I. INTRODUCTION

Any interactive graphics program is more complex than the usual "sample program" unless its only function is to plot a simple picture. If the program also defines and manipulates a data structure containing relational information about the problem, the complexity of the program rapidly increases. This example attempts to provide the "flavor" of a program which performs a useful function, but with enough simplifying assumptions to make it tractable as a demonstration program.

A variety of applications suitable for treatment using interactive graphics can be considered as a variation upon the basic problem of building, editing and analyzing networks. Examples of this type of problem are computer program flowcharting, design of piping networks, construction of PERT diagrams and the design of electrical circuits. A typical program of this type having sufficient facilities to make it useful to a professional in the applications field would run well over 100 pages of listings in a high-level programming language. In order to provide a vehicle for treating many of the issues important in the design and implementation of interactive graphics programs, a simple network drawing and editing program was implemented. The program could have been designed to deal with 'black boxes' and their interconnections. However, in order to give the program some degree of reality, the network components are represented by components from electrical engineering, but the program was designed as a

teaching vehicle rather than as a useful program for circuit design. It is therefore completely unnecessary to know anything about electrical engineering; components may be treated as different flavors of black boxes.

The program would have to be expanded considerably to be useful for circuit design and analysis, both in the types of elements it can handle and in the analysis routines provided, which are completely missing in this case. Therefore, the sample program should be regarded as being only a fraction of the size and complexity of a useful on-line circuit design system; the principal limitations are summarized in Section IX. In studying the program, one should regard the example as a vehicle to illustrate:

1. typical manipulations upon complex data structures,
2. programming techniques used to allow a user to interact with a program through the use of "light buttons",
3. the correlation of graphic input with the program data structure,
4. high-level language facilities which facilitate graphics programming.

The program allows the user to construct an electrical circuit containing nodes, capacitors, resistors, and short circuits, to modify the circuit, and to assign names and values to resistors and capacitors.

In order to specify large circuits, it is desirable to be able to define devices. A device is a circuit which will be used as a unit in building a larger circuit; it will be represented whenever it is used by a unique symbol. For example, it is simpler to use a symbol to represent a transistor than to draw the circuit defining the transistor every time a transistor is needed. It is frequently necessary to be able to substitute the definition of a device (i.e., the circuit specifying it) for its symbol when the properties of the circuit are being investigated. A device in the sample program serves much the same purpose as does a subroutine in a programming language; if the network problem considered had been computer program flowcharting, each subroutine would have a unique symbol to represent a call to it. As would be expected, a device can always be defined using previously defined devices and the basic circuit components, nodes, capacitors, resistors and shorts. The device definition must specify how it can be connected when used, just as the definition of a subroutine must include the number of arguments. The definition, usage and editing of devices is one of the more sophisticated portions of the sample program.

II. DISPLAY EQUIPMENT USED

The sample program has been designed to operate upon a display terminal using a storage CRT rather than the more usual type of terminal which continuously redisplays the picture. When a storage CRT is used, the image is transmitted to the terminal only once and is stored locally within the display tube. The display can be partially altered only for additions; to delete any portion of the picture, all the information on the screen must be erased and the display regenerated with the alterations. As the picture is drawn only once, light pens cannot be used with storage CRT displays, but a device such as a joystick driving a cursor can be used to point to objects; this mechanism is handled locally at the console. Applications that involve frequent picture changes, such as animated movies or the rotation or translation of objects, cannot be performed using a storage CRT terminal. However, for those applications for which it is sufficient, the cost of a commercially available terminal including a keyboard and graphic input facilities is about $10,000.

The terminal used for operating the sample program is the Advanced Remote Display Station (ARDS) which was developed at the M.I.T. Electronics System Laboratory as a remote display for a time-sharing system (2,3). The main design requirement was a graphic terminal that could operate over switched voice-grade telephone lines using the full ASCII character set with

minimum demands on the time-sharing system. The terminal includes a character generator and line generator. For graphics the display screen is defined to contain 1081 X 1415 addressable, but not necessarily resolvable, points; the point with x and y addresses 0,0 is in the center of the screen.

Two types of graphic input devices are available, a joystick and a "Mouse," both of which can be used to position a cursor on the screen. A feature of the terminal design is that the cursor can be displayed at an intensity sufficient to make it visible, but insufficient to cause it to store on the screen. Buttons are provided on the graphic input devices which allow either the cursor position, or the relative vector from the last beam position, to be transmitted.

The sample program was written using this terminal in order to illustrate that useful interactive programs can be created using reasonably-priced hardware which can communicate with computers over voice-grade telephone lines (preferably at speeds of at least 1200 baud). In addition, the program illustrates how the pointing function can be implemented using devices other than light pens.

III. PROGRAM FUNCTIONS

The program allows the user to perform the following seven functions:

1. To begin drawing the circuit by specifying a point on the screen where the first node is to be defined. A node is a point of intersection of circuit elements and can only occur at specified grid points on the screen.

2. To draw, from the node last indicated with the graphic input device, an element which may be:

   a. a horizontal capacitor, resistor or short circuit

   b. a vertical capacitor, resistor or short circuit.

3. To define as a device a circuit to be drawn, and to assign a symbol and a name to a device.

4. To use the device in constructing a circuit.

5. To delete a device, a node or an element pointed to by the graphic input device.

6. To assign a value to a resistor or capacitor.

7. To edit a device definition in order to alter the elements comprising the equivalent circuit for the device.

The user operates the program by using the graphic input device to point to a "light button" in order to indicate the program option desired. Nodes, devices and elements of interest are also denoted using the graphic input device. Values and names for devices, capacitors and resistors are input via a keyboard.

A light button is a descriptive phrase displayed on the screen to indicate a program option. When the user points

to a phrase with the graphic input device, the program identifies the light button and associates it with information in the program's data structure which indicates the proper action to be taken. In this program two sets of light buttons, or "menus," are used, one for operations upon the main circuit and one for operations involving devices. If changing the menu did not involve erasing the screen and rewriting all the information on it, which may take several seconds when using the ARDS terminal on a telephone line, more selective menus could have been used to further reduce the user options to include only those which the user could employ within the present context of his work. The two menus are shown on the next page.

This empty page was substituted for a blank page in the original document.

LIGHT BUTTONS

Light Button Text	Functions
DELETE	Delete a node, element or usage of a device
VALUE	Assign a value to a resistor or capacitor
HORIZ RESISTOR	Draw a horizontal resistor to the right from the node previously specified
VERT RESISTOR	Draw a vertical resistor down from the node last specified
HORIZ CAPACITOR	Draw a horizontal capacitor
VERT CAPACITOR	Draw a vertical capacitor
HORIZ SHORT	Draw a horizontal short
VERT SHORT	Draw a vertical short
AUTO ERASE ON	Set mode to erase and redraw the display whenever a deletion occurs
AUTO ERASE OFF	Opposite of AUTO ERASE ON (Normal program mode)
ERASE AND REDRAW	Erase the screen and redraw the picture immediately
QUIT	Terminate program execution, returning control to the timesharing system command level
ATTACH DEVICE	Replace a node previously specified with a device to be specified
TERMINATE	Terminate the device definition which is being defined or edited
DEFINE DEVICE	Begin a device definition, saving the present state of the main circuit for use when the definition is terminated
EDIT DEVICE	Retrieve a previously defined device for editing, saving the present state of the main circuit until the editing is terminated

The TERMINATE light button only occurs on menu 2 and the DEFINE DEVICE and EDIT DEVICE buttons only occur on menu 1. All other light buttons are on both menus.

IV. MODELING THE CIRCUIT

If the program is to deal with the circuit, it must have access to certain information about it. For instance, one might want to know what is the value of capacitor "C3". To answer this question, the program must have stored the fact that there is an element named "C3" which has a certain value. But a circuit is more than just data of this type. Editing operations, such as deleting a node, require a knowledge of many relationships; to delete a node, one must remove all of the elements of any type attached to it. This type of information will be called structure. Each node or element in the circuit has certain data pertaining to it and there is a structure which interrelates the nodes and elements.

The following data items are stored for each node:

1. A "name".
2. An X and Y position in some system of coordinates.
3. The number of elements attached to the nodes.

The structural information for each node specifies which elements are attached to the node.

For a resistor or capacitor, the program stores as data:

1. The value of the resistance or capacitance.
2. A "name" so the user can refer to the element.

The structural information for an element specifies the two nodes to which the element is connected. For an element representing a short circuit one need store, in this example, only the nodes between which the short is connected. All of the data plus the structure indicating

relationships between the individual parts of the model is called the model's data structure.

Many computer representations of these elements and their interconnections can be devised. To create a sample program we shall use the AED (Algol Extended for Design) System which has been developed by the M. I. T. Electronic Systems Laboratory Computer Applications Group and which operates on several computers, including the IBM 360 and 7094, and the UNIVAC 1108. The circuit will be modeled by using a bead, or block of contiguous storage, to represent each node or element. Each bead contains a code to indicate what type of circuit component it represents and also all the information (both data and structure) to be stored for the component. The connection of circuit components is implemented by the use of pointers; an element bead will contain a pointer which will allow the program to locate the node bead corresponding to the node to which the element is connected. Figure 1b shows the format of the data structure beads used in the program. In a resistor bead, for example, the code to indicate that the bead represents a resistor is stored in the word labelled TYPE. FROMNODE and TONODE are pointers to the node beads representing the nodes between which the resistor is connected. NAME is a pointer to the character string specifying a unique name for the resistor. VALUE is the numeric value of the resistance. Each item in a bead can be referenced symbolically by the use of the AED components feature; an item in a bead such as FROMNODE will be referred to in subsequent sections as a component of the bead. Beads are obtained from a free storage system (a storage management

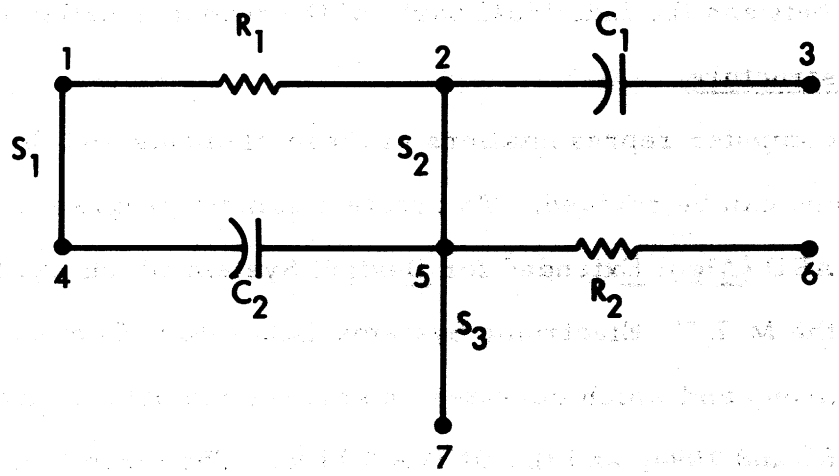
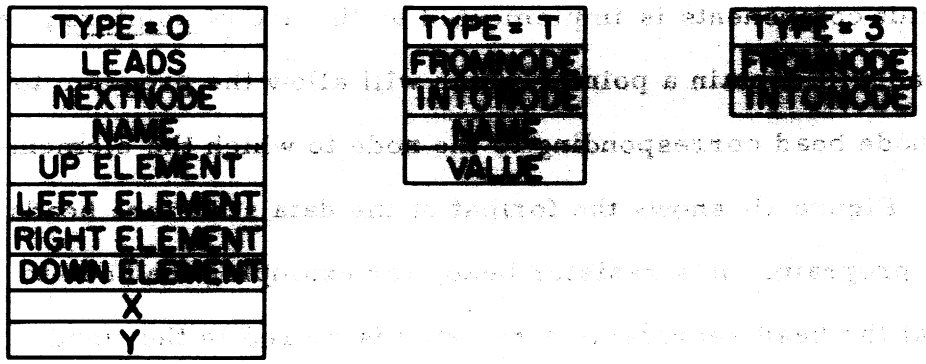


Fig. 1a Schematic Diagram of a Partially Completed Circuit



Node Bead T = 1 Resistor Bead Short Circuit Bead
T = 2 Capacitor Bead

Fig. 1b Data Structure Beads used to Model the Circuit

system). The facilities of the AED system used in this program are described more completely in Section VIII.

V. PROGRAM OPERATIONS ON THE DATA STRUCTURE

To perform each of the functions of the program, operations must be performed on the data structure; the modification of the graphical representation of the circuit is only an obvious manifestation of the alteration made to the structure. To begin to draw a circuit the program must:

1. Obtain a block of storage, and store in it the code to indicate that it represents a node. We will call this block a node bead. It will contain all the information (both data and structure) about the node.

2. Store in the node bead the coordinates of the node in some coordinate system. We will use a coordinate system in which all nodes lie on grid positions separated by a Δx or Δy of 200; the center of the grid has coordinates (0, 0). The first node is drawn at the nearest grid position to the position of the graphic input device.

To draw an element (resistor, capacitor, or short) and attach it to a node, the program must:

1. Obtain an element bead which will contain all the information about the element; the type of element is stored in the TYPE component of the bead.

2. Find the node bead corresponding to the node the user pointed to with the graphic input device.

3. Modify the node bead to indicate a new relation exists, i.e., the element just created is to be "attached". Increase the count of the number of leads attached to the node which is stored in the LEADS component of the node bead.

4. Determine if a node already exists at the other end of the element. If not, one must be created and then the element must be "attached" to the new node.

"Attachment" of an element can be done in many ways, depending upon how relations are to be implemented in the computer. In this problem,

attachment means that pointers are stored in the node bead to indicate the elements which are attached to the node. Each element bead contains pointers to the two node beads representing the two nodes to which it is connected.

To delete an element pointed to by the graphic input device the program must:

1. Find the element bead corresponding to the element to which the user pointed with the graphic input device.

2. Locate the nodes to which the element is attached, and set the pointers to indicate that the element is no longer connected and reduce the count of the number of leads attached to the node. If no leads are attached to the node, it can also be deleted.

3. Return the element bead to the list of available storage so that it may be used again.

In addition the graphical representation of the item must be removed from the screen.

Figure 1c illustrates the data structure for a partially completed circuit; the circuit is shown schematically in Fig. 1a. For node beads only the up, left, right and down element connections are shown; for resistor, capacitor and short circuit beads only the connections to the node beads are shown. The figure also shows the pointer chain linking all nodes. This chain, which permits the program to follow pointers from one node to another, is useful since the node chain is searched frequently. Figure 1d shows the data structure as modified by adding an element, R3, between node 3 and node 6.

