REM THE "BASIC MATRIX" IS AT $82510-$9000. THE MATRIX SUBROUTINE, WHICH READS OR WRITES FROM THE "BASIC MATRIX," IS LOCATED AT $80300-$80390. THE CLEAR SUBROUTINE, WHICH CLEARS THE "BASIC MATRIX" TO ZERO ES, IS AT $80271-$802AE.

110 REM THE ERROR RECOVERY ROUTINE OCCUPIES $03AF-$03BE. THE DISPLAY SUBROUTINE, WHICH EXAMINES EACH MATRIX LOCATION AND DRAWS IT TO THE SCREEN IS LOCATED AT $-

112 D$ = "REM CONTROL D" 210 PRINT: PRINT D$:"LOAD MATRIX, AS0300,DI" 510 IF D$ = 1 THEN HOME: VTAB 10: PRINT "INSERT DATA DISK, PRESS KEY" : GET Fs:PRINT CHR$(13)

510 GOTO 2810
610 EC = PEEK(222)
710 PRINT D$: CALL 943
720 IF ER = 2 THEN ER = ER: IF EC = 5 THEN HOME: VTAB 10: PRINT "END OF EXISTING WIRE LIST": PRINT "ADD ANY OTHER NODES? ": GOSUB 1410:

721 IF ER = 2 THEN ER = ER: IF EC = 5 THEN EC = EC: IF F$ < "Y" THEN SOTD 720
722 IF ER = 2 THEN ER = ER: IF EC = 5 THEN FLAG = 0: GOTO 3915
810 IF EC = 176 THEN HOME: VTAB 10: PRINT "THIS NODE IS TOO LARGE TO HANDLE": PRINT "PLEASE ENTER AS TWO SEPARATE NODES": GOSUB 2410: J = 1: GOSUB 7105: GOTO 4110

910 PRINT "ERROR NUMBER":EC 1010 PRINT D$:"CLOSE":" 1110 ON ER SETO 3:49110 1210 GOTO 10
1210 HOME$: PRINT "THE CATALOG LISTING IS;": PRINT D$:"CATALOG,D;D2: RETURN

1410 WY = 1: PRINT "Y/1": INVERSE: PRINT "N": NORMAL: PRINT CHR$(B): CHR$(8): CHRS$(B): GOTO 1410
1510 WY = 0: INVERSE: PRINT "Y/1": NORMAL: PRINT "/N": CHRS$(B): CHRS$(B): 1610 POKE 49168,0 1710 IF F$ = POKE (49152): IF F$ < 127 THEN GOTO 1710
1810 FS = CHRS$(F)
1910 IF FS = CHRS$(217): THEN FS = "Y": INVERSE: PRINT "Y": NORMAL: PRINT "N": POKE 45168,0: RETURN
2010 IF FS = CHRS$(206): THEN FS = "N": PRINT "Y/1": INVERSE: PRINT "N": NORMAL: POKE 45168,0: RETURN
2110 IF FS = CHRS$(141) THEN F$ = F$: IF WY = 0 THEN FS = CHRS$(217): GOTO 1910
2210 IF FS = CHRS$(141) THEN F$ = F$: IF WY! = 1 THEN FS = CHRS$(206): GOTO 2310
2310 IF FS = CHRS$(136) THEN FS = CHRS$(217): GOTO 1910
2410 IF FS = CHRS$(149) THEN FS = CHRS$(206): GOTO 2010
2510 FS = "G": POKE 49168,0: PRINT : RETURN
2610 FOR BY = 1 TO 4000: NEXT : RETURN
2612 FOR L = 1 TO 401MS(L) = 0: NEXT: FOR L = 1 TO IC1:MS(L) = 1 FOR L = 2 TO IC1:MS(L) = MS(L):MS(L) = LC(L): FOR L = 2 TO IC1:MS(L) = MS(L):MS(L-1) = NEXT 2620 ER = 2: GOSUB GOTO 610
2630 PRINT: PRINT D$:"READ":IN$: INPUT SL$: PRINT D$:J = INT IN$: 2640 PLS = "";BLS = "" 2650 KI = KI + 1: IF KI > LEN (SL$) THEN GOSUB 2700: FOR L = 1 TO IC1:MS(L) = 1 FOR L = 2 TO IC1:MS(L) = MS(L):MS(L) = LC(L): FOR L = 2 TO IC1:MS(L) = MS(L):MS(L-1) = NEXT 2660 L$ = MID$(SL$,KI,1): IF L$ = CHRS$(46) THEN GOSUB 2700: GOTO 26
INVERSE : PRINT MS(FX): NORMAL

2510 IF OL = 0 THEN FOR J = 1 TO NCC; VTAB 1: WS: \( (j - 1) \) : INT (40 / NCC): FOR I = 1 TO MAX: QR = INT ((C = 1) \* MAX = 1): POKE CS,WH: PRINT MS(QR): NEXT I; NEXT J


5710 POKE 49168, 0

5810 F = PEEK (49132); IF F < 127 THEN GOTO 5910

5910 FS = CHR (F); IF FS = CHR (141) THEN PEEK 49168,0: RETURN

6190 IF FS = CHR (155) THEN POKE 49168,0: POP : NODE = NODE + 1: GOTO 7015

6110 IF FS = CHR (136) THEN FX = FX + 1; IF FX < (COL - 1) \* MAX THEN COL = COL + 1: GOTO 6110

6115 IF FS = CHR (132) THEN GOTO 6810

6210 IF FS = CHR (149) THEN FX = FX + 1; IF FX > COL \* MAX THEN COL = COL + 1: GOTO 6110

6215 IF FS = CHR (149) THEN GOTO 6810

6310 IF LEFTS (MS(FX),J) = " " THEN GOTO 6110

6315 PRINT " **": GOTO 6140: REM CONTROL G

6410 IF COL < 0 THEN COL = NCC:FX = MAX = NCC: GOTO 6810

6510 FX = INT ((COL = 1) \* MAX = 1): POKE 6D,WH: INVERSE : PRINT MS(FX): NEXT I; NEXT J

6610 IF COL = NCC THEN COL = 1: FX = 1: GOTO 6810

6710 FX = INT ((COL = 1) \* MAX = 1): GOTO 6810

6810 IF LEFTS (MS(FX),J) = " " THEN GOTO 6110

6910 GOTO 5410: GOTO 5410

7015 HOME : VTAB 10: PRINT SPC (15): "ANALYZING": FOR J = 1 TO IC; MS(J) = RIGHTS (MS(J),16): NEXT J; MS(J) = " "

7020 T$ = "": QS = "": I = J + 1: IF CH(J) = 0 THEN GOSUB 7080

7025 I = I + 1: FS = MIDS (MS(J),-1,1): IF FS > " ": THEN I = I: IF FS < " ": THEN QS = "": I = J + 1: IF CH(J) = 0 THEN GOSUB 7080

7040 IF QS = " " THEN I = I: IF I < 16 THEN GOTO 7030

7045 I = I + 16 THEN I = I: IF T$ = " ": THEN GOS = " ": GOSUB 7080

7050 L = VAL (QS): IF L < 1 THEN GOTO 11: PRINT " ITEM "; J$: PIN ZERO N

7060 IF L > CH(J) THEN VTAB 1: PRINT " ITEM "; J$: PIN ONLY "CH(J)": P

7070 G$ = STRS (L); GS = G$ = CHR (46)

7080 SL$ = SL$ + G$ + QS

7090 IF LEN (SL$) > 250 THEN VTAB 11: PRINT " THIS NODE IS TOO LARGE." HANDLE: PRINT "PLEASE ENTER AS TWO SEPARATE NODES": GOSUB 2610: J = 1:

7095 G$ = "": GOSUB 7102

7100 IF I < 16 THEN G$ = "": GOSUB 7030

7105 SL$ = LEFTS (SL$, LEN (SL$) - 1)

7106 SL$ = SL$ = CHR (47)

7110 IF J < IC THEN GOTO 7020

7115 S$ = LEFTS (SL$, LEN (SL$) - 1): GOTO 7110

7120 FOR L = 1 TO IC; MS(L) = " ": STRS (L) = " ": MS(L) = RIGHTS (MS(L),19): NEXT I; FX = J:IC = 1: IF J > MAX OR 0 THEN COL = 2

7125 RETURN

7130 PRINT DS; "WRITE"; WS; PRINT SL$; PRINT DS

7135 ON FLAG GOTO 7310, 3915

7510 HOME : VTAB 10: PRINT " IS THE WIRING LIST COMPLETE? "; GOSUB 1410

7410 IF FS = "y" THEN PRINT : PRINT DS; "CLOSE"; WS; PRINT : PRINT DS ; DELETE"; WS; ," ,D; D1: PRINT : PRINT DS; "RENAME"; WS; ," ,D; D2: GOTO 7710

7510 GOTO 3915

7710 IF D2 = 1 THEN HOME : VTAB 10: PRINT "INSERT PROGRAM DISK, PRESS KEY": GET FS

7720 PRINT : PRINT DS; "BLDCHAIN, A520, D1"

7810 CALL 320: "PART III"

7310 IF D2 = 1 THEN HOME : VTAB 10: PRINT "INSERT PROGRAM DISK": PRINT "PRESS ANY KEY TO RESTART": GET FS
7830 PRINT D$: "LOAD CHAIN, A520,01": CALL 520 "PART I"
PART III TRACE ROUTING

10 X = FRED (0):DS = "":BS = "": REM CONTROL D AND G
20 PRINT: PRINT DS: "LOAD MATRIX. A@0500,21"
30 IF D1 = 1 THEN HOME: VTAB 10: PRINT "INSERT DATA DISK, PRESS KEY": SET P$
40 GOTO 690
50 EC = PEEK (222)
60 PRINT DS: CALL 943
70 IF EC = 4 THEN GOTO 2260
80 IF EC = 5 THEN GOTO 90
90 PRINT "ERROR NUMBER":EC: GOSUB 570
100 ON ER GOTO 2420, 770, 2170, 2240
110 POKE 27:X; POKE 28,Y; CALL 768:A = PEEK (26): RETURN
120 POKE 26,A; POKE 27:X; POKE 28,Y; CALL 777: RETURN
130 A = PEEK (4912)
140 IF SIDE = 2 THEN SIDE = SIDE: IF A = 155 THEN AA = PEEK (787): IF
A8; NORMAL : POKE 56,39: VTAB 24: CALL 32450: HGR : GOSUB 600: CALL
3652: RETURN
150 IF SIDE = 2 THEN SIDE = SIDE: IF A = 155 THEN GOSUB 580; GOSUB 550 :
VTAB 21: POKE 56,25; INVERSE : PRINT A8; NORMAL : POKE 36,0: VTAB
24: CALL 32450: HGR : GOSUB 600: CALL 3652
160 RETURN
161 IF XF = XG THEN XF = XF; IF YF = YQ THEN YV = 15: RETURN
162 BY = 15: RETURN
163 GOTO 162
170 LX = 0:IX = XI; Y = YI: IF LST = 1 THEN GOSUB 590; GOSUB 110: IF A <
> 3 THEN A = A; IF A < 4 THEN A = A; IF A < 8.9 OR A > 14.1 OR
A = 10 THEN GOSUB 240; GOSUB 590; GOSUB 550: GOSUB 240: LX = 1:IX = XI:
Y = YI: GOSUB 110: IF A = 10 THEN LX = 2
180 LY = 0:IX = XI; Y = YF; IF LST = 1 THEN GOSUB 590; GOSUB 110: IF A <
> 3 THEN A = A; IF A < 4 THEN A = A; IF A < 8.9 OR A > 14.1 OR
A = 10 THEN GOSUB 240; GOSUB 590; GOSUB 550: GOSUB 240: LY = 1:IX = XI:
Y = YF: GOSUB 110: IF A = 10 THEN LY = 2
190 IF LX = 1 THEN LX = LX: IF LY = 1 THEN GOSUB 580:X = XI; Y = YI; A =
15: GOSUB 120; GOSUB 590:X = XI; Y = YI; A = 15: GOSUB 120; GOSUB 580
X = XF; Y = YF; A = 15: GOSUB 120: GOSUB 590: X = XF; Y = YF; A = 15:
GOSUB 120; GOSUB 580: X = XF; Y = YF; A = 15: GOSUB 120: GOSUB 161
RETURN
210 IF LX = 1 THEN LX = LX: IF LY = 2 THEN GOSUB 580:X = XI; Y = YI; A =
15: GOSUB 120; GOSUB 590:X = XI; Y = YI; A = 15: GOSUB 120; GOSUB 164
RETURN
215 IF LX = 2 THEN LX = LX: IF LY = 2 THEN GOSUB 590; RETURN
220 IF LST = 1 THEN HOME: VTAB 22: PRINT "PLATED HOLE NOT POSSIBLE AT
ENDPOINT": CALL 36815; GOSUB 570: POP : GOSUB 580: GOSUB 640: GOSUB
590; GOSUB 550; GOTO 790
230 RETURN
240 X = X - 1: GOSUB 110: IF A < 10 THEN A = A: IF A < 15 THEN GOTO
260
250 GOTO 310
260 X = X + 1; Y = Y + 1: GOSUB 110: IF A < 10 THEN A = A: IF A < 15
5 THEN GOTO 280
270 GOTO 310
280 X = X - 1; Y = Y - 1: GOSUB 110: IF A < 10 THEN A = A: IF A < 15
5 THEN GOTO 300
290 GOTO 310
300 X = X - 1; Y = Y - 1: GOSUB 110: IF A < 10 THEN A = A: IF A < 15