A more complicated program operation is deleting a node. If a node is deleted, all elements attached to it are to be deleted. Since

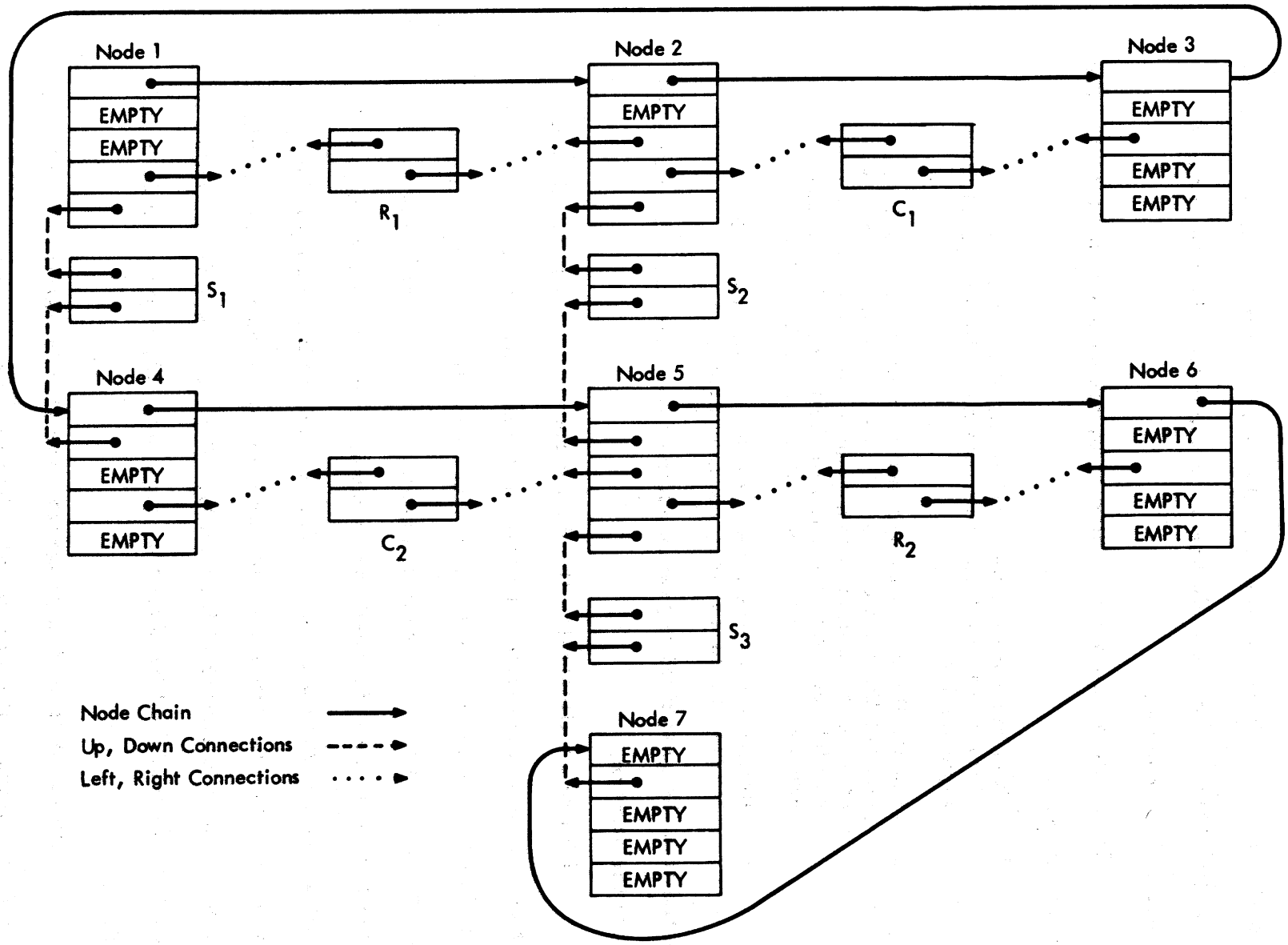


Fig. 1c Data Structure for Partially Completed Circuit

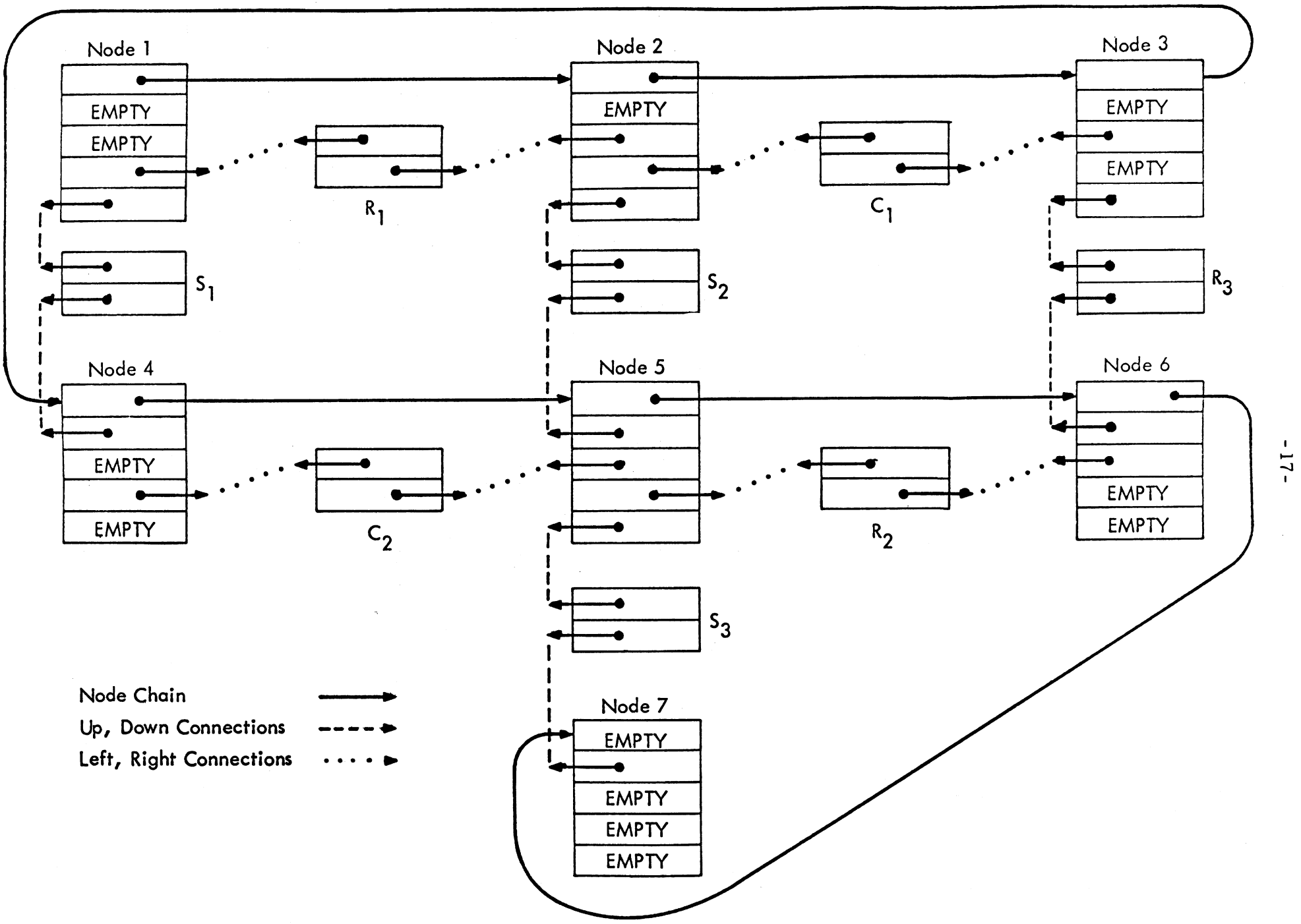


Fig. 1d Data Structure for Partially Completed Circuit with Element R_3 Added

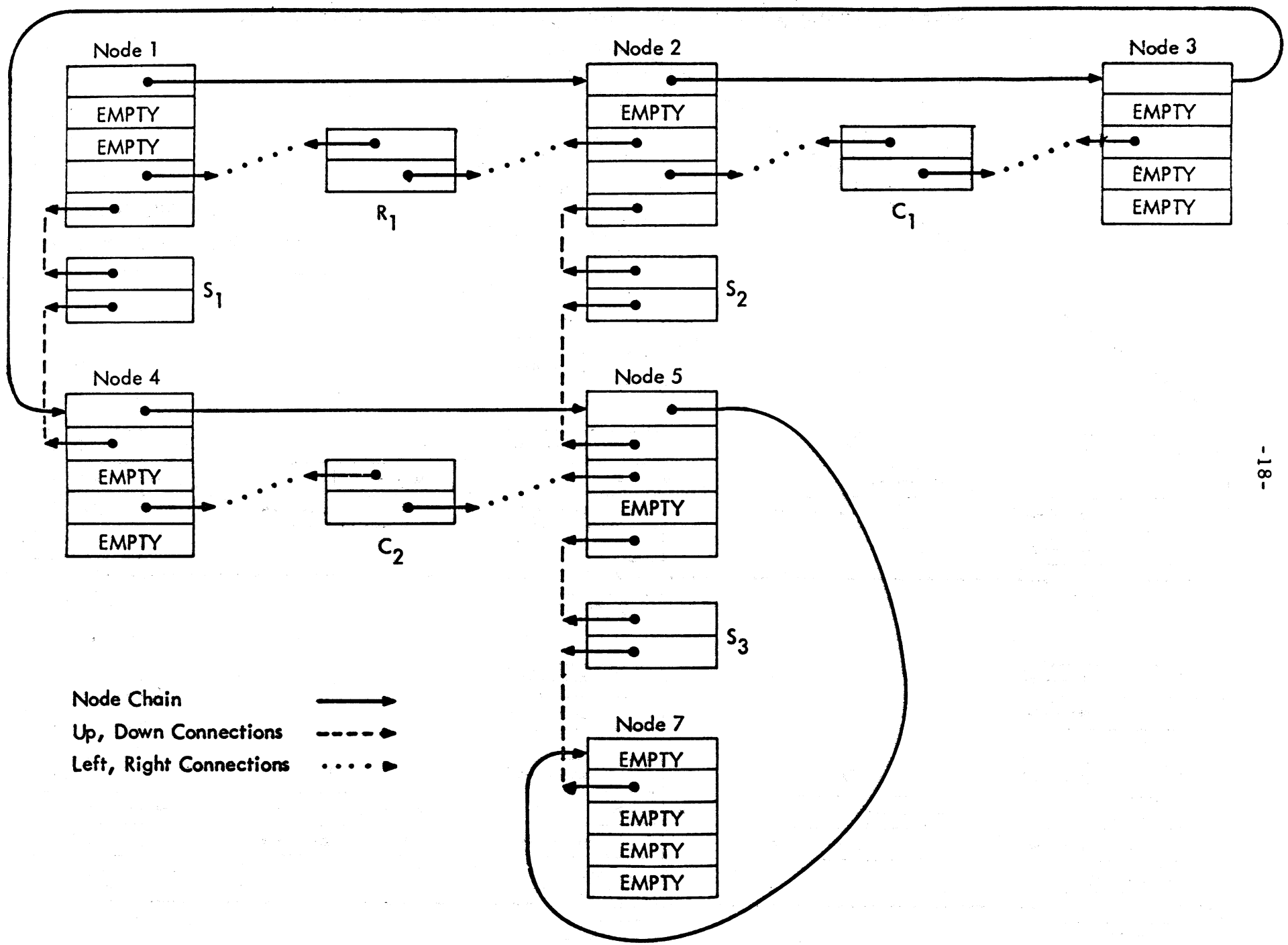


Fig. 1e Data Structure for Partially Completed Circuit with Node 6 Deleted

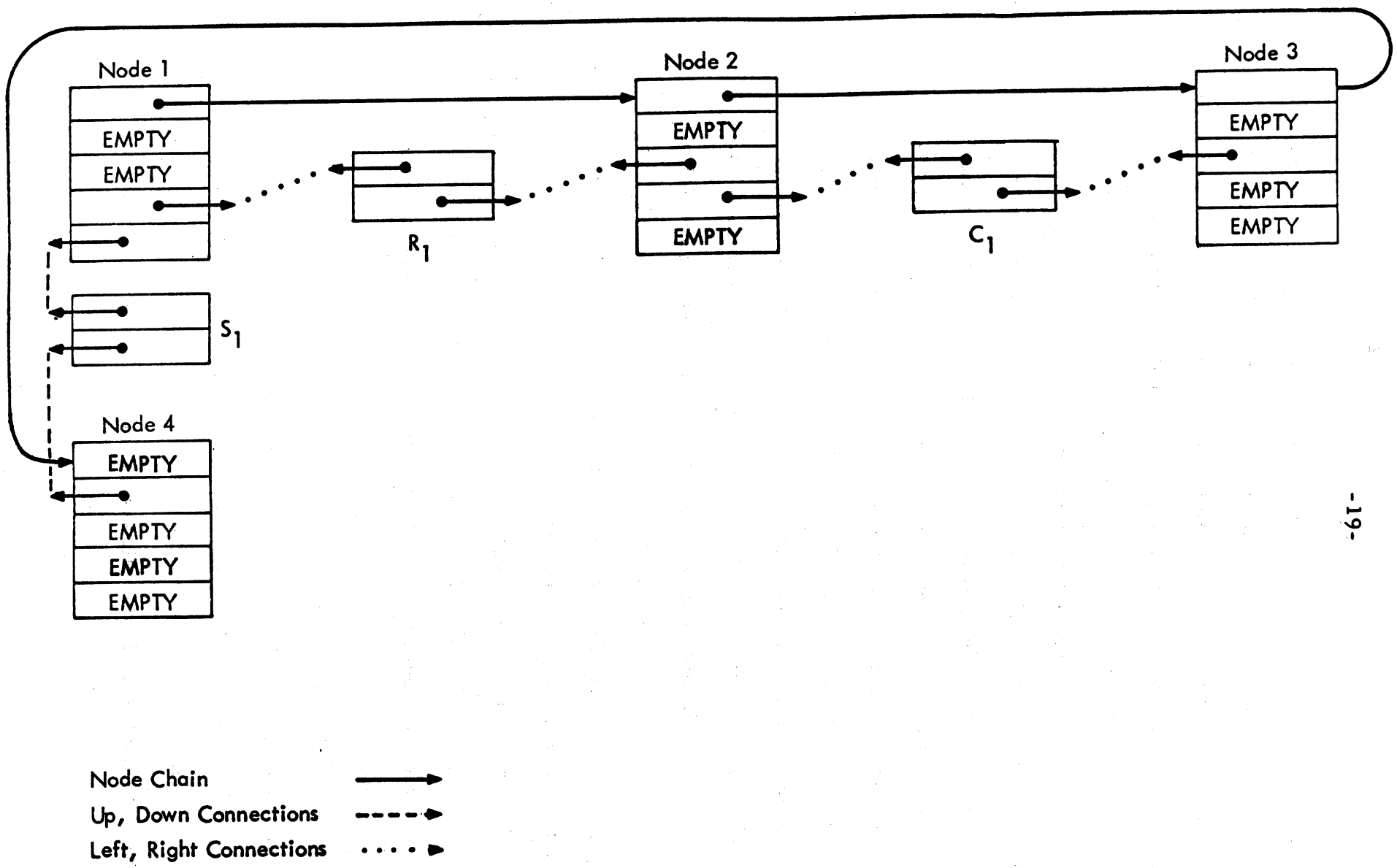


Fig. 1f Data Structure for Partially Completed Circuit with Node 5 Deleted

these elements are also attached to other nodes, one must update the information about these nodes to indicate the elements have been removed. Since the data structure implements a connection between all node beads by means of a pointer from one node bead to the next node bead, this connection must be altered to join together the nodes remaining. Of course, one must also modify the display to reflect the changes in the circuit; changing the model of the circuit in this program has no direct effect on the graphical representation of the circuit. All alterations to the display must be independently specified. Figure 1e shows the data structure in Fig. 1c after node 6 was deleted, and Fig. 1f shows the effect of deleting node 5.

The most complicated program functions are the definition, editing and usage of devices. A device must be defined before it can be used, but the definition may be begun at any time during the definition of a circuit. If a circuit, such as the one shown in Fig. 1a, is partially complete at the time a definition is to begin, the state of the circuit will be preserved until the device definition is complete and then it will be restored. The program does not allow the user to partially complete a circuit and then declare it to be a device; he must enter the device definition mode before beginning to specify the circuit which will represent the device.