5 THEN Y = Y + 1: RETURN
310 HOME: VTab 22: PRINT "PLATED HOLE NOT POSSIBLE- TOUCHING PAT": CALL
340 POKE 49168, 0
350 P = PEEK (49152); IF F < 127 THEN GOTO 340
360 IF F = CHR$(217) THEN FS = "Y": POKE 49168, 0: PRINT: RETURN
370 IF F = CHR$(206) THEN FS = "N": PRINT "Y": POKE 35, PEEK (25) +
380 IF F = CHR$(141) THEN FS = CHR$(217): GOTO 340
390 IF FS = CHR$(136) THEN FS = CHR$(217): GOTO 340
400 IF FS = CHR$(149) THEN FS = CHR$(206): GOTO 370
410 FS = "2": POKE 49168, 0: PRINT: RETURN
430 IF OL = 1 THEN VTab 22: HTAB 1: PRINT SPC(40): VTab 22: HTAB 1: INVERSE
440 PRINT M$(F): NORMAL : POKE 49168, 0
450 IF OL = 0 THEN VTab 10: FOR I = 1 TO MAX: HTAB 1: PRINT M$(I): NEXT
460 IF FS = PEEK (49152); IF IC < 127 THEN GOTO 450
470 IF FS = CHR$(141) THEN RETURN
480 IF FS = CHR$(136) THEN F = F - 1: IF F < 0 THEN F = MAX
490 IF FS = CHR$(149) THEN F = F + 1: IF F > MAX THEN F = 1
500 GOTO 420
510 HOME: PRINT "THE CATALOG LISTING IS": PRINT D$: "CATALOG,D": D2: RETURN
520 IF YC = UR THEN KT = KT + 1: IF KT > 10 THEN POP : GOTO 1810
530 IF YC = UR THEN RETURN
540 IF UR = UR THEN KT = KT + 1: IF KT > 10 THEN POP : GOTO 1810
550 IF UU = UR THEN RETURN
560 IF KT = 0: UU = UR: RETURN
570 A = PEEK (797); IF A = 8 THEN A$ = "COMPONENT SIDE": RETURN
580 A$ = "SOLDER SIDE": RETURN
590 FOR DY = 1 TO 4000: NEXT: RETURN
600 POKE 36437, 22: POKE 36440, 8: POKE 36528, 58: POKE 36529, 145: POKE 78
610 7, 8: POKE 916, 8: POKE 923, 19: POKE 932, 19: RETURN : REM SWITCH TO COMPONENT SIDE
620 POKE 36437, 11: POKE 36440, 20: POKE 36528, 132: POKE 36529, 142: POKE 78
630 7, 19: POKE 916, 19: POKE 923, 30: POKE 932, 20: RETURN : REM SWITCH TO SOLDER SIDE
640 POKE 36437, 12: POKE 36440, 20: POKE 36528, 132: POKE 36529, 142: POKE 78
650 7, 19: POKE 916, 19: POKE 923, 30: POKE 932, 20: RETURN : REM TEMP1 TO TEMPI
660 HPLLOT 0, 271, 159 TO 0, 159 TO 0, 0: RETURN
670 POKE 35888, 4: POKE 35889, 7: POKE 35894, 234: POKE 3589
690 : REM TEMP2 TO TEMPI
700 POKE 35883, 2: POKE 35887, 5: POKE 35891, 8: POKE 35894, 201: POKE 3589
710 5, 15: POKE 35900, 135: POKE 35901, 261: POKE 35902, 230: CALL 35867: RETURN
720 : REM ERASE TEMPI'S
730 POKE 35883, 2: POKE 35887, 5: POKE 35891, 8: POKE 35894, 201: POKE 3589
740 5, 15: POKE 35900, 135: POKE 35901, 0: POKE 35902, 135: CALL 35867: RETURN
750 : REM ERASE TEMPI'S
760 IF TRY = 0 THEN TRY = 1: Y = YC + 1: X = XC: GOSUB 110: IF A = 0 THEN
770 Y = Y + 1: X = X: LOOP = 0: GOTO 1280
780 GOTO 1820
790 Y = YC - 1: X = XC: GOSUB 110: IF A = 0 THEN YC = Y: XC = X: LOOP = 0: GOTO
800 1280
810 GOTO 1840
104