The circuit representing a device, or its equivalent circuit, can contain elements, nodes and occurrences of previously defined devices. Each device has a user-supplied name and a symbol which indicates the number of leads the device has available for attachment. The maximum number of leads allowed is four, since in this program

a device is used in a circuit as a substitute for a node. Before a device definition is complete, a symbol having the correct number of leads must be selected. The correspondence between each lead on the symbol and the terminal nodes in the equivalent circuit for the devices must also be established. This correspondence is required so that an analysis program could properly substitute the equivalent circuit for the device in order to evaluate the properties of the overall circuit.

To define a device, the program must enter device definition mode and:

1. Obtain a device definition bead which will contain all the information about the device definition; it will contain no information about how the device is used.

2. Allow the user to design the equivalent circuit after having stored the circuit being constructed on the main circuit level. When the definition is complete, the main circuit will be restored. The FIRST·NODE component of the device definition bead points to the beginning of the data structure for the equivalent circuit.

3. Ask the user to provide a name to be given to the device and to select a symbol to represent the device when it is used. The style of symbol is given by the component BOX·STYLE in the device definition bead. The number of leads on the symbol selected is the number of terminal nodes in the circuit. A terminal node must be a node (or device) to which less than four elements have been connected.

4. Establish the connection between each lead on the symbol and a terminal node in the circuit.

5. Add the newly created device definition bead to the device definition chain, which links all device definitions.

The format of the device definition bead is shown in Fig. 2a; the data structure built when two devices are defined is shown in Fig. 2c.

SUBPICTURE
CORRELATION WORDS
BOX STYLE
NEXT DEFINITION
NAME
FIRST NODE
NUMBER OF USAGES
TERMINAL 1
TERMINAL 2
TERMINAL 3
TERMINAL 4

FIG. 2A DEVICE DEFINITION BEAD

CORRELATION WORD
TYPE=4
LEADS
NEXT NODE
NAME
UP ELEMENT
LEFT ELEMENT
RIGHT ELEMENT
DOWN ELEMENT
X
Y
DEVICE DEFINITION

FIG. 2B DEVICE USAGE BEAD

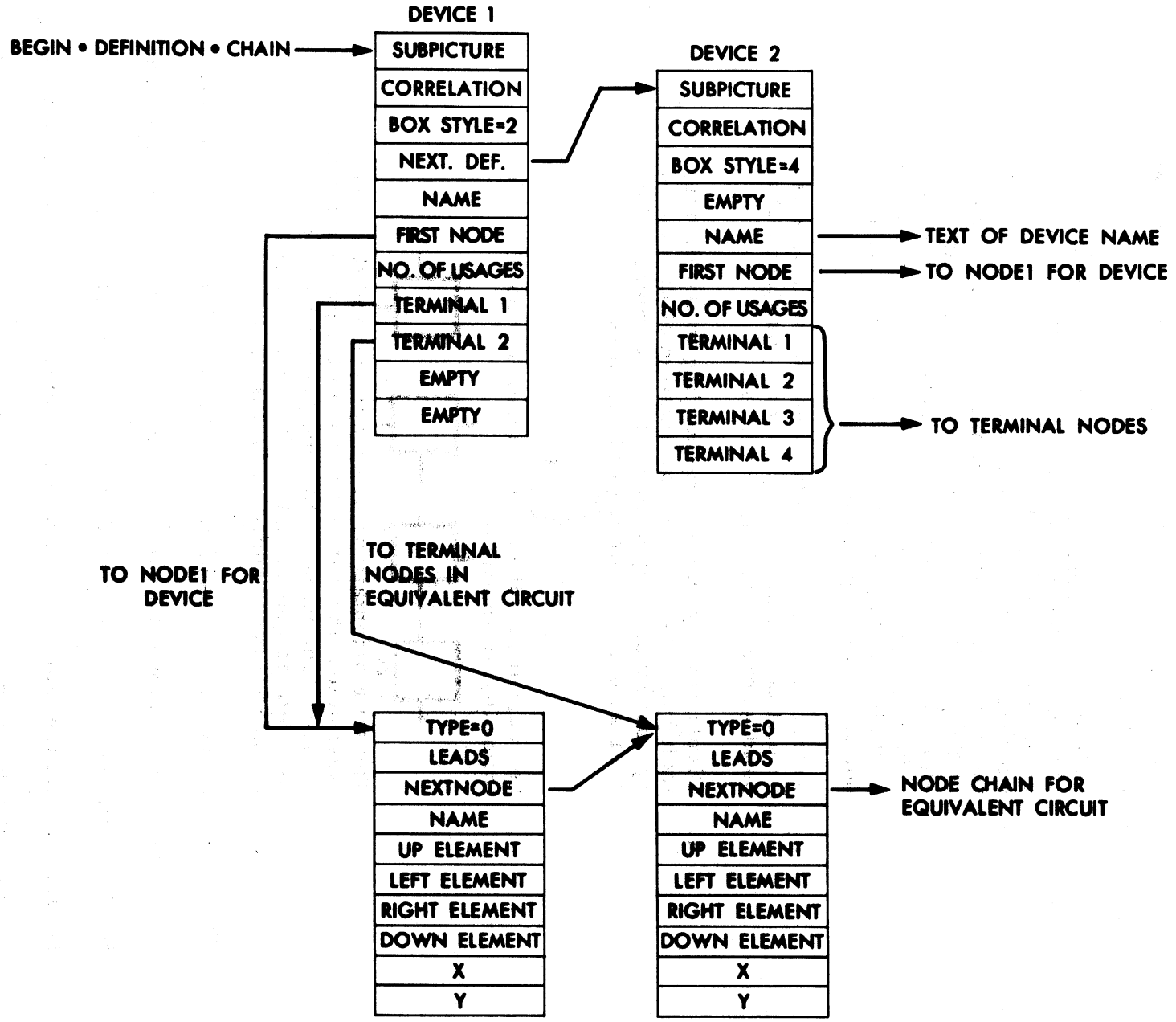


FIG. 2C DEVICE DEFINITION CHAIN FOR 2 DEVICES

To use a device in a circuit, the program must:

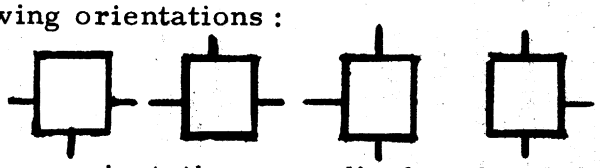
1. Determine that the device to be attached has been defined.

2. Obtain a device usage bead which will contain all the information about this usage of the device, such as the elements to which it is attached and whether or not certain leads can be used (i.e., if the device had only two terminal nodes, only two of the possible four leads can be used to attach elements). The bead also contains a pointer to the device definition bead so that the device definition could be used by an analysis program (this fact is not used in this program).

3. Delete the node at the place where the device is to be plotted. A device can only be positioned at a point where a node already exists.

4. Display the symbol corresponding to the device, including a name. The orientation of the device symbol must be specified. For example, if the user selected a symbol style such as

he could insert it into the circuit in one of the four following orientations:



The four orientations are displayed and the user must select the correct one with the graphic input device. The name is the one given when the device was defined concatenated with a number indicating the particular occurrence of the device.

The format of the device usage bead is shown in Fig. 2b. It should be noted the bead is nearly the same as a node, except that the type code is different, the existence of leads in certain directions may be forbidden, and the last component is a pointer to the device definition bead for the device being used.

VI. SPECIFICATION OF THE DISPLAY

The programming system that allows the different types of display consoles to be used with the M. I. T. time-sharing system (CTSS) has been given the name GRAPHSYS. (1, 4) The user interface is a set of procedure calls which allow the user to plot objects such as lines and points to remove objects from the screen, and to determine which object was "seen" by a graphic input device.

Although a display console deals only in terms of simple console commands such as "draw a point" or "draw a line," the user wants to deal with objects. An object is a group of console commands which are added to the display file at the same time and are to be thought of as an atomic entity, e.g., a capacitor consisting of several short, straight-line segments (made by several line-generate commands). The display file is the ordered sequence of console commands which is sent to a display console to produce a picture.

A. Naming of Display Objects

When an object is placed in the display file, it must be given a unique descriptor, or name, if it is to be identified again. The user may wish to specify some object upon which GRAPHSYS should perform a function (e.g., delete this object), or GRAPHSYS may wish to inform the user of some action concerning it (e.g., this object was seen by the light pen). The name is used to refer to an object in all communications between GRAPHSYS and the user; the name must be both unique and invariant so that an object can be uniquely identified at any time.

The form of a unique and invariant name which appears simplest, at least conceptually, is a subscript in an array. In order to perform the correlation function, which transforms (via a correlation map) the address of a console command causing a light pen interrupt to the user's name for the object, GRAPHSYS would have to build an array of length equal to the total number of names. An element in this array would contain the initial location of the console commands corresponding to the object whose name is the subscript of the element in the array. In addition, the user would need a similar array giving the item in his data structure corresponding to the name. Thus, there would be a double map, one from the user's name to the console command locations and one relating the name to the user's data structure. These maps correlate the data structure for an object with its display representation. Double maps, especially in array form, are expensive in their use of core storage.

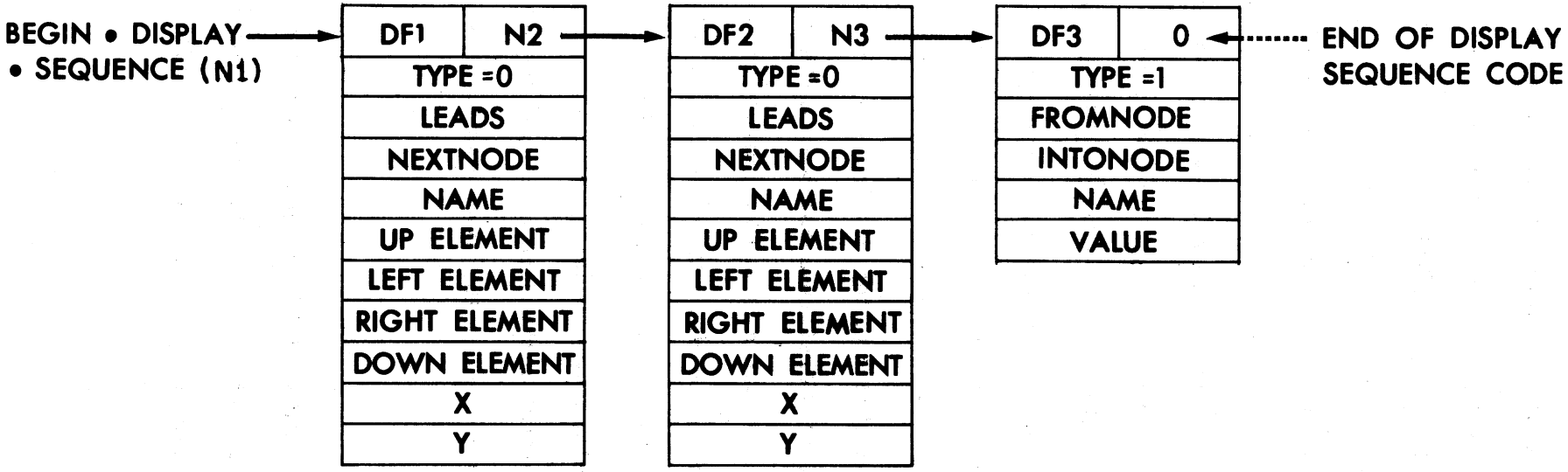
The naming scheme which has been employed by GRAPHSYS meets the criteria of uniqueness, invariance and convenience, at least within the context of an <u>in-core</u> data structure. To use the GRAPHSYS naming convention, the user extends each bead by one or two words, the <u>correlation word(s)</u> which contain the appropriate correlation information. In this manner explicit arrays are avoided,

and when the appropriate correlation word is located by the correlation mechanism, the corresponding data element (bead) is found immediately adjacent, thereby avoiding the map between the user's name and an element in the data structure. GRAPHSYS will be responsible for storing into and updating these correlation words, and performing the actual correlation. In a correlation word GRAPHSYS will store the address of the console commands for the object. The unique "name" of the object is the address of the correlation word, which is also the address of the user's data structure bead for the object.

Figure 3 illustrates the combined user-and-GRAPHSYS data structure for a display file containing three objects. In our example the correlation word(s) will occur as the first word in the entire block of storage representing an element, node or device. Each correlation word contains the name of the next object in the display sequence. The GRAPHSYS display data structure is therefore a one-way list threading through the user's data structure. A name of zero signifies the end of the list. The contents of the correlation word are maintained solely by GRAPHSYS; the user does not have to be concerned with the location of the console commands. Note also that this particular thread (list) through the beads is independent of the data associations between beads, i.e., the other list structures which the user's programs may have built.

B. Use of Subpictures

The data structure built by GRAPHSYS using the correlation words becomes more complex when subpicture calls and definitions are considered. Subpictures are defined using the procedures DEFSUB and



N1, N2, N3 are pointers to correlation words of objects; they are the GRAPHSYS "names" for the objects.

DF1 DF2, DF3 are the display file locations of display commands corresponding to the objects.

Figure 3. GRAPHSYS Data Structure

ENDSUB which work like BEGIN and END in PL/I or AED. Subpicture definitions may include calls to other subpictures to any depth, except that subpicture calls can not be recursive. Although subpictures may be defined within subpictures, they may still be called from outside those subpictures, i.e., subpictures do not possess a PL/I-type block structure. They behave exactly as if they had all been defined in parallel at the highest level.

A separate list is built by GRAPHSYS linking all subpicture definitions. Each element in this list is the start of a subpicture definition and contains 3 pieces of information:

1. The number of times the subpicture is called.
2. A pointer to the first correlation word in the subpicture definition. The list beginning with this correlation word is of the same format as shown in Fig. 3.
3. A pointer to the next subpicture in the subpicture definition list.

A subpicture call requires two correlation words which contain:

1. A pointer to an element in the subpicture definition list, i.e., a pointer to the beginning of the definition for the subpicture being called.
2. A pointer to the next correlation word, i.e., the next object in the display sequence following the call.
3. The display buffer address of the subpicture call (for refresh-type displays).

This structure is built automatically by GRAPHSYS and the user's only responsibility is to provide the storage used for the correlation words. Figure 4 provides a schematic view of the structure built in the following case:

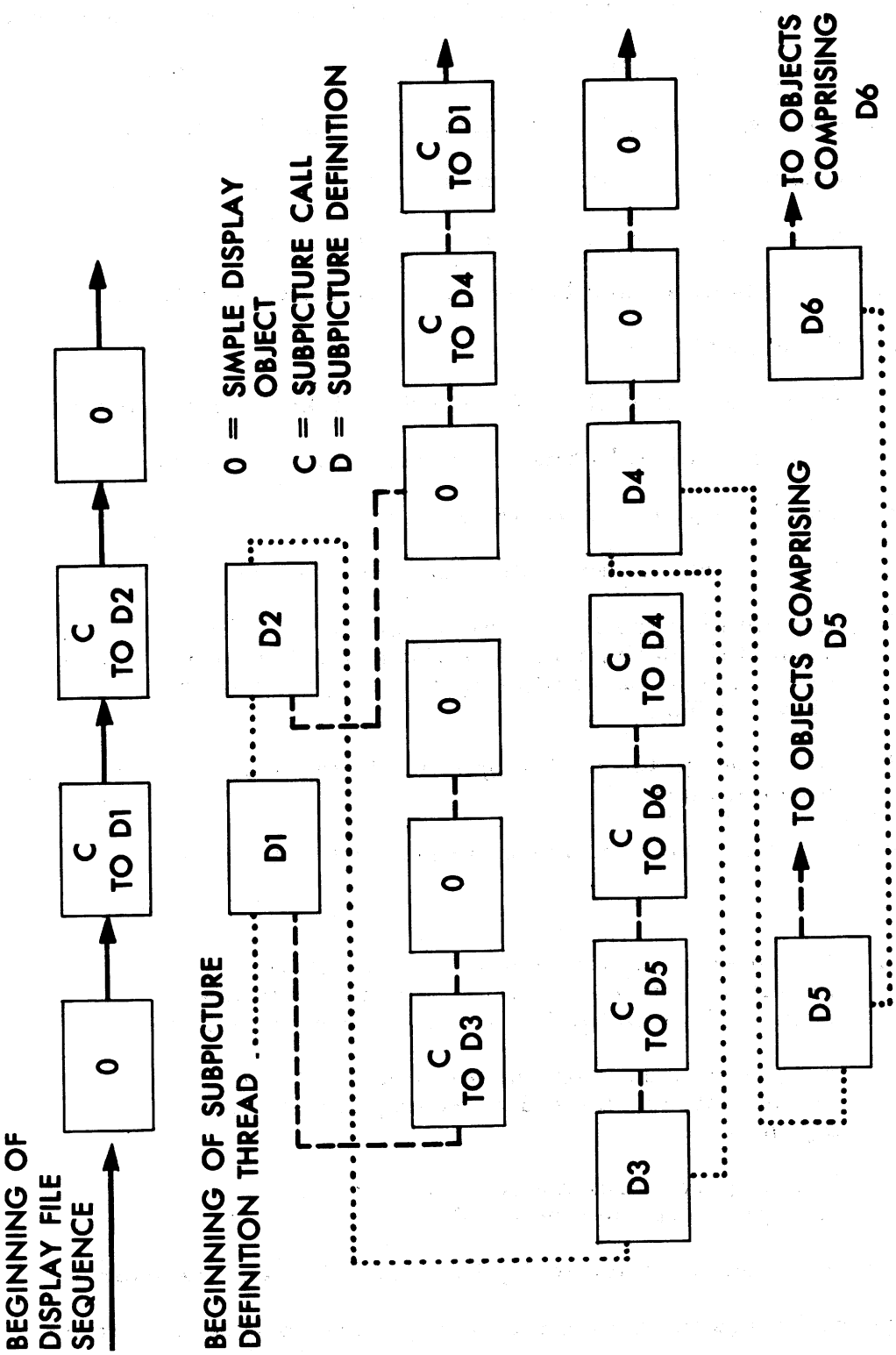


FIG. 4 GRAPHSYS DATA STRUCTURE WITH SUBPICTURES

1. The main display file contains an object and 2 subpicture calls, to subpictures D1 and D2.

2. The subpicture definitions corresponding to these calls are D1 and D2.

3. D1 contains a call to definition D3, and D2 contains a call to definition D4 and to definition D1.

4. D3 contains only subpicture calls, to definitions D5, D6 and D4.

The data structure built when a subpicture is defined retains the information about how the subpicture is built in terms of individual display objects. However, the same pictorial representation can be obtained by defining what is called a compound object in GRAPHSYS. A compound object is produced by combining physically the display commands from all the objects specified in order to create the display code for the new compound object. The difference between compound objects and subpictures is analogous to the difference between compiling and interpreting computer programs. In a compiled program the input statements cannot be deduced by examining the compiled code; in an interpreted program, the structure of the statements is retrievable. The data structure created at the time a subpicture is defined permits a 'pensee' to be decoded as referring to a certain object within the subpicture; in terms of the analogy it is easy for a language interpreter to report the source language statement corresponding to an execution-time error. However, if 'pensee' occurs upon a compound object, there is no information available about which object of those originally used to create the object was responsible for the pensee. Compound objects should be used when the object is the smallest piece of the picture to which the user expects to make

reference; subpictures preserve the structure used in the definition and therefore should be used if this structure is important.

When the user in the sample program defines a device, it is represented internally as a subpicture since the user may wish to modify the definition of the device in terms of nodes, devices, and elements comprising it. In order to display the equivalent circuit for the device, the subpicture for it is called. Devices may be defined in terms of previously-defined devices; this hierarchy is represented by a subpicture definition for the new device, which will contain a subpicture call to the definition for the previously-defined device. When an object within a subpicture is "seen" by the graphic input device, the identity of the subpicture call and the corresponding subpicture definition is available, in addition to the identity of the object upon which the pensee occurred. If subpicture definitions are nested, the entire subpicture hierarchy is returned upon a pensee; it is up to the user to decide how much of this history of formation is meaningful at that time.

C. Correlation of Graphic Input with Problem Data Structure

When a display with a light pen is being employed, the display hardware provides, at the time a light pen interrupt occurs, the X and Y beam coordinates and the address in the display buffer from which the current display instruction was fetched. In order to report to the user the object "seen" by the light pen, GRAPHSYS searches the data structure shown in Fig. 3, serially via the correlation word pointers for each bead (object), checking if the address resulting from the light pen interrupt corresponds to any of the display buffer locations containing the display commands to draw this object. If, for example, in

Fig. 3 the address at the time of the light pen interrupt was DF2+3 and if the object starting at DF2 had 7 display commands, the address of the correlation word of the object seen would be found by the search to be N2. Since the correlation word in this program is always the first component of the data structure bead describing the object, the user has immediate access to the data about the object.

Unfortunately, in a non-refreshed display system, such as is used in this example, the address corresponding to the display commands for the object cannot be obtained by an interrupt since only the X-Y coordinates of the cursor are available. GRAPHSYS must decide which object the user tried to designate with the cursor. The most convenient terminology to describe the correlation of graphic input with objects is by analogy with the use of a light pen. Therefore, we will use the term "pensee" to indicate a correlation between a graphic input device and a displayed object and "pen sensitivity" to indicate an object is eligible for such a correlation.

The algorithm used assumes that the area within which an object is pen sensitive is a rectangular area associated with the object. To make a displayed item "pen-sensitive" the user calls the procedure PENSNS, giving the size and the relative position of the rectangle. A special bit is set in the correlation word to indicate it is pen sensitive. If he does not wish to specify the position of the rectangle, a program-computed center is used. When the user indicates he wants to identify an object being pointed to with the cursor, GRAPHSYS searches the data structure illustrated in Fig. 4 for the first pen-sensitive object having a rectangle within which the cursor coordinates fall. A pen-

sensitive subpicture call requires that the subpicture definition be searched; upon a pensee, the name of object seen, the name of subpicture definition which includes the object, and the name of the subpicture call are returned to the user. If the subpicture call is within a subpicture definition, the names of all outer subpicture calls and definitions are also returned.

When the pen-sensitivity package is being used, the model of the picture maintained in the computer must correspond exactly to the picture on the screen. If the two differ, the user could point to an object which his program believes has been deleted, or the pen-sensitivity program may "see" an object which has not yet been displayed on the screen. In the program an object which has been deleted will be written over with a large X to indicate that it has been removed from the problem and display file data structures, but it will remain on the screen until the picture is redrawn.

D. GRAPHSYS Procedures

A variety of GRAPHSYS procedures are used in this example; they can be divided into six classes:

1. Those procedures for creating standard display objects such as lines, point, and character strings (procedures LIN, SETPT, TEXT).

2. Those procedures for modifying display objects, such as those procedures to make a line invisible, a line dotted, or to make an object pen-sensitive (INVIS, DOTTED, PENSNS).

3. Those procedures required in order to use compound objects, which are objects consisting of several standard objects; for example, an object consisting of several lines and a string of text (DEFOBJ, ADDOBJ, ENDOBJ, CRYOBJ).

4. Those procedures required in order to use sub-pictures (DEFSUB, ENDSUB, CALL).

5. Those procedures which add objects to the display file data structure and those which transmit the display file, or a portion of it, to the terminal (PLOT, DISPLAY).

6. Those procedures which alter the display file data structure by removing objects or replacing objects with other objects (RMV, RPL).

All GRAPHSYS procedures used in this program which return a value (or in FORTRAN terms are functions rather than subroutines) have a pointer as their value. A GRAPHSYS standard object is a group of display console commands which are added to the display file at the same time, and which are thought of as an atomic entity, i.e., a string of characters. The internal representation of an object is a block of storage containing the display commands with a header denoting the number of machine words in the block; the object is referenced via a pointer, which will be denoted in the procedure descriptions as PTR. A compound object consists of several standard objects which have been merged to form a single object, which is also referenced via a pointer.

Calling Sequences of GRAPHSYS Procedures

1. Procedures to create standard objects

   a. Pointer Procedure LIN
   Purpose: To build a standard object for a line.
   Calling Sequence: PTR=LIN (DELX, DELY)
   Where: PTR is a pointer to the object created,
   DELX is the Δx component of the line,
   DELY is the Δy component of the line.

   b. Pointer Procedure SETPT
   Purpose: To build a standard object for a point.
   Calling Sequence: PTR=SETPT (X, Y)
   Where: X is the horizontal coordinate of the point.
   Y is the vertical coordinate of the point.

   c. Pointer Procedure TEXT
   Purpose: To build a standard object for a text string.
   Calling Sequence: PTR=TEXT (.C. string)
   Where: .C. string is a text string generated by the AED
   .C. operator, i.e., .C. /DELETE/ (see Sect. VIII).

2. Procedures to modify standard objects

   a. Pointer Procedure INVIS
   Purpose: To make all lines or points in an object invisible.
   Calling Sequence: PTR=INVIS (PTR)
   Where: PTR is a pointer to an object, such as is produced
   by LIN or SETPT.

   b. Pointer Procedure DOTTED
   Purpose: To make all lines in an object dotted.
   Calling Sequence: PTR=DOTTED (PTR)

   c. Pointer Procedure PENSNS
   Purpose: To make an object pen-sensitive.
   Calling Sequence: PTR=PENSNS (PTR)

3. Procedures for constructing and using compound objects

   a. Procedure DEFOBJ
   Purpose: Begin a compound object definition.
   Calling Sequence: DEFOBJ ( )
   Arguments: None
   DEFOBJ must be called before ADDOBJ can be called to add
   standard objects to a compound object.

b. Procedure ADDOBJ
Purpose: To add a standard object to the compound object currently being constructed.
Calling Sequence: ADDOBJ (PTR)

c. Procedure CPYOBJ
Purpose: To make a copy of a standard or compound object.
Calling Sequence: NEWPTR=CPYOBJ (PTR)
Where: NEWPTR is a pointer to the copy.

d. Pointer Procedure ENDOBJ
Purpose: To terminate the definition of the compound object and to get a pointer to the definition.
Calling Sequence: PTR=ENDOBJ ( )
Arguments: None

4. Procedures for constructing and using subpictures

a. Pointer Procedure DEFSUB
Purpose: To start defining a subpicture.
Calling Sequence: SUBNAME=DEFSUB (SUBNAME*)
Where: SUBNAME is the pointer to a block of two words which are to be used as the correlation words for the subpicture definition. If SUBNAME is not provided, DEFSUB will obtain two words from free storage and the value of the procedure will be a pointer to the words obtained. Upon a pensee within the subpicture definition SUBNAME will be among the information returned in order to designate which definition included the object seen. A * indicates an argument optional and need not be supplied.

b. Procedure ENDSUB
Purpose: To terminate the definition of a subpicture.
Calling Sequence: ENDSUB ( )
Arguments: None

c. Pointer Procedure CALL
Purpose: To call a previously designed subpicture.
Calling Sequence: NAME=CALL (SUBNAME, NAME*)
Where: SUBNAME is the argument supplied to DEFSUB when the subpicture to be called was defined.
NAME is a pointer to a block of two words which are used as the correlation words for the subpicture call, Upon a pensee within the subpicture definition, NAME will be among the information returned in order to designate which call of the subpicture definition corresponds to the pensee.

5. Procedures to add, modify and transmit the display file

a. Pointer Procedure PLOT
Purpose: To add objects to the display file.
Calling Sequence: NAME=PLOT (PTR, NAME*, AFTER*)
Where: PTR is a pointer to the standard or compound object to be added to the display file.
NAME and AFTER are optional arguments which may be omitted. NAME is a pointer to a word of storage to be used for the correlation word for the object. If NAME is not provided, PLOT will obtain a word from free storage and the value of the procedure will be a pointer to the word obtained. AFTER is a pointer to the correlation word of the object after which, in the display sequence, the new object should be plotted. If AFTER is not provided, the new object will be added to the end of the display file.

b. Procedure DISPLAY
Purpose: To send the display file, or a portion of it, to the terminal.
Calling Sequence: DISPLAY (HERE*, THERE*)
Where: HERE is a pointer to the correlation word of the first object to be displayed.
THERE is a pointer to the correlation word of the last object to be displayed. HERE must precede THERE in the display sequence. If no arguments are provided, the entire display file will be output to the terminal.

c. Procedure RMV
Purpose: To remove objects from the display file.
Calling Sequence: RMV (HERE*, THERE*)
Where: All objects between HERE and THERE will be removed from the display file. HERE and THERE have the same meaning as for DISPLAY.

d. Pointer Procedure RPL
Purpose: To replace an object in the display file with another.
Calling Sequence: NAME=RPL (NEW, OLD, NAME*)
Where: NEW is a pointer to the object to replace OLD.
NAME is the pointer to the correlation word of the new object. If NAME is omitted, the system will supply the correlation word, as is done in PLOT.

VII. INPUT PROCESSING

The sample program can receive input from both the keyboard and graphic input devices, and it is useful to consider the graphics terminal user as having an "extended keyboard" consisting of a typewriter, light pen and/or position input devices, push buttons, etc. In the sample program all input is handled via one input routine, GET.ITEM, which is called with the type of input item to be read.

In the call to GET.ITEM the allowable type of item expected by the program is indicated. In this program the program flow is controlled by the use of "light buttons" and only one type of input is allowable at a given time. In case of an error GET.ITEM could indicate to the user the type of input he was expected to provide and give him an opportunity to correct his error.

GET.ITEM uses a special purpose AED system RWORD processor (5). The allowable input types in the program are:

1. Numeric value.
2. Character string.
3. Graphic input item containing the absolute position of the cursor.
4. Graphic input item containing the information regarding a "pensee" (X,Y cursor coordinates, pointer to the correlation word of the object seen, and the stack of subpicture calls and subpicture definitions, if any). This item will replace an item of type 3 if the cursor is positioned over a pen-sensitive picture element.

GET.ITEM calls upon system routines that examine the contents of the input buffer and decide what types of input items have been received, performing type conversions when necessary and doing some processing such as the cursor correlation function. Different versions of GET.ITEM are required for each type of display console, due to different input message formats, but the user's interface to GET.ITEM is independent of the type of display upon which he is working.

The use of GET.ITEM allows the graphics programmer not to worry about attention or interrupt handling. Input from a display terminal is handled by the M.I.T. timesharing system (CTSS) in much the same way as it handles input from typewriters. When an executing program requests input from the terminal and none has been provided, the timesharing supervisor places the program in "input wait" status and it is removed from the queue of programs waiting to execute. The ARDS transmits information on a character-by-character basis via its telephone line connection to a small communications computer, which assembles a line of input. When the "carriage return" character is received, and the user's program is in input wait status, the timesharing supervisor removes the user's program from input wait status and queues it to run; it can then read the input.

If, on the other hand, the user's program is executing when input is received, the program is not interrupted and the program must specifically request the input in order to determine if it has arrived. The user's

program can read the line of input and act upon it in any way. There may be a considerable time delay between the time the user indicates an action is to be performed and any confirmation is received from his program. All information from the ARDS is transmitted in terms of characters; for example, the cursor position is encoded as a sequence of character codes. The M.I.T. timesharing system therefore simply regards the ARDS as a "high-speed teletype", although other techniques for handling the terminal in timesharing may be preferable for some applications.