690 IF SIDE < > 0 THEN GOTO 720
695 HOME: VTAB 10: PRINT "IS THIS A DOUBLE SIDED PC BOARD?": GOSUB 3
700 IF F$ = "Y" THEN SIDE = 2: GOTO 720
705 IF F$ < > "N" THEN PRINT "INVALID ENTRY": CALL 36815: GOSUB 370: GOTO 690
710 SIDE = 1
720 X = FRE (0): ER = 2: NODE = 1: FLAG = 0: RAN = 0: CHNERR GOTO 50
725 PRINT TRI": PRINT DS: 1: "OPEN":I,L4" :D$: 02
740 PRINT DS: 1: "READ":I,L4",R": RN: INPUT DS:I = RN / 3 + 1: RAN = RN + 1: XH(I) = VAL (DS)
750 PRINT DS: 1: "READ":I,L4",R": RN: INPUT DS:I = RN + 1: XH(I) = VAL (DS)
770 POKE 230,21: NV = 0: OV = 0: CZ = 0: ER = 0: ONERR GOTO 50: PRINT TRI": PRINT DS: 1: "OPEN":I,L4",D$: 02
790 LOC = 0: HCOLOR = 0: CALL 62330: HBR = 303: GOSUB 300: CALL 36825: J = 1: K = 0: IF OV = 1 THEN OV = 0: IF SIDE = 2 THEN A = PEEK (787): PEEK 787, A
800 L$ = "':LS = "'
810 KI = KI + 1: IF KI > LEN (LS) THEN GOSUB 370: GOTO 730
820 LS = MID (LS,KI-1): IF LS = CHR (46) THEN GOTO 360
830 IF LS = CHR (47) THEN J = J - 1: GOTO 590
840 IF LS = CHR (45) THEN J = J + 1: KI = KI + 1: GOTO 300
850 BLS = BL$ = LS: GOTO 310
860 GOSUB 370: GOTO 220
870 JJ = J: LS = CHR (47) THEN JJ = J - 1
875 IF VAL (LS) = 0 THEN RETURN
880 IF CH(JJ) = 0 THEN RETURN
890 IF CH(JJ) > 0 THEN GOTO 910
900 X = X + XH(JJ) + (P - 1) * 6 - 1: Y = YH(JJ) - J - 1: IF Y > 255 THEN Y = Y - 256: GOTO 120
910 YT = 0: YL = Y
920 POKE 36351,X: POKE 36357,YL: POKE 36389,YT: POKE 36382, J: CALL 3638
930 IF DZ = 0 THEN GOTO 900
940 HOME: VTAB 22: PRINT "BEGIN NODE ":NODE = J: GOSUB 330
950 IF F$ = "N" THEN HOME: NODE = NODE + 1: GOTO 790
960 IF F$ = "Y" THEN PRINT "INVALID ENTRY": CALL 36815: GOSUB 370: GOTO 730
970 LT$ = "TRY = 0: TST = 0: CH = 0: HOME: VTAB 22: IF DZ = 0 THEN PRINT "POSITION JOYSTICK AT ": INVERSE: PRINT "START": PRINT "OF TRACE": PRINT "PRESS BUTTON 0"
975 IF DZ = 1 THEN XI = X: Y = Y: IF Y = 255 THEN YT = 1: YL = INT (Y - 256): GOTO 970
980 IF DZ = 1 THEN YT = 0: YL = Y
985 IF DZ = 1 THEN DZ = DZ: POKE 36391,X: POKE 36387,YL: POKE 36389,YT: CALL 36381: GOTO 1070
990 IF XI = 0 THEN XI = 1: XI = 3
1000 IF XI = 1 THEN XI = 2: XI = 5
1010 IF XI = 2 THEN XI = 1: XI = 3
1020 YT = 0: YL = Y
1040 IF XI = 0 THEN XI = 1: XI = 3
1050 IF XI = 1 THEN XI = 2: XI = 5
1060 IF XI = 2 THEN XI = 1: XI = 3
1070 YT = 0: YL = Y
1040 X = XI; Y = YI; XF = XI; YF = Y; GOSUB 110; BY = A; HOME : VTAB 22: PRINT 
"WAIT = CONTINUE / BUTTON 1 = RE-ENTER"; DY = 100
1050 SW = PEEK (49250): IF SW > 127 THEN CALL JAS81: GOTO 970
1060 DY = DY - 1; IF DY > 0 THEN GOTO 1050
1070 X = PRE (0); HOME : VTAB 22: PRINT "POSITION JOYSTICK AT ": INVERSE 
PRINT "END"; NORMAL : PRINT "OF TRACE"; PRINT "PRESS BUTTON 0"
1080 Y = SY / 255 X POL (0): FOR F = 1 TO 10: NEXT I = SY / 255 X POL
1100 IF XF = 0 THEN XF = 1: Y = 3
1101 IF YF = 1 THEN YF = 1: Y = 6
1110 IF Y > 255 THEN YL = INT (Y - 255): GOTO 1130
1120 YT = SY YF = Y
1130 POKE JAS79.1: POKE JAS87. YL: POKE JAS89. YT: CALL JAS81: SW = PEEK 
(49249): IF SW < 128 THEN FOR F = 1 TO 10: NEXT I = SY X POL
1140 HOME : VTAB 22: PRINT "WAIT = CONTINUE / BUTTON 1 = RE-ENTER"; DY = 100
1150 SW = PEEK (49250): IF SW > 127 THEN CALL JAS81: GOTO 1070
1160 DY = DY - 1; IF DY > 0 THEN GOTO 1150
SPOC (15) "ANALYZING"
1180 IF X1 < XF THEN X = XI X1: XF = XF YF = Y1 YF = Y: IF YF = Y
1190 VF = 0: IF X1 = XF THEN X1 = XI: IF Y1 = YF THEN XI = YF YF = Y: IF YF = Y
GOTO 1930
1220 IF A = 2 THEN GOTO 1260
1230 IF A = 6 THEN GOTO 1260
1240 IF A > 8.9 THEN A = A: IF A < 15.1 THEN GOTO 1260
1250 A = 1260
1260 IF XI = XF THEN FLAG = 1; GOTO 1330
1270 IF YF = YF THEN GOTO 1360
1280 IF CHK = 2 THEN TST = 0:FLAG = 0; GOTO 1330
1290 IF CHK = 3 THEN TST = 0:FLAG = 0; GOTO 1330
1300 IF CHK = 4 THEN TST = 0:FLAG = 0; GOTO 1330
1310 IF CHK = 6 THEN TST = 0:FLAG = 0; GOTO 1330
1311 IF CHK = 7 THEN TST = 0:FLAG = 0; GOTO 1330
1320 TST = 0:FLAG = 0; SLEP = (YF - YF) / 2: SLEP = INT (SLEP): SLEP = ABS 
(SLEP): SV = SLEP: IF SLEP < 1 THEN SLEP = 1: SV = SLEP
1330 IF YF < YF THEN GOTO 1440
1340 IF YF > YF THEN GOTO 1580
1350 GOTO 1520
1360 Y = YC X1 = XC; GOSUB 110: IF A > 10 THEN A = A: IF A > 15 THEN GOTO 1540
1370 IF XI > XF THEN GOTO 1390
1380 IF XI < XF THEN GOTO 1420
1390 IF X > XF THEN X = X - 1; GOSUB 110: IF A > 10 THEN A = A: IF A > 15 THEN GOTO 1390
1400 IF X = XF THEN GOTO 1540
1410 GOTO 1810
1420 IF X > XF THEN X = X + 1; GOSUB 110: IF A > 10 THEN A = A: IF A > 15 THEN GOTO 1420
1430 GOTO 1400
1440 IF TST < 1 THEN Y = Y: YF = YF: IF YC = YC.YC = YC IF XC < XF THEN 
TST = TST + 1: GOTO 1530
1450 X = XE YC = YC + 1; GOSUB 110: IF A > 0 THEN Y = YF: IF A = 10 THEN 
GOTO 2220
1460 IF A < 0 THEN Y = Y: IF A = 15 THEN URC = YC: GOTO 2220
1470 IF A < 0 THEN Y = Y: IF Y = YF THEN GOTO 1520
106