If the user of a display console equipped with a light pen and function keys is working continuously, but his program receives only intermittent service in the timesharing system, input from the display console must be buffered in some way so that his program does not lose any of the input intended for it. In the M.I.T. timesharing system, the highly interactive consoles are connected to small computers (PDP-7 or PDP-9) which handle all real-time interrupts. When the small computer has some information for the user's program in the timesharing system, it signals that it has a message which can be read by the timesharing system at its convenience; if the user's program is waiting for input, it will be queued for execution. The small computer can confirm that the user's graphic input has been received, and "package" the information in a convenient form for transmission to the timesharing

system; in the case of a pensee the message would include the pen coordinates, the address in the small computer memory corresponding to the interrupt and the subpicture call stack, which is needed in order to fully interpret a pensee. When the user's program begins to run on the timesharing system, the message received from the display buffer computer can be processed.

VIII. AED LANGUAGE FEATURES

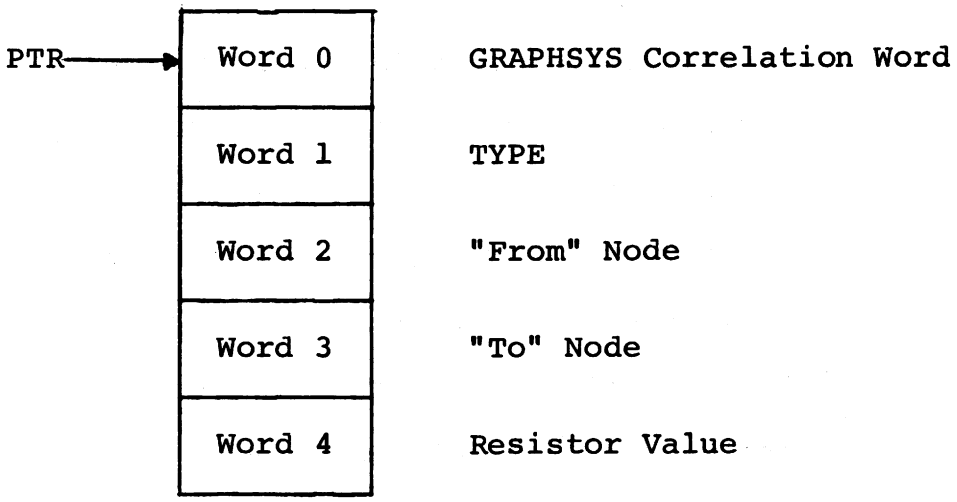
In order to study the sample program, one must be familiar with some of the features of the AED language in which it is written. AED is an extension of ALGOL and includes many features especially suited for building large systems and for manipulating complex data structures. (6,7) A knowledge of ALGOL sufficient to understand the sample program in detail can be obtained from any ALGOL primer, such as the one written by McCracken (8).

The most important AED features used in this program which are not included in ALGOL are the free storage system, the representation of character strings, pointer variables, and components. In our previous discussion we have used the term "bead" to describe a block of storage used for some particular purpose, such as to represent all the information about a resistor. To create a new bead by obtaining it from a list of available storage, the system procedure FREE is called as follows:

PTR = FREE (N) $, (the $, is the AED statement delimiter)

The result of executing this statement will be that the pointer variable PTR will point to a block of contiguous storage N-words long. A pointer in AED on the IBM 360 or 7094 has as its value an absolute core location; the value of PTR is the address of the first word of the block of storage allocated by FREE. The bead can be returned to the list of available storage by calling procedure FRET.

If such blocks are to be useful, the programmer should have a way to refer to individual items within a block; for example, in working with a resistor bead, he would want to refer individually to the name and value of the resistor and also to each of the two connections to the nodes. The ability to name items within a bead is provided by the AED "component" declaration statement and the $=$ assignment operator. Assume that a bead representing a resistor is to have the following format:



To indicate that each bead would have this layout, one would first have to decide to use mnemonics such as TYPE, FROM, INTO and VAL for words 1-4 of the bead. These names would then be used in the declaration statements:

```
INTEGER COMPONENT TYPE $,
REAL    COMPONENT VAL   $,
POINTER COMPONENT FROM, INTO $,
```

Note that unlike arrays, each component of a bead can have a different type.

In order to use a component one must indicate in which bead the component is to be found, since there will be many similar beads all representing resistors. The pointer returned by the call to FREE uniquely identifies the bead in which a component is located; the AED notation to assign a value to VAL in the bead whose pointer is P would be

```
VAL (P) = 7.2 $,
```

(This is read "Val of P equals 7.2".)

Obviously the system must know which word in the bead pointed to by P the user considers to be VAL; this assignment is carried out by the $=$ operator which is evaluated at compile time. To set VAL to be the 4th word in all resistor blocks, one would insert the statement

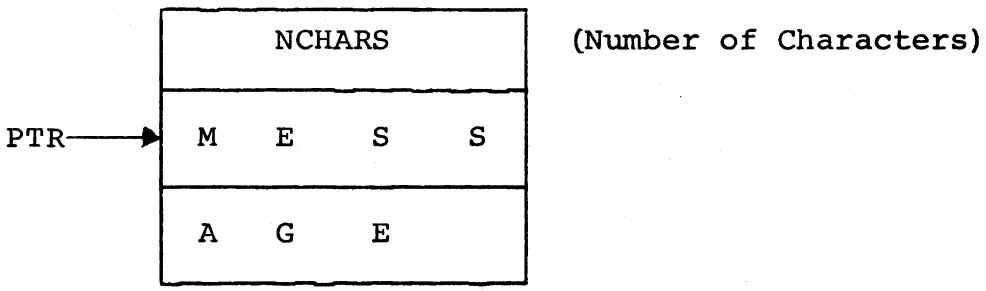
```
VAL $=$ 4  $,
```

in the program following the statement where VAL was declared to be a component. At execution time the expression VAL (P) = 7.2 $, is evaluated as "take the location given by the pointer P, add 4 to it and store 7.2 in that location."

Character strings are manipulated in AED by the use of the .C. operator and pointers. For example, the statement

```
PTR =  .C.  /MESSAGE/  $,
```

will result in PTR being a pointer to a bead containing the number of characters and the text in the following format:



PTR can therefore be stored in a data structure which has been designed without considering the maximum number of characters in the text string.

A powerful feature of the AED language is the use of phrase substitution, which permits substituting for a variable any construct whose value agrees in type with the variable. For example, suppose a programmer wants to set P to be the square root of A and transfer to statement label SOLVE if the square root is positive. In AED he could write this in one of two forms

```
P = SQRT (A)  $,
IF P GRT  0.  THEN GOTO   SOLVE  $,
```

or by using phrase substitution

```
IF   (P=SQRT (A)) GRT 0.  THEN GOTO   SOLVE  $,
```

The latter case allows the compiler to generate slightly more efficient code. A statement involving phrase substitution can always be simplified by writing out each step in turn, starting with the innermost set of parentheses.

Nested AED procedure calls are used frequently in the sample program; for example, to plot an invisible line of

length $\Delta x = 50$, $\Delta y = -50$ one writes

```
PLOT (INVIS (LINE (50,-50)))  $,
```

which corresponds to

```
PTR = LINE ( 50,-50)  $,
```

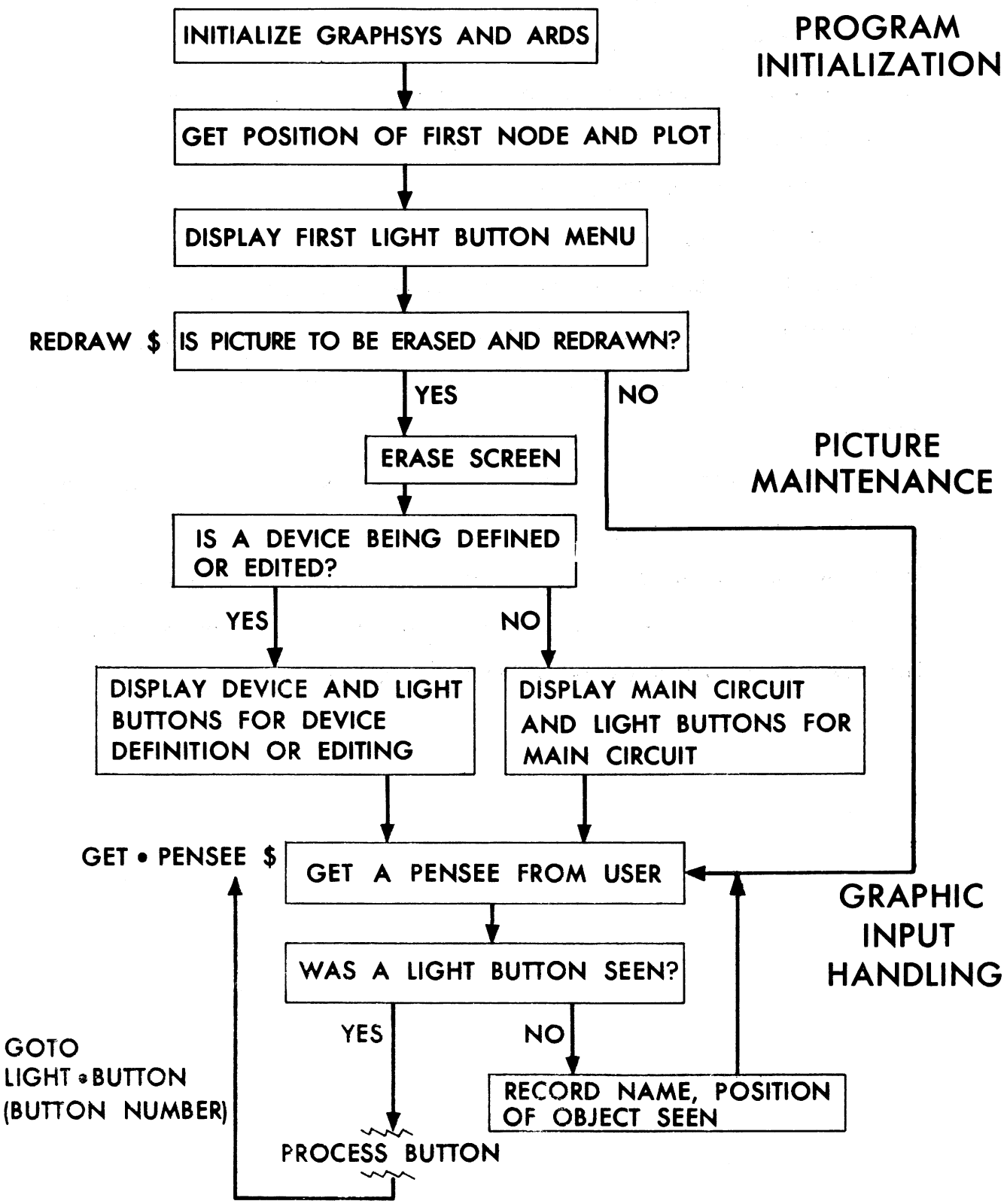
where PTR is a pointer to the object representing the desired line, followed by

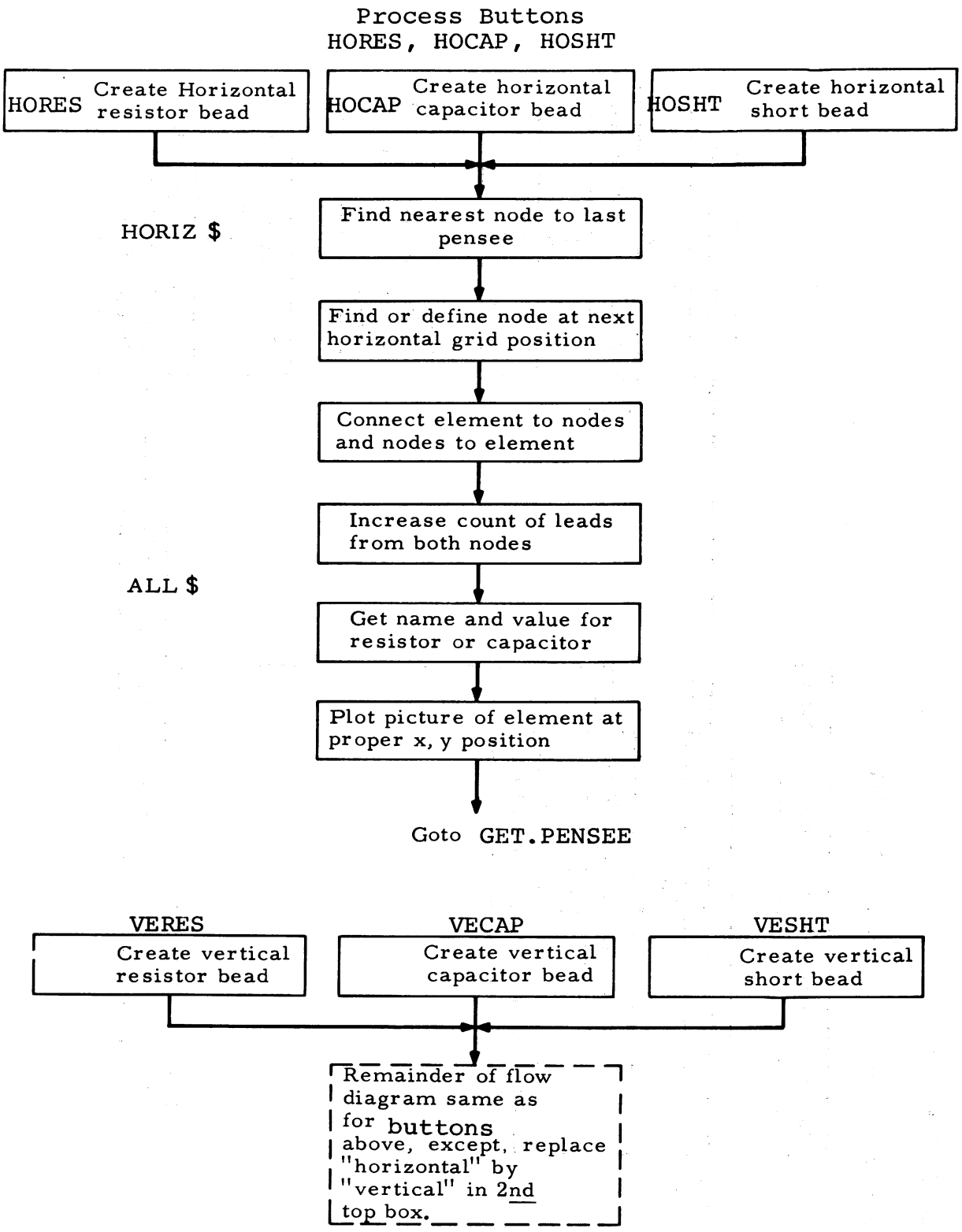
```
PTR = INVIS (PTR)  $,
PLOT (PTR)  $,
```

INVIS modifies the line command rather than making a new copy. All of the procedures in GRAPHSYS which manipulate display objects and which are valued procedures have a pointer as their value.

Other features of AED which are likely to be confusing have been commented upon in the program listing. It is hoped that the following set of flow diagrams and the comments in the program will make the program meaningful to anyone seriously interested in the details of its operation.