1480 IF A > 0 THEN GOTO 1530
1490 YC = YC + 1; A = 0; GOSUB 120: SLEP = SLEP - 1: LOOP = 0; IF FLAG < > 0 THEN GOTO 1510
1500 IF SLEP = 0 THEN GOTO 1530
1510 IF YC > YF THEN GOTO 1440
1520 IF XC = XF THEN GOTO 1500
1520 IF A < > 0 THEN A = A: IF A < > 9 THEN A = 1: GOSUB 120
1540 IF XC < XF THEN GOTO 1670
1550 IF XC > XF THEN GOTO 1740
1560 GOTO 1810
1580 IF TST < 7 THEN Y = YF + 3: IF YC = Y THEN YC = YC: IF XC > XF THEN TST = TST - 1: GOTO 1530
1590 X = XG: YC = YC - 1: GOSUB 110: IF A < > 0 THEN Y = Y: IF A = 10 THEN UR = YC: GOTO 2220
1600 IF A < > 0 THEN Y = Y: IF A = 15 THEN UR = YC: GOTO 2220
1610 IF A < > 0 THEN Y = Y: IF Y = YF THEN GOTO 1520
1620 IF A < > 0 THEN GOTO 1530
1630 IF XC = YC - 1: A = 07: GOSUB 120: SLEP = SLEP - 1: LOOP = 0; IF FLAG < > 0 THEN GOTO 1530
1640 IF SLEP = 0 THEN GOTO 1530
1650 IF YC < YF THEN GOTO 1580
1660 GOTO 1520
1670 X = XC - 1: Y = YC: GOSUB 110: IF A < > 0 THEN X = X: IF X = XF THEN X = X: IF YC = Y THEN XC = XF
1680 IF A < > 0 THEN LOOP = LOOP + 1: IF LOOP = 10 THEN SLEP = 90: IF XC = IF THEN X = X: IF YC = Y THEN GOTO 2000
1700 X = XC - 1: Y = YC: GOSUB 110: IF A < > 0 THEN X = X: IF LOOP < 10 THEN GOTO 1530
1710 IF A < > 0 THEN GOTO 1810
1720 XC = XC - 1: A = 04: GOSUB 120: LOOP = 0; IF XC < > XF THEN GOTO 1670
1730 FLAG = 1: X = XC: Y = YC: A = 1: GOSUB 120: GOTO 1750
1740 X = XC - 1: Y = YC: GOSUB 110: IF A < > 0 THEN X = X: IF X = XF THEN X = X: IF YC = Y THEN XC = XF
1750 IF A < > 0 THEN LOOP = LOOP + 1: IF LOOP < 10 THEN SLEP = 90: IF XC = IF THEN X = X: IF YC = Y THEN GOTO 2000
1770 X = XC - 1: Y = YC: GOSUB 110: IF A < > 0 THEN X = X: IF LOOP < 10 THEN GOTO 1530
1780 IF A < > 0 THEN GOTO 1810
1790 XC = XC - 1: A = 04: GOSUB 120: LOOP = 0; IF XC < > XF THEN GOTO 1640
1810 IF TRY = 0 THEN TRY = 0; IF YC = YF THEN GOTO 630
1820 IF TRY = 1 THEN TRY = 2: GOSUB 640: TST = 0: LOOP = 0: FLG = 0: SLEP = SV: GOTO 1190
1830 IF TRY = 2 THEN TRY = 3: IF YC = YF THEN GOTO 670
1840 IF CHK = 0 THEN FLAG = 0: LOOP = 0: TRY = 0: X = XI: Y = YI: CHK = 1: XI
1850 IF X = YF THEN Y = YF: X = XI: THEN X = X: IF Y = YF THEN GOSUB 640: GOTO 1190
1850 IF CHK = 0 THEN CHK = 1: LOOP = 0: TRY = 0: FLG = 0: SLEP = 90: SV = 5
1860 IF CHK = 2 THEN CHK = 3: LOOP = 0: TRY = 0: FLG = 0: SLEP = 90: SV = 5
1860 IF X = X: THEN X = X: IF Y = YF THEN GOSUB 640: GOTO 1190
1870 IF CHK = 3 THEN CHK = 4: LOOP = 0: TRY = 0: FLG = 0: SLEP = 3: SV = 5
1880 IF CHK = 4 THEN CHK = 5: LOOP = 0: TRY = 0: FLG = 0: SLEP = 3: SV = 5
1890 IF X = X: THEN X = X: IF Y = YF THEN GOSUB 640: GOTO 1190
MATRIX AND CLEAR ROUTINES

0200: A9 FF      ORG $0300
0202: BS 19      LDA #$FF
0204: AC 08 03   JMP H030B  ! START
0207: A9 00      LDA #00
0209: BS 19      STA #19
020B: 08 H0308   PHP
020D: 08 CLD
020F: 20 87 03   JSR H0387  ! SET UP (A)=1A, (X)=1B, (Y)=1C
0310: 48         PHA
0312: CA         DEX
0314: A9 13      LDA #$13
0316: 00 STA     $09
0318: BS 07 STA   #00
031A: BS 06 STA   #07
031C: 22 A0      LDX #$22
031E: 18 H031E   CLC
0320: EA 06 ADC   $06
0322: 90 03       BCD H0326  IF CARRY, INCREMENT NEXT BYTE
0324: 18       CLC
0326: EA 07 INC   $07
0328: CA H032A   DEX
032A: DO FS      BNE H031E  ! CONTINUE FOR ALL COLUMNS
032C: 69 10      ADC #$10
032E: 90 02      BCD H032F  ! IF CARRY SET, INCREMENT HI BYTE
0330: EA 09 INC   $09
0332: BS 08 H032F STA $08  ! SAVE LOW BYTE TO $0008
0334: A5 07      LDA $07  ! ADD ($0007) TO ($0009)
0336: 18       CLC
0338: EA 09 ADC   $09
033A: 83 09 STA   $09  ! SAVE BACK TO $0009
033C: 9B       TYA
033E: 4A LSR
0340: 90 05      BCD H0341  ! IF NO CARRY, COLUMN IS EVEN
0342: 69 00      ADC #$00  ! IF ODD COLUMN, INCREMENT A
0344: 4C 81 03   JMP H0361  ! JUMP TO ODD
0346: A8 H0341   TAY
0348: A5 19      LDA $19  ! CHECK FLAG, SEE IF READ OR WRITE
034A: C9 00      CMP #$00
034C: F0 0A      BEQ H0352  ! WRITE TO EVEN MATRIX LOCATION
034E: 48 PLA
0350: B1 08      LDA ($08),Y  ! READ ARRAY LOCATION
0352: 29 0F AND   #$0F
0354: 20 8E 03   JSR H038E  ! KEEP LOW NIBBLE ONLY, SINCE EVEN
0356: 28 PLP
0358: 60 RTS
035A: A9 F0 H0332 LDA #$F0  ! EVEN WRITE; CLEAR LOW NIBBLE
035C: 31 08 AND   ($08),Y
035E: 91 08 STA   ($08),Y
0360: 68 PLA
0362: 27 0F AND   #$0F
0364: 11 08 ORA   ($08),Y
0366: 91 08 STA   ($08),Y
0368: 28 PLP
036A: 60 RTS
036C: A8 H0361   TAY
036E: A5 19      LDA $19  ! CHECK FLAG; READ OR WRITE
0370: C9 00      CMP #$00  ! IF 00 THEN WRITE TO ARRAY
0372: F0 0E BEQ H0376  ! WRITE TO ODD ARRAY LOCATION
0374: 48 PLA
LDA ($0b),Y
AND #$F0
KEEP HI NIBBLE SINCE ODD

JSR HOJBE
SAVE TO RAM LOCATIONS

PLP

RTS

LDA #$0F
ODD WRITE: CLEAR HI NIBBLE

STA ($0b),Y
SAVE BACK TO ARRAY

ASL

LDA ($08),Y
ADJUST ARRAY LOCATION

LDA ($08),Y
STORE IN ARRAY LOCATION

PLP

RTS

LDA $1A
LOAD REGISTERS FROM RAM

LDD $1A
LDA $1B

LDY $1C

RTS

LDA $1A
ARRAY CLEAR

LDA $15
START AT 1510

STA $09
SAVE TO ZERO PAGE LOCATION

LDA $10

STA $1E00
SET ENDPOINT TO NONZERO

STA $08
SAVE TO ZERO PAGE LOCATION

LDA $100
CLEAR BYTES

STA ($0b),Y

LDD $1E00
CHECK IF ENDPOINT IS ZERO

LDX $E00

BNE HOJAE

INY

SNE $09

SNE $09E

RTS

PLA

TAY

PLA

LDX #$DF

TIX

PHA

TYA

PHA

RTS
III
ORS SSC.1B
·C1S:
06 ?HF
S010: OS CL.D
8010: ACf
01-
LOA
4f.=01
S7;'RT Ai ROW 1
CCLW~N
1
6C1F:
as
0:;: 01 H1CZ1 LDA *$01
3C23:
8~
.,... 5TA SlC
ASS
VARIABLE
TO MATRIX ROUTINE
8C::S:
=0
'JO
1,)3
H1C2~
JSR H0300 READ USING MATRIX ROUTE!
127x553
ac2: EA
SC2: EA
Nop
SC3: DO 09
BNE H1C43
SC4: A9 09
LDA #409
SC5: SS 1A
H1C3C
STA $1A
INC $1A
CHANGE TO LESS
RECENT
SC6: ES 1A
LDA SlA
SC7: JSR H0307 WRITE TO ARRAY WITH MATRIX ROUTINE
SC8: ES 1C
INC $1C
NEXT ARRAY LOCATION
SC9: A9 SB
H1C43
LDA $6B
SAME ROW?
SCA: CS 1C
CMP $1C
SCB: DO 5A
BNE H1C25
SCE: ES 13
INC $1B
NEW ROW
SCD: A9 75
LDA $45
SAFE RCW
SCF: CS 1B
CMP $1B
LAST ROW, LAST COLUMN
SCG: DO CE
BNE H1C21
PLP
RTS
SC1: 28
PLP
RTS
SC2: 08
PHP
SC3: 0B
CLD
SC4: A9 02
LDA $62
SC5: 35 1B
H1C3B
LDA $402
SC6: 35 1C
LDA $402
SC7: A9 02
H1C3F
STA $1B
READ MATRIX LOCATION
SC8: 35 1A
LDA $1A
SC9: C9 02
CMP $02
SEE IF LESS RECENT "CORNER" HERE
SCA: DO 03
BNE H1C68
IF YES, TRY TO CONVERT
SCB: 20 7D 8C
H1C68
INC $1C
NEXT ARRAY LOCATION
SCE: A9 5A
LDA $5A
SAME ROW?
SCF: CS 1C
CMP $1C
SAME ROW
SCG: DO EC
BNE H1C6F
SCH: ES 1B
INC $1B
NEW ROW
SCI: A9 54
LDA $54
CHECK IF AT END
SCJ: CS 1B
CMP $1B
IF NO, KEEP GOING
SCK: DO E0
BNE H1C58
PLP SAVE
REGISTER
SCL: 60
RTS
### Corners and Screen Show Routines

<table>
<thead>
<tr>
<th>SC7D:</th>
<th>C6 1B</th>
<th>ORG $8C7D</th>
<th>Above the dot</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC7F:</td>
<td>C6 1C</td>
<td>DEC $1B</td>
<td></td>
</tr>
<tr>
<td>SC82:</td>
<td>20 2D 8D</td>
<td>JSR HBD2D</td>
<td>Read matrix</td>
</tr>
<tr>
<td>SC85:</td>
<td>A5 1A</td>
<td>JSR HBD1A</td>
<td>Change permanent to blank</td>
</tr>
<tr>
<td>SC87:</td>
<td>04</td>
<td>LDA #1A</td>
<td>Effective array contents</td>
</tr>
<tr>
<td>SC89:</td>
<td>0A</td>
<td>ASL</td>
<td>TO THE RIGHT OF DOT</td>
</tr>
<tr>
<td>SC9A:</td>
<td>0A</td>
<td>ASL</td>
<td></td>
</tr>
<tr>
<td>SC9B:</td>
<td>E6 1B</td>
<td>INC $1B</td>
<td></td>
</tr>
<tr>
<td>SC9D:</td>
<td>E6 1C</td>
<td>INC $1C</td>
<td></td>
</tr>
<tr>
<td>SC9F:</td>
<td>20 2D 8D</td>
<td>JSR HBD2D</td>
<td>Read matrix</td>
</tr>
<tr>
<td>SCA7:</td>
<td>05 1A</td>
<td>ORA $1A</td>
<td>ADD hi nibble to form 1 byte</td>
</tr>
<tr>
<td>SCA9:</td>
<td>35 F9</td>
<td>STA $FB</td>
<td>Temporary storage</td>
</tr>
<tr>
<td>SCAF:</td>
<td>E6 1B</td>
<td>INC $1B</td>
<td>Below the dot</td>
</tr>
<tr>
<td>SCB1:</td>
<td>C6 1C</td>
<td>DEC $1C</td>
<td></td>
</tr>
<tr>
<td>SCB3:</td>
<td>20 2D 8D</td>
<td>JSR HBD2D</td>
<td>Change permanent to blank</td>
</tr>
<tr>
<td>SCB6:</td>
<td>02 1A</td>
<td>ORA $1A</td>
<td></td>
</tr>
<tr>
<td>SCB8:</td>
<td>35 FC</td>
<td>STA $FC</td>
<td>Temporary storage of lo nibble</td>
</tr>
<tr>
<td>SCB9:</td>
<td>E6 1C</td>
<td>INC $1C</td>
<td>Back to original location</td>
</tr>
<tr>
<td>SCB9:</td>
<td>A9 3C</td>
<td>LDA #$C3</td>
<td>Table address</td>
</tr>
<tr>
<td>SCC0:</td>
<td>02 1D</td>
<td>STA $F9</td>
<td>LO Byte</td>
</tr>
<tr>
<td>SCC0:</td>
<td>A9 8D</td>
<td>LDA #$8D</td>
<td>TABLE ADDRESS</td>
</tr>
<tr>
<td>SCC3:</td>
<td>85 FA</td>
<td>STA $FA</td>
<td>HI byte</td>
</tr>
<tr>
<td>SCC4:</td>
<td>A5 00</td>
<td>LDX #$00</td>
<td></td>
</tr>
<tr>
<td>SCC5:</td>
<td>A5 F9</td>
<td>LDY #$03</td>
<td></td>
</tr>
<tr>
<td>SCC7:</td>
<td>D1 F9</td>
<td>CMP ($F9),Y</td>
<td>Recover top and right</td>
</tr>
<tr>
<td>SCC9:</td>
<td>F0 16</td>
<td>SEC H1CE1</td>
<td>Check table</td>
</tr>
<tr>
<td>SCC9:</td>
<td>CB</td>
<td>INY</td>
<td>IF MATCH, CHECK OTHER TABLE</td>
</tr>
<tr>
<td>SCCC:</td>
<td>98</td>
<td>TYA</td>
<td></td>
</tr>
<tr>
<td>SCCD:</td>
<td>C1 F9</td>
<td>CMP ($F9,X)</td>
<td>Make sure not equal entries</td>
</tr>
<tr>
<td>SCCF:</td>
<td>D0 F4</td>
<td>BNE H1CC5</td>
<td>Continue with this table</td>
</tr>
<tr>
<td>SCD1:</td>
<td>18</td>
<td>CLC</td>
<td></td>
</tr>
<tr>
<td>SCD2:</td>
<td>65 F9</td>
<td>ADD $F9</td>
<td>Go to next list within table</td>
</tr>
<tr>
<td>SCD4:</td>
<td>85 F9</td>
<td>STA $F9</td>
<td></td>
</tr>
<tr>
<td>SCD5:</td>
<td>90 02</td>
<td>BCC H1CDA</td>
<td>Base address adjust</td>
</tr>
<tr>
<td>SCD8:</td>
<td>E6 FA</td>
<td>INC $FA</td>
<td></td>
</tr>
<tr>
<td>SCD9:</td>
<td>A9 FF</td>
<td>HICDA LDA $FF</td>
<td>See if at end of table</td>
</tr>
<tr>
<td>SCCD:</td>
<td>C1 F9</td>
<td>CMP ($F9,X)</td>
<td>Continue</td>
</tr>
<tr>
<td>SCDE:</td>
<td>D0 E9</td>
<td>BNE H1CC5</td>
<td></td>
</tr>
<tr>
<td>SCFO:</td>
<td>60</td>
<td>RTS</td>
<td>No match found</td>
</tr>
<tr>
<td>SCE1:</td>
<td>9B</td>
<td>H1CE1 TYA</td>
<td>Find address of next table</td>
</tr>
<tr>
<td>SCE2:</td>
<td>4B</td>
<td>PHA</td>
<td></td>
</tr>
<tr>
<td>SCE3:</td>
<td>8A</td>
<td>TAX</td>
<td>Store to $00FD</td>
</tr>
<tr>
<td>SCE4:</td>
<td>4B</td>
<td>PHA</td>
<td></td>
</tr>
<tr>
<td>SCE5:</td>
<td>A0 01</td>
<td>LDY #$01</td>
<td></td>
</tr>
<tr>
<td>SCE7:</td>
<td>81 F9</td>
<td>LDA ($F9),Y</td>
<td></td>
</tr>
<tr>
<td>SCE9:</td>
<td>85 FD</td>
<td>STA $FD</td>
<td></td>
</tr>
<tr>
<td>SCEB:</td>
<td>C9</td>
<td>INY</td>
<td></td>
</tr>
<tr>
<td>SCEC:</td>
<td>81 F9</td>
<td>LDA ($F9),Y</td>
<td>Store to $00FE</td>
</tr>
<tr>
<td>SCEE:</td>
<td>85 FE</td>
<td>STA $FE</td>
<td></td>
</tr>
</tbody>
</table>
113