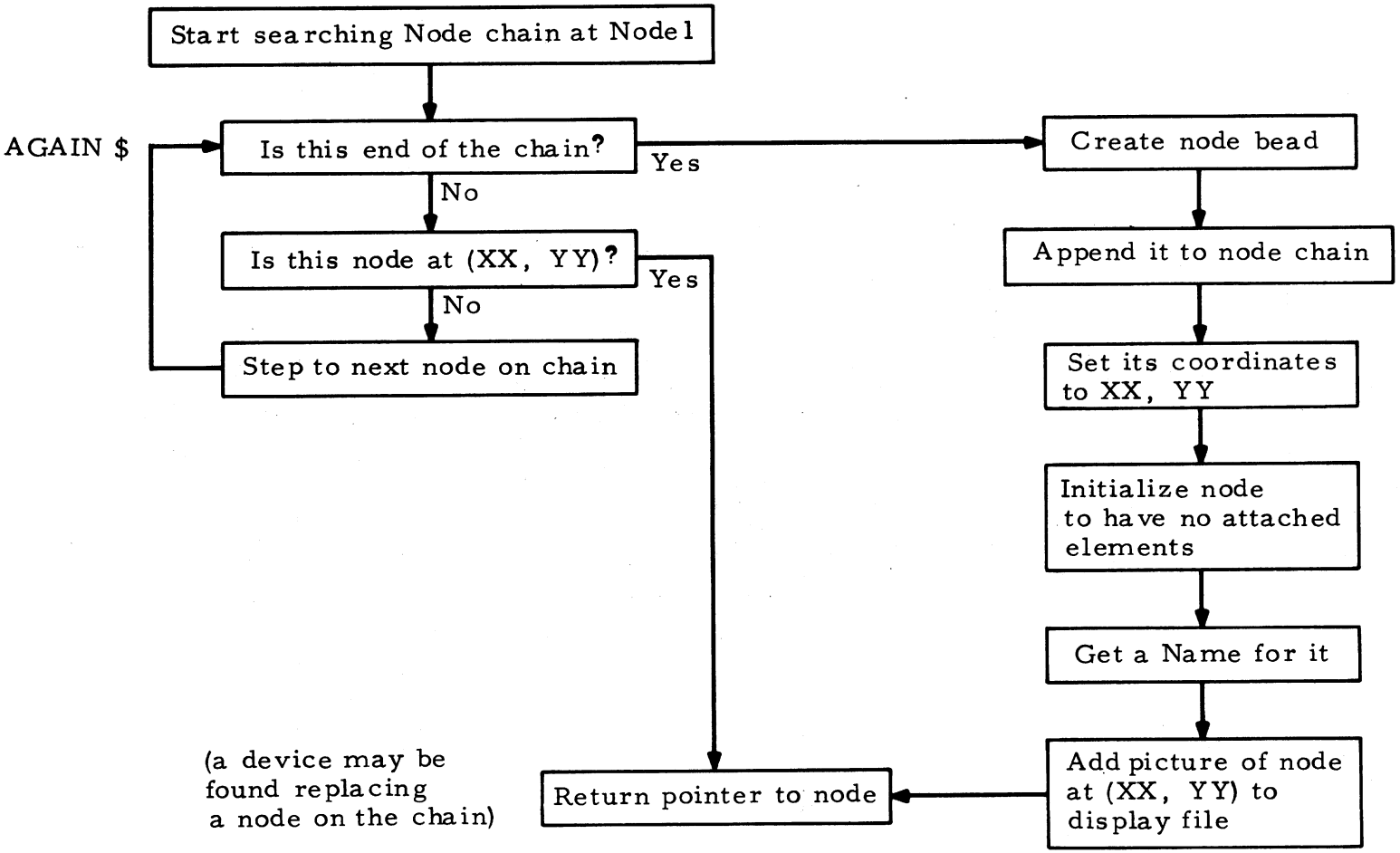
MAIN PROGRAM LOOP



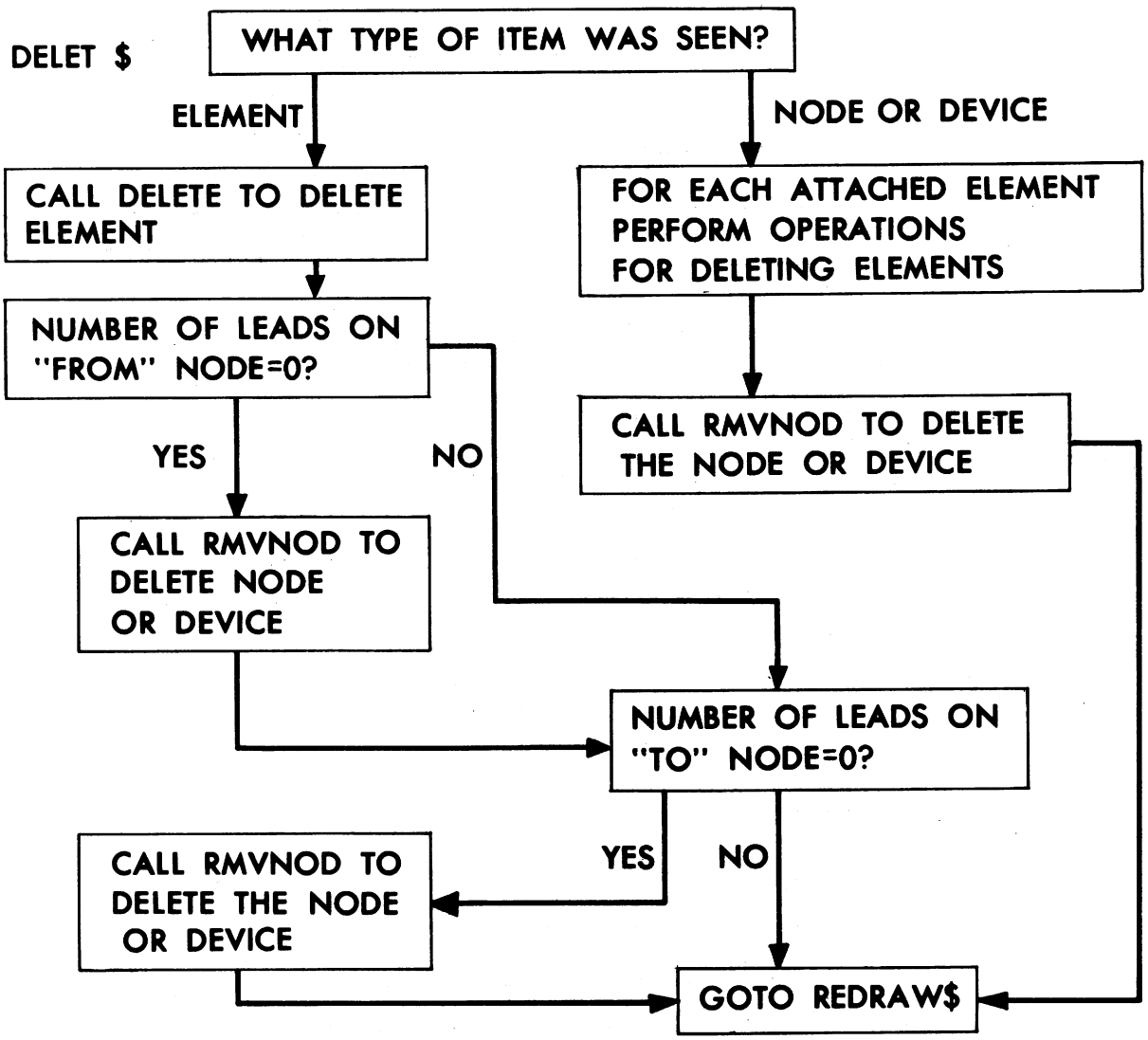

Process Buttons
HORES, HOCAP, HOSHT
HORES Create Horizontal resistor bead
HOCAP Create horizontal capacitor bead
HOSHT Create horizontal short bead
HORIZ $
Find nearest node to last pensee
Find or define node at next horizontal grid position
Connect element to nodes and nodes to element
Increase count of leads from both nodes
ALL $
Get name and value for resistor or capacitor
Plot picture of element at proper x, y position
Goto GET.PENSEE
VERES
Create vertical resistor bead
VECAP
Create vertical capacitor bead
VESHT
Create vertical short bead
Remainder of flow diagram same as for buttons above, except, replace "horizontal" by "vertical" in 2nd top box.

GETNODE(XX, YY)

Find or create Node bead at (XX, YY)

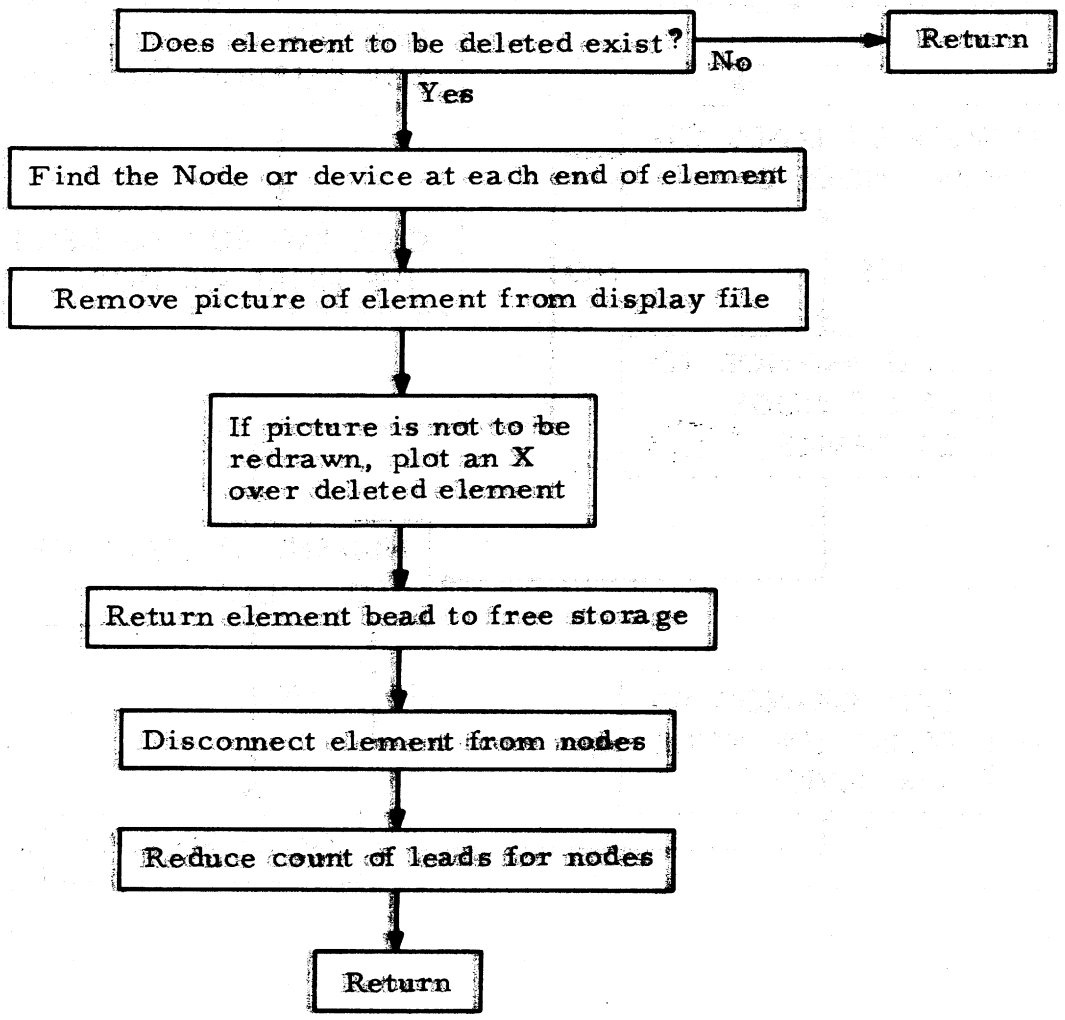


PROCESS DELETE BUTTON



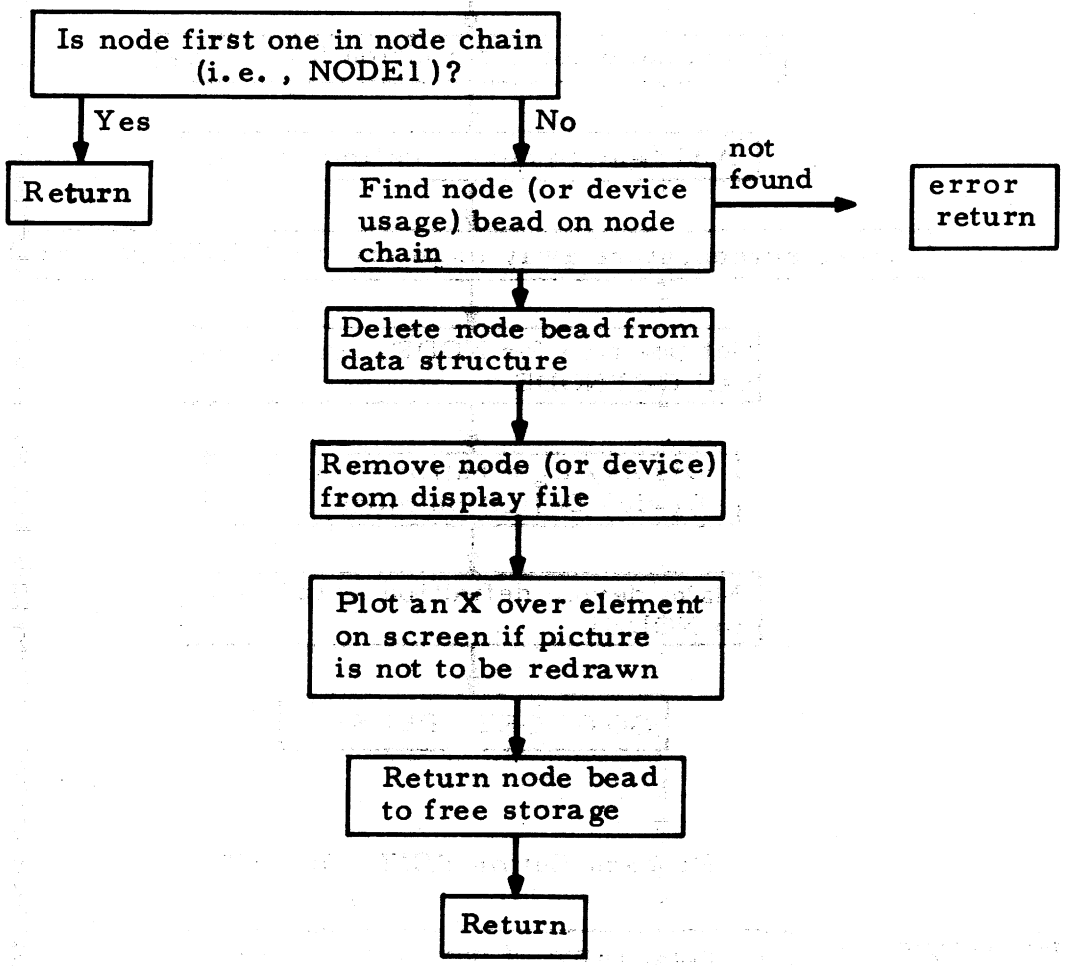
DELETE (ELEMENT, FROMNODE, TONODE)