A2 00
A0 02
A2 02
D1 FD
F0 14
C3
P8
C1 FD
D0 F4
A9 03
25 1A
20 07 03
68
AA
68
A8
4C 03 0C
A0 01
B1 FD
85 1A
20 07 03
68
68
60
A6 1A
E0 03
F0 08
E0 06
F0 04
E0 09
D0 04
A2 00
E0 00
6B 1A
60
B6 1A
60
4B
20 00 03
68
60
9B 0C 00
02 08 02
88 82 88
05 99 50
E3 4B 08
A2
B0 02 08
B2 3B 82
88 82 EB
0A AC 9D
50 55 5B
5C
C0 C5 CC
07 B6 6D
D0 05 0B
DC 0B 8E
BD 50 55
5B
5C CO CC
C0 C5 CC
07 B6 6D
D0 05 0B
DC 0B 8E
D9 06 06
52

LDX #00
LDY #02
H1CF4
LDA #0C
CMP ($FD),Y
BEQ H1DOE
IF YES, THEN COMPLETE MATCH
INY
TYA
CMP ($FD,Y)
MAKE SURE LIST NOT EXCEEDED
BNE H1CF4
NOT END OF LIST, CONTINUE
STA $1A
JSR H0307
IF NO MATCH, DOT BECOMES PERMANENT
PLA
TAX
PLA
RTS
HIDOE
LDY #$01
MATCH TO TABLE
LDA ($FD),Y
STA $1A
JSR H0307
WRITE PROPER CORNER TO ARRAY
PLA
PLA
RTS
LDX $1A
SEE IF CONTENT OF MATRIX PERMANENT
CPX #$03
BEQ H12B
CPX #$06
BEQ H12B
CPX #$09
BNE H12B
PERMANENT BECOMES BLANK
LDA #$00
STX $1A
EFFECTIVE ARRAY CONTENTS
RTS
PHA
JSR H0300
READ ARRAY AFTER SAVING ACCUM
PLA
RTS
HEX 098C5D0209B828B2B0598BE2B082A
HEX 8D0206B2B8B2B0E20AC3D50555B5C
HEX C0C5CC07B66D0D6D6DC0B8E6D50555B5
HEX 5C0C2C8C07C85D0D6D6DC096D6D52
SESB: A5 1C  ADC $1C  CHANGE ARRAY TO PIXEL COORDINATES
SESD: $C HIE94
SESS: a:5
SE71: 90 01
SE72: CB
SE74: A4  HIE94
SE7S: C6 1B  DEC $13
SE77: A5 1B  LDA $1B
SE79: 1B  CLC
SE8A: 16  ADD $1B
SE8B: 3F  STA $F9
SE8C: 94 FC  STY $FC
SE84: 36 3F  STX $FD
SE86: 30  RTS
SE87: EA  NOP
SE88: EA  NOP
SE89: EA  NOP
SE8A: EA  NOP
SE8B: 35 3F  STA $F9  GET READY TO DRAW SHAPE
SE8D: 94 FA  STY $FA  STORE SHAPE ADDRESS
SE8E: 20 2A 3F  JSR H8F3A
SE90: 20 3E 3E  JSR H8E73
SE92: 4C 19 3F  JMP H8F19
SE93: 08  PHP
SE94: 30  CLD
SE95: A9 01  LDA #$01  START AT FIRST ARRAY LOCATION
SE96: 35 1B  STA $1B
SE97: A9 01  HIE8E  LDA #$01
SE98: 95 1B  STA $1C
SE99: A5 13  HIEC2  LDA $1B  WHICH LOCATION
SE9A: 48  PHA
SE9B: A5 1C  LDA $1C
SE9C: 48  PHA
SE9D: 20 00 03  JSR H0000  READ THAT ARRAY LOCATION
SE9E: A5 1A  LDA $1A  CONTENTS?
SE9F: C9 00  CMP #$00  SEE IF BLANK
SEAC: F0 4B  BEQ H1F19
SEAD: A2 00  LDX #$00
SEAE: DD EC 8E  HIE1  CMP HBECC,X IF NOT BLANK, CHECK IN TABLE
SEAF: F0 09  BEQ H1EE1  IF MATCH FOUND, GO
SEAG: EB  INX
SEAH: EB  INX
SEAI: EB  INX
SEAJ: EB  INX
SEAK: EB  INX
SEAL: EB  INX
SEAM: EB  H1EE1  INX
SEAN: 8D EC 8E  LDA HBECC,X
SEAO: EB  INX
SEAP: EC EC 8E  LDY HBECC,X
SEAQ: 4C AB 8E  JMP HBEAB  DRAW SHAPE
SEAR: 01 03 8E
SEAS: 02 03 8E
SEAT: 03 03 8E
SEAU: 04 FE 8D
SEAV: 05 FE 8D
SEAW: 06  HEX 0103BE0203BE0303BE04FE3D05FE5D06
SEAX: FE 8D 07
SEAY: F9 8D 08
SEAZ: F9 8D 09
SEBA: F9 8D 0A
SEBB: FO 8D 0B
SEBC: 0C  HEX FE3D07F8D08F9D09F8D0AF08D0BC
117