Delete an element connected between FROMNODE and TONODE

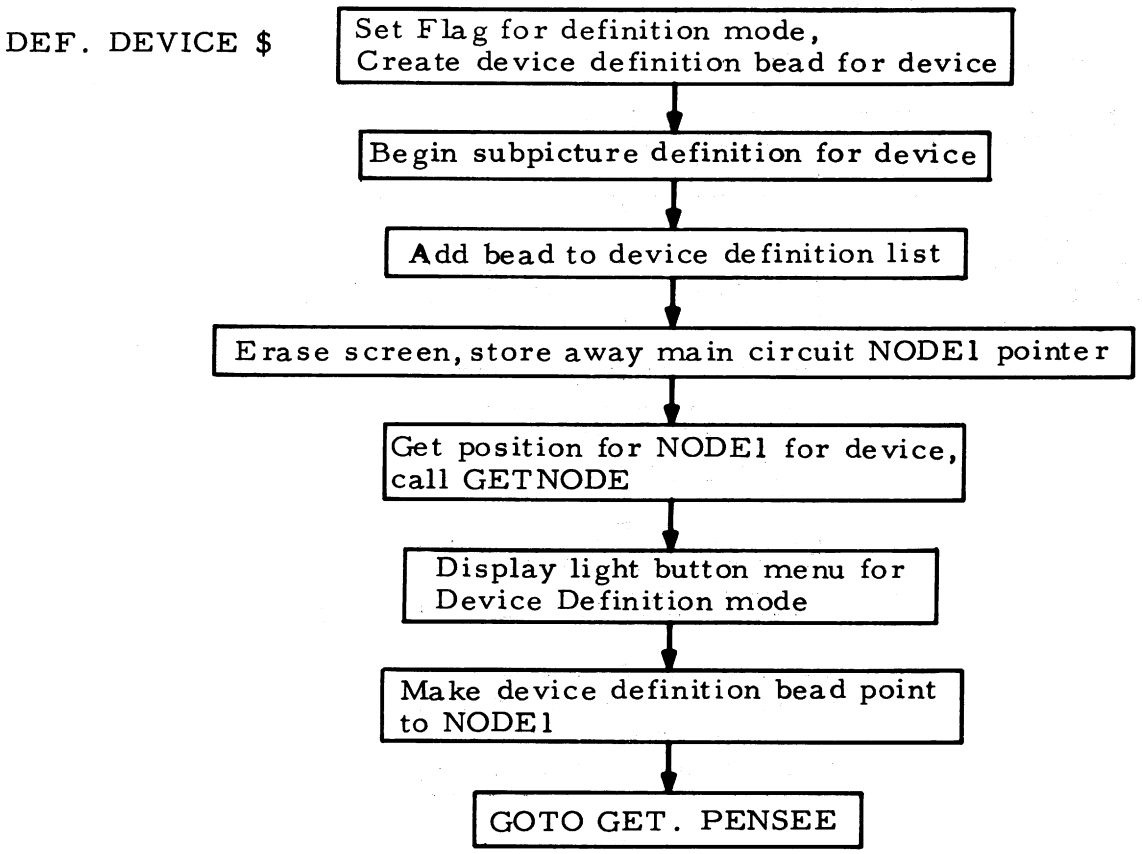


Procedure RMVNODE (NODE)

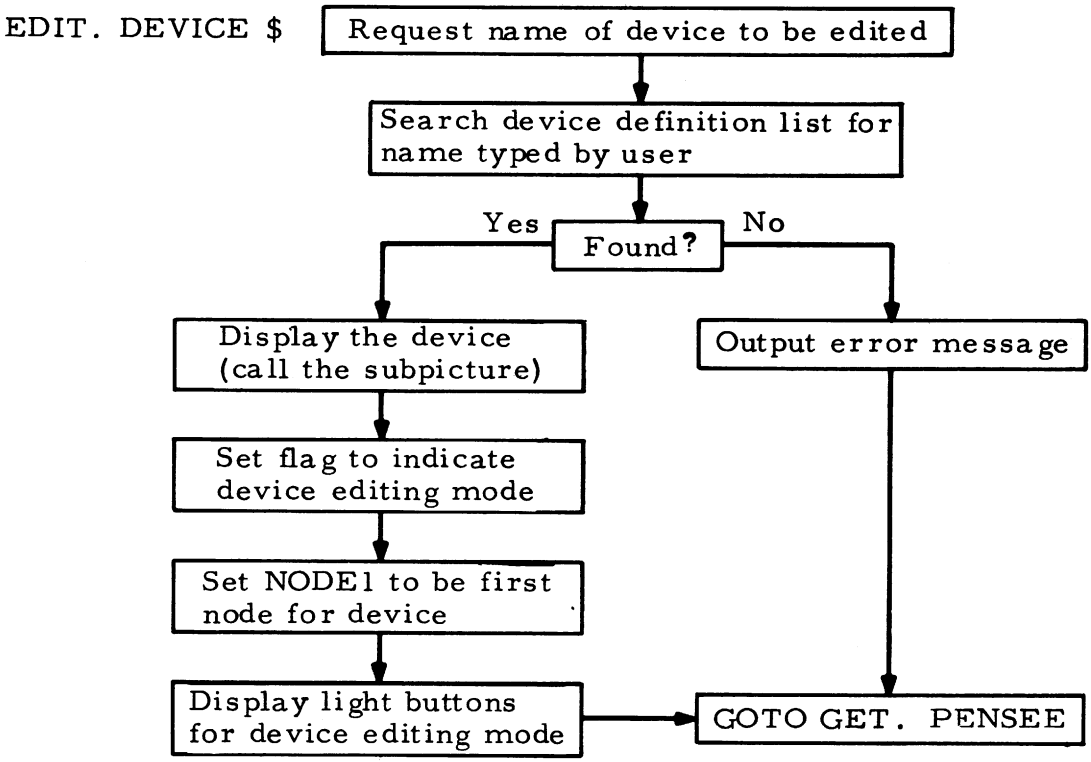
Delete a node from the data structure and display file



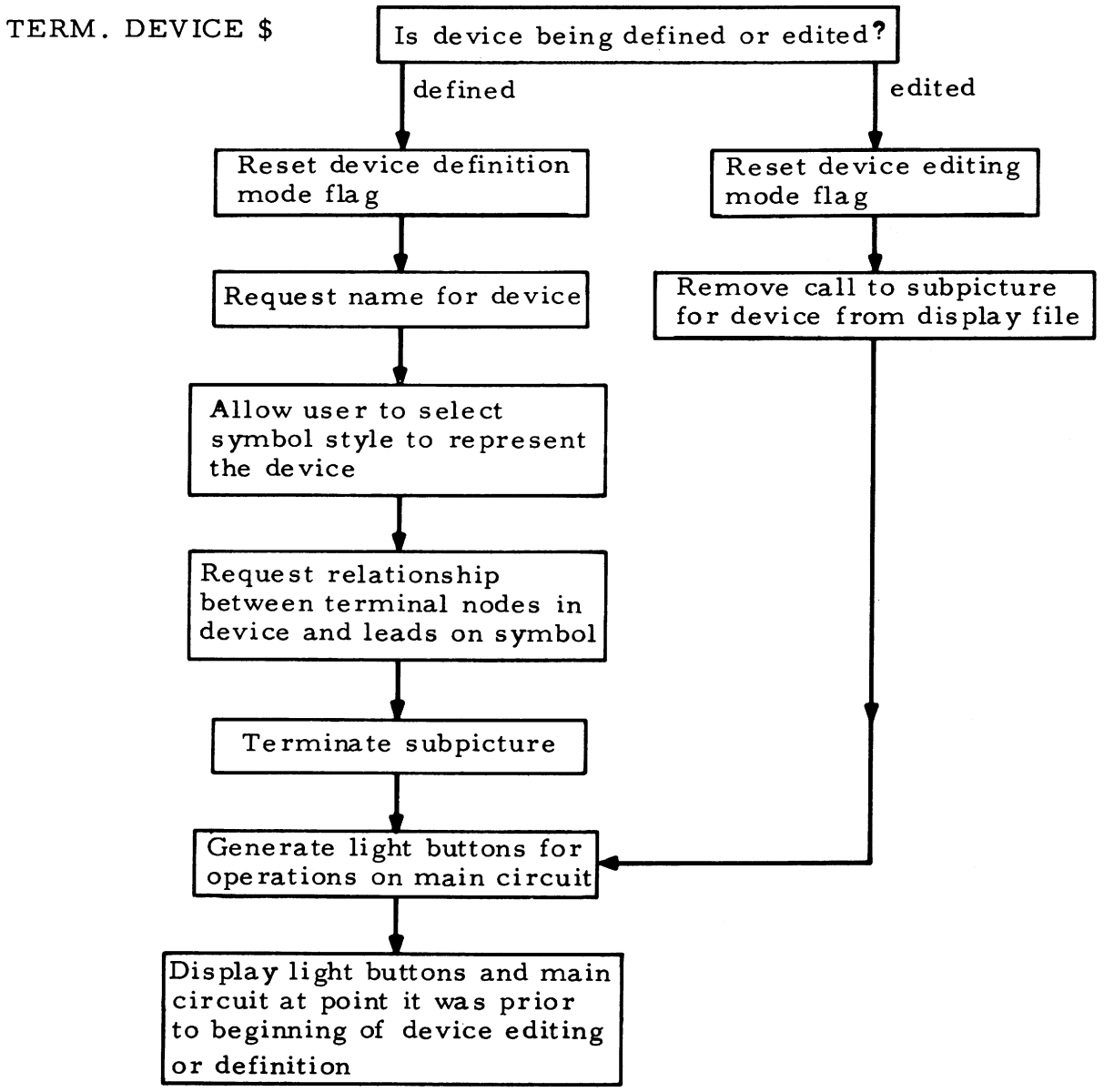
Process Button DEF. DEVICE



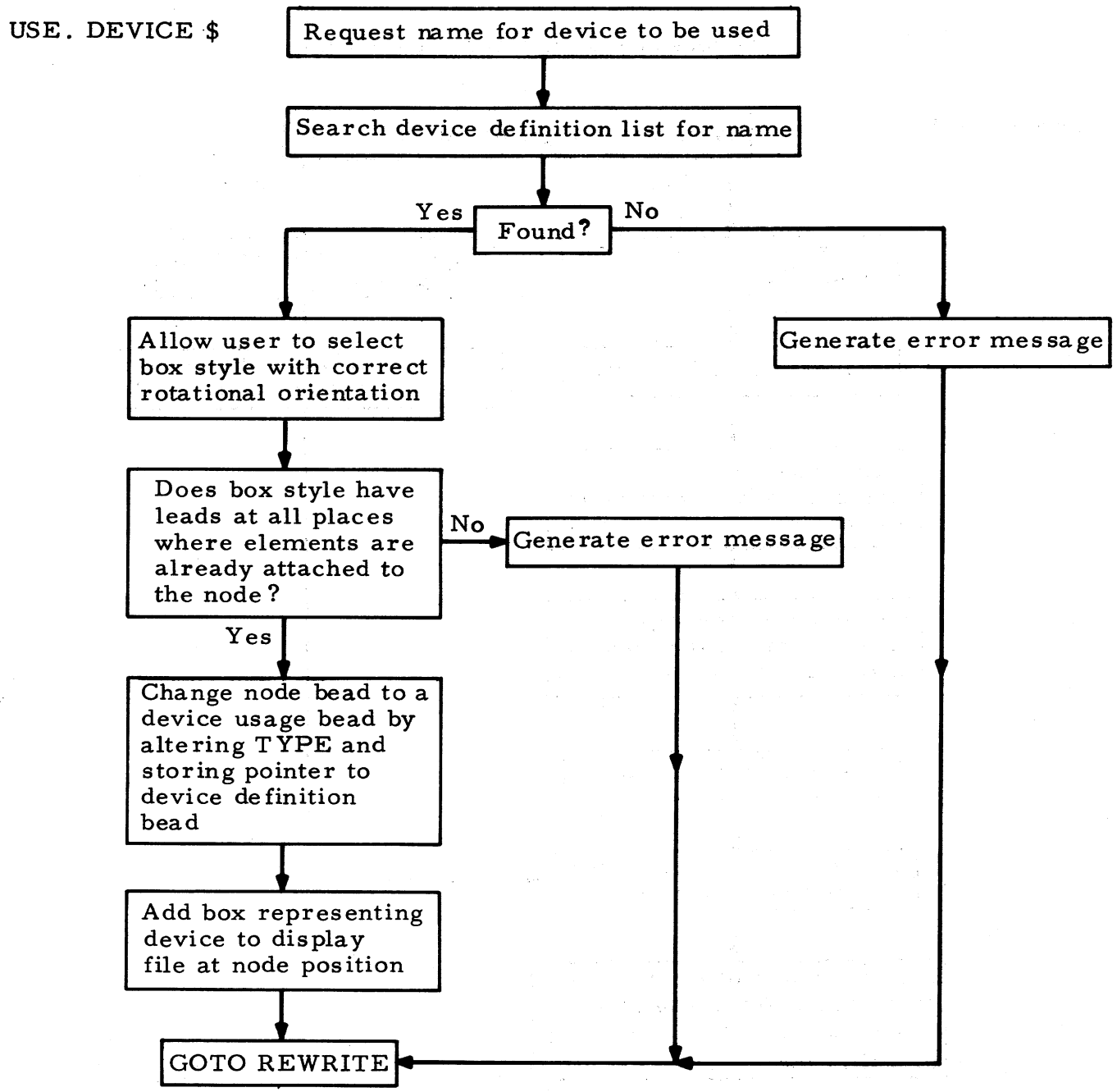
Process Button EDIT. DEVICE



Process Button TERM. DEVICE



Process Button USE. DEVICE



IX. PRINCIPAL LIMITATIONS OF THE SAMPLE PROGRAM

The limitations inherent in the sample program are probably obvious to anyone familiar with circuit design. Several restrictions are basic to the program design:

1. The circuit must be laid out on a fixed rectangular grid, and circuit elements may be attached only between neighboring grid positions. An arbitrary layout of elements is desirable; in the program components may not be arbitrarily located, rotated or scaled on the grid.
2. Circuits larger than those that can be displayed on the screen at one time cannot be constructed; a "windowing" capability is required.
3. The data structure used by the problem cannot use secondary storage, thereby also restricting the size of the circuit that can be designed.
4. The technique for performing correlation assumes that any of the data which might correspond to a pensee is in core.
5. The treatment of devices is very crude in several respects:
    a. Devices should be able to have an arbitrary, user-designed symbol used as their circuit representation.
    b. A device should have an arbitrary number of leads by which it can be connected, not the maximum of four imposed by the grid layout.
    c. The leads of a device should be identified, and the user asked to indicate how each lead should be connected. When the device is being placed

upon a rectangular grid, the present program assumes certain attachments when others are specified. The selection of the proper orientation of an arbitrary symbol for a device, including reflections, translations and rotation, is difficult.

6. The program contains no analysis programs nor does it store more than a trivial amount of numeric data of the type needed by analysis programs.

If all of these restrictions were removed, the program listing would probably triple in size, at least. A more useful system is AEDNET, a large subsystem for analyzing non-linear networks, which is described in references 9 and 10. It was written using AED and GRAPHSYS and has restrictions 1,2,3,5 and 6 removed.

Case Study Bibliography

1. Thornhill, Stotz, Ross, Ward, "An Integrated Hardware-Software System for Computer Graphics in Time-sharing," Project MAC Technical Report 56, Project MAC, Mass. Institute of Technology, Dec. 1968.

2. Stotz, R. H., Cheek, T. B., "A Low-Cost Graphical Display for a Computer Time-Sharing Console," Proc. 8th National Symposium on Information Display (1967), p. 91-97.

3. Ward, J. E., "Graphic Output Performance of the ARDS Terminal," Project MAC Memorandum MAC-M-368, Project MAC, Mass. Institute of Technology, March 1968.

4. Thornhill, D. E., "GRAPHSYS for ARDS," M.I.T. Electronic Systems Laboratory Memorandum 71425-M-318, Mass. Inst. of Technology, Jan. 1969.