SF0C: BE 0C 11
SF0F: BE 0D 18
SF12: BE 0E 07
SF13: BE 0F 0D
SF18: 3D       HEX 3EC11BE0D18BE0E07BE0F0D
SF19: 5B       H1F19
SF1A: 5B 1C
SF1C: 5B
SF1D: 5B 19
SF1F: 61 1C
SF21: A9 3B
SF22: C0 1C
SF23: D0 9B
SF27: EA 1B
SF29: A9 33
SF2A: C2 19
SF2D: 30 3F
SF2F: 3B
SF30: 60
SF31: EA
SF32: EA
SF33: EA
SF34: EA
SF35: EA
SF36: EA
SF37: EA
SF38: EA
SF39: EA
SF3A: A0 00
SF3C: A0 1C
SF3E: A0 1D
SF40: 18
SF41: 85 1C
SF42: E6 1C
SF45: E5 1C
SF47: 90 01
SF49: CB
SF4A: 85 FD       H1F4A
SF4C: A9 0E
SF4E: 18
SF50: EA FD
SF51: AA
SF52: 30 04
SF54: A0 00
SF56: F0 06
SF58: CD 01
SF5A: A0 01
SF5E: 4C 95 BE

SF19: PLA
STA $1C SAVE COLUMN
SF1C: PLA
STA $1B SAVE ROW
SF21: INC $1C
SF22: LDA #$3B COLUMN COUNTER
SF23: CMP $1C CHECK IF Y EXCEEDS ROW
SF27: INC $1B NEXT ROW
SF29: LDA #$33 CHECK IF LAST ROW
SF2A: CMP $1B
SF2D: BNE H1E3E CONTINUE
SF2F: RT3
SF30: NOP
SF31: NOP
SF33: NOP
SF34: NOP
SF35: NOP
SF36: NOP
SF37: NOP
SF38: NOP
SF39: LDY #$00 HI BYTE INITIALIZED TO 0
SF3C: DEC $1C X-1
SF3E: LDA #$1C A=X-1
SF40: CLC
SF41: ADC $1C (A)=(X-1)+(X-1)
SF42: INC $1C X
SF45: ADC $1C (A)=(X-1)+(X-1)+X
SF47: BCC H1F4A IF CARRY SET, HI BYTE SET
SF49: INY
SF4A: STA $FD $FD=SUM
SF4C: LDA #$0E
SF4E: CLC
SF50: SBC $FD (A)=3X-16
SF51: TAX
SF52: BCS H1F5B
SF54: A0 00
SF56: LDY #$00
SF58: BEQ H1F5E
SF5A: CD 01
SF5C: F0 0B
SF5E: A0 01
SF6E: 4C 95 BE

SF19: H1F19
SF21: $1C
SF22: $1C
SF23: $1C
SF27: $1B
SF29: $33
SF2A: $1B
SF2D: H1E3E
SF30: RT3
SF31: NOP
SF33: NOP
SF34: NOP
SF35: NOP
SF36: NOP
SF37: NOP
SF38: NOP
SF39: LDY #$00
SF3C: DEC $1C
SF3E: LDA #$1C
SF40: CLC
SF41: ADC $1C
SF42: INC $1C
SF45: ADC $1C
SF47: BCC H1F4A
SF49: INY
SF4A: STA $FD
SF4C: LDA #$0E
SF4E: CLC
SF50: SBC $FD
SF51: TAX
SF52: BCS H1F5B
SF54: A0 00
SF56: LDY #$00
SF58: BEQ H1F5E
SF5A: CD 01
SF5C: F0 0B
SF5E: A0 01
SF6E: 4C 95 BE

SF0C: BE 0C 11
SF0F: BE 0D 18
SF12: BE 0E 07
SF13: BE 0F 0D
SF18: 3D
SF19: 5B
SF1A: 5B 1C
SF1C: 5B
SF1D: 5B 19
SF1F: 61 1C
SF21: A9 3B
SF22: C0 1C
SF23: D0 9B
SF27: EA 1B
SF29: A9 33
SF2A: C2 19
SF2D: 30 3F
SF2F: 3B
SF30: 60
SF31: EA
SF32: EA
SF33: EA
SF34: EA
SF35: EA
SF36: EA
SF37: EA
SF38: EA
SF39: EA
SF3A: A0 00
SF3C: A0 1C
SF3E: A0 1D
SF40: 18
SF41: 85 1C
SF42: E6 1C
SF45: E5 1C
SF47: 90 01
SF49: CB
SF4A: 85 FD
SF4C: A9 0E
SF4E: 18
SF50: EA FD
SF51: AA
SF52: 30 04
SF54: A0 00
SF56: F0 06
SF58: CD 01
SF5A: A0 01
SF5E: 4C 95 BE

SF0C: BE 0C 11
SF0F: BE 0D 18
SF12: BE 0E 07
SF13: BE 0F 0D
SF18: 3D
SF19: 5B
SF1A: 5B 1C
SF1C: 5B
SF1D: 5B 19
SF1F: 61 1C
SF21: A9 3B
SF22: C0 1C
SF23: D0 9B
SF27: EA 1B
SF29: A9 33
SF2A: C2 19
SF2D: 30 3F
SF2F: 3B
SF30: 60
SF31: EA
SF32: EA
SF33: EA
SF34: EA
SF35: EA
SF36: EA
SF37: EA
SF38: EA
SF39: EA
SF3A: A0 00
SF3C: A0 1C
SF3E: A0 1D
SF40: 18
SF41: 85 1C
SF42: E6 1C
SF45: E5 1C
SF47: 90 01
SF49: CB
SF4A: 85 FD
SF4C: A9 0E
SF4E: 18
SF50: EA FD
SF51: AA
SF52: 30 04
SF54: A0 00
SF56: F0 06
SF58: CD 01
SF5A: A0 01
SF5E: 4C 95 BE
Bibliography