5. Johnson, Porter, Ackley, Ross, "Automatic Generation of Efficient Lexical Processors Using Finite State Techniques," Comm. ACM, Vol. 11, No. 12, Dec. 1968, p. 805-813.

6. Ross, D. T., "The AED Approach to Generalized Computer-Aided Design," Proc. ACM 22nd. Nat. Conf., 1967, p. 367-385.

7. Ross, D. T., "The AED Free Storage Package," Comm. ACM, Vol. 10, No. 8, Aug. 1967, p. 481-492.

8. McCracken, D. D., "A Guide to ALGOL Programming," (John Wiley, 1964).

For a discussion of a complete circuit analysis system built using the AED language, see

9. Katzenelson, J., "AEDNET: A Simulator for Nonlinear Networks," Proc. IEEE, Vol. 54, p. 1536-1552, Nov. 1966.

10. Evans, D. S., Katzenelson, J., "Data Structure and Man-Machine Communication for Network Problems," Proc. IEEE, Vol. 55, p. 1135-1144, July, 1967. (An excellent paper to supplement the case study.)

This paper is a summary of the work in:
Evans, D. S., "Man-Machine Communication for the Simulation of Non-linear Circuits," M.S. thesis, M.I.T. Department of Electrical Engineering, 1966 (available from M.I.T. Engineering Library).

LISTING INDEX

Page	Contents
62-64	Declaration statements for first compilation
65	Compile time declarations
66-67	Procedure MAKEPICS
68	Program initialization
69-71	Procedure definitions
	INITBUT (p.67)
	GET.ITEM (p.68-69)
	ERROR (p.69)
	GETNODE (p.70)
	BUTTONS (p.71)
74	Picture maintenance (ERASE AND REDRAW light button); graphic input processing
75	DELETE light button
75-76	Procedure DELETE
76	Procedure RMVNOD
77-78	CHANGE VALUE, HORIZONTAL RESISTOR light buttons
79	VERTICAL RESISTOR, HORIZONTAL and VERTICAL CAPACITOR, HORIZONTAL AND VERTICAL SHORT light buttons. AUTO ERASE ON/OFF light buttons.
80	DEFINE DEVICE light button
81	TERMINATE DEVICE light button
82	USE DEVICE light button
83	EDIT DEVICE light button
84	Procedure FIXEND, NEARNODE, INITARDS
85	Procedure NAMEELEM,NAMENODE
86	Declarations for second compilation

87-94 Procedures BOXES (p.87), CHOICE(p88-90), TRMNLS (p.91), SAMPLE.BOX (p.92), MAKEBOX(p.93-94)

```
      BEGIN

COMMENT   ALL STATEMENTS BEGINNING WITH 'COMMENT' ARE COMMENTS. IN LINE REMARKS MAY ALSO BE INSERTED AT ANY POINT
        IN THE LISTING BY PRECEEDING THEM WITH 3 PERIODS AND FOLLOWING THEM BY 2 SLASHES OR THE USUAL STATEMENT
        DELIMETER, A $ FOLLOWED BY A COMMA.$,

COMMENT DECLARATIONS OF ALL INTEGER VARIABLES, EXCEPT COMPONENTS $,
       INTEGER BUTTON.PUSH,BUT.TYPE,BUT.SIZE,CAPACITOR,CAPSZ,DEFINITION.SIZE,DEVICE.TYPE,END.OF.SUBPIC.DEF,GRIDSP,I,
        MENU1,MENU2,N,NBUT,NEED.ABPOS,NEED.CHARS,NEED.PENSEE,NEED.VALUE,NODE,NODESZ,NOTBUT,PENSEE,RESISTOR,RESSZ,SHORT,
        SHTSZ,XSEEN,YSEEN $,
       INTEGER ARRAY NUMBER.OF.OCCURRENCES(2) $,

COMMENT DECLARATIONS OF BOOLEAN AND REAL VARIABLES $,
       BOOLEAN DEFINITION.MODE,EDITING.DEFINITION,ELEMENT.DELETED,ILLEGAL.ATTACHMENT,NODE.CREATED,REDRAW.MODE $,
       REAL RVAL $,

COMMENT DECLARATION OF ALL POINTER VARIABLES, EXCEPT COMPONENTS $,
       POINTER ADDED.ELEMENT,BEGIN.DEVICE.CHAIN,BOKS1,BOKS2,BOKS3,BOKS4,BOX1,BOX2,BOX3,BOX4,BOX5,BOX6,BOX7,BOX8,BOX9,
        BOX10,BOX11,BOX.STYLE,BT.EDIT,BT.DEFINE,BT.TERM,CREATED.NODE,DEFINITION.POINTER,DEVICE,ELEM.PIC,EMPTY,
        END.OF.DISPLAY.FILE,EX,FBUT,FORBIDDEN.LEAD,FROMNODE,HCAP,HRES,HSHT,INPUT.ITEM,ITEM,LASTDEF,LASTSEE,LBUT,
        MAIN.FILE,NODE1,NODE.PIC,NODEPT,OBSN,P,P1,P2,P3,P4,PREV,PTR,PT.EDIT,PT.DEFINE,PT.TERM,RBOX,STORE.NODE,SUB,
        THIS,TONODE,USAGE,VCAP,VRES,VSHT $,

COMMENT DECLARATION OF ALL NON-VALUED PROCEDURES (SUBROUTINES) WRITTEN AS PART OF CASE STUDY AND DEFINED IN SUBSEQUENT
        PAGES OF THE NOTES. $,
       PROCEDURE BOXES,BUTTONS,DELETE,DELET.ELEMENT,ERROR,FIXEND,GET.ITEM,INITARDS,INITBUT,MAKEPICS,RMVNOD,TRMNLS $,

COMMENT DECLARATIONS OF ALL VALUED PROCEDURES (FUNCTIONS) WRITTEN FOR CASE STUDY. $,
COMMENT  AN INTEGER PROCEDURE HAS AN INTEGER AS ITS VALUE (I.E., IS AN INTEGER FUNCTION) $,
       INTEGER PROCEDURE ISBUTT $,
       POINTER PROCEDURE CHOICE,GETNODE,NAMEELEM,NAMENODE,NEARNODE $,

COMMENT DECLARATION OF ALL PROCEDURES FROM GRAPHSYS LIBRARY $,
       PROCEDURE ADDOBJ,DEFOBJ,DISPLAY,ERASER,PENCHK,PENDLT,SGNON,RMV $,
       POINTER PROCEDURE CALL,CPYOBJ,DEFSUB,DOTTED,ENDOBJ,ENDSUB,INVIS,LASTOBJ,LIN,PENSNS,PLOT,RPL,SETPT,TEXT $,

COMMENT SYSTEM SUPPORT PROCEDURES $,
       PROCEDURE FRET,GOUT,CHNCOM,COMPAR $,
       POINTER PROCEDURE COPYC,FREE,NUMTOC $,

COMMENT FOLLOWING PORTION OF PROGRAM IS DECLARATION OF COMPONENTS FOR ALL BEADS $,

COMMENT COMPONENTS COMMON TO SEVERAL BEADS  $,
       INTEGER COMPONENT TYPE $,
       TYPE $=$ 1                                  ... TYPE ALWAYS IS ONE WORD FROM THE TOP OF A BEAD $,
       POINTER COMPONENT NAME $,
       NAME $=$ 4 $,                               ... NAME ALWAYS IS 4 WORDS FROM THE TOP OF A BEAD $,
```

```
COMMENT DECLARATIONS FOR NODE AND DEVICE USAGE BEADS. THE 2 BEADS ARE THE SAME EXCEPT FOR THE EXTRA COMPONENT IN THE
        DEVICE USAGE BEAD. TYPE = 0 FOR NODE BEAD, TYPE = 4 FOR DEVICE USAGE BEAD. $,
        INTEGER COMPONENT LEADS                          ... NUMBER OF LEADS ATTACHED $,
        LEADS $=$ 2 $,
        POINTER COMPONENT NEXT                           ... POINTER TO NEXT NODE BEAD $,
        NEXT $=$ 3 $,
COMMENT NAME IS 4TH COMPONENT $,
        POINTER COMPONENT U                              ... POINTER TO UP ELEMENT BEAD $,
        U $=$ 5 $,
        POINTER COMPONENT L                              ... POINTER TO LEFT ELEMENT BEAD $,
        L $=$ 6 $,
        POINTER COMPONENT R                              ... POINTER TO RIGHT ELEMENT BEAD $,
        R $=$ 7 $,
        POINTER COMPONENT D                              ... POINTER TO DOWN ELEMENT BEAD $,
        D $=$ 8 $,
        INTEGER COMPONENT X                              ... X COORDINATE ON SCREEN OF NODE OR SYMBOL REPRESENTING DEVICE
                                                          USAGE $,
        X $=$ 9 $,
        INTEGER COMPONENT Y                              ... CORRESPONDING Y COORDINATE $,
        Y $=$ 10 $,
        POINTER COMPONENT DEF.PTR                        ... THIS COMPONENT, WHICH EXISTS ONLY FOR DEVICE USAGE BEADS, IS
                                                          A POINTER TO THE DEVICE DEFINITION BEING USED $,
        DEF.PTR $=$ 11 $,

COMMENT DECLARATIONS FOR RESISTOR, CAPACITOR AND SHORT BEADS, TYPE = 1 FOR RESISTOR, = 2 FOR CAPACITOR, = 3 FOR SHORT. $,
        POINTER COMPONENT FROM                           ... POINTER TO NODE FROM WHICH ELEMENT IS ATTACHED $,
        FROM $=$ 2 $,
        POINTER COMPONENT INTO                           ... POINTER TO NODE TO WHICH ELEMENT IS ATTACHED $,
        INTO $=$ 3 $,

COMMENT RESISTOR AND CAPACITOR BEADS ALSO HAVE NAME AND VALUE COMPONENTS $,
        REAL COMPONENT VAL                               ... VALUE OF RESISTANCE OR CAPACITANCE $,
        VAL $=$ 5                                        ... THIS COMPONENT IN A USEFUL SYSTEM MIGHT BE A POINTER TO A
                                                          PROPERTY LIST CONTAINING A VARIETY OF INFORMATION REQUIRED BY
                                                          ANALYSIS PROGRAMS $,

COMMENT DECLARATIONS FOR DEVICE DEFINITION BEADS. THE FIRST 2 WORDS ARE RESERVED FOR THE GRAPHSYS CORRELATION WORDS.
        IN NODE, RESISTOR, CAPACITOR AND SHORT BEADS, ONLY A SINGLE WORD IS REQUIRED AS THESE OBJECTS DO NOT INVOLVE A
        SUBPICTURE DEFINITION OR CALL $,
        INTEGER COMPONENT STYLE                          ... CODE FOR STYLE OF SYMBOL TO BE USED TO REPRESENT THE DEVICE $,
        STYLE $=$ 2 $,

COMMENT THIRD COMPONENT, NEXT, POINTS TO NEXT DEVICE ON CHAIN IN SAME WAY NEXT POINTS TO NEXT NODE ON NODE CHAIN.
        FOURTH COMPONENT IS DEVICE NAME $,
        POINTER COMPONENT FSTNODE                        ... POINTER TO FIRST NODE IN EQUIVALENT CIRCUIT FOR DEVICE $,
        FSTNODE $=$ 5 $,
        INTEGER COMPONENT NUM                            ... NUMBER OF USAGES OF THE DEVICE. IT IS USED TO CREATE A UNIQUE
                                                          NAME FOR EACH USAGE BY CONCATENATING THE NAME WITH NUM. $,
        NUM $=$ 6 $,
        POINTER COMPONENT TRMNL1                         ... POINTER TO FIRST TERMINAL NODE BEAD IN DATA STRUCTURE FOR
                                                          EQUIVALENT CIRCUIT $,
        TRMNL1 $=$ 7 $,
        POINTER COMPONENT TRMNL2,TRMNL3,TRMNL4           ... POINTERS TO OTHER TERMINAL NODE BEADS $,
        TRMNL2 $=$ 8 $,
        TRMNL3 $=$ 9 $,
        TRMNL4 $=$ 10 $,
```

```
COMMENT DECLARATION OF BEADS RETURNED BY GET.ITEM FOR GRAPHIC INPUT ITEMS $,
        INTEGER COMPONENT XX                          ... X POSITION OF CURSOR $,
        XX $=$ 0 $,
        INTEGER COMPONENT YY                          ... Y POSITION OF CURSOR $,
        YY $=$ 1 $,
        POINTER COMPONENT OBJECT.NAME                 ... POINTER TO CORRELATION WORD, IF A 'PENSEE' WAS DETECTED.
                                                      USER MUST HAVE RECORDED CORRESPONDENCE BETWEEN THIS POINTER AND
                                                      AN OBJECT IN HIS DATA STRUCTURE. IN THIS PROGRAM THE
                                                      CORRESPONDENCE IS IMPLEMENTED BY HAVING THE USER'S DATA BEAD BEGIN
                                                      WITH THE CORRELATION WORD $,
        OBJECT.NAME $=$ 2                             ... IF THE OBJECT RECEIVING A 'PENSEE' WAS IN A SUBPICTURE, THE
                                                      NEXT 2N POINTER COMPONENTS WILL CONTAIN POINTERS TO THE
                                                      CORRELATION WORDS FOR THE N SUBPICTURE CALLS AND THE N
                                                      SUBPICTURE DEFINITIONS, WHERE N IS THE DEPTH OF SUBPICTURE
                                                      NESTING $,

COMMENT COMPONENT DECLARATIONS FOR BEADS CREATED FOR LIGHT BUTTONS $,
COMMENT  LIGHT BUTTON BEAD ALSO HAS TYPE CODE IN THE SECOND WORD, THE
          FIRST WORD IS THE GRAPHSYS CORRELATION WORD $,
        INTEGER COMPONENT NUMBER $,
        NUMBER $=$ 2                                  ... NUMBER ASSIGNED TO BUTTON IS IN THIRD WORD $,
```

```
      PRESET                                        ... INITIALIZE CONSTANT VALUES AT COMPILE TIME //
           BEGIN
           ELEMENT.DELETED = 0B                     ... INDICATES AN ELEMENT WAS DELETED IN ORDER TO PLACE
                                                     A NEW ELEMENT IN ITS POSITION. 0B IS EQUIVALENT TO FALSE. $,
           REDRAW.MODE = 0B                         ... INDICATES PICTURE IS TO BE REDRAWN AT NEXT OPPORTUNITY $,
           DEFINITION.MODE = 0B                     ... INDICATES IF A DEVICE IS BEING DEFINED $,
           EDITING.DEFINITION = 0B                  ... INDICATES IF A DEVICE IS BEING EDITED $,
           END.OF.SUBPIC.DEF = 0                    ... CODE TO PROCEDURE DISPLAY TO INDICATE THE END OF A PARTIALLY
                                                     COMPLETED SUBPICTURE $,
           MAIN.FILE = 0                            ... CODE TO INDICATE THE PROGRAM IS WORKING IN THE MAIN DISPLAY
                                                     FILE, RATHER THAN WITHIN A SUBPICTURE $,
           NOTBUT = 0                               ... CODE TO INDICATE THAT THE MOST RECENT 'PEN SEE' WAS ON AN
                                                     OBJECT OTHER THAN A LIGHT BUTTON $,
           FORBIDDEN.LEAD = -1                      ... CODE TO INDICATE THIS DIRECTION ON A DEVICE HAS NO LEAD
                                                     AVAILABLE $,
           NUMBER.OF.OCCURRENCES = 0,0,0            ... ARRAY USED TO STORE NUMBER OF OCCURRENCES OF NODES,
                                                     RESISTORS, AND CAPACITORS. USED TO GENERATE A UNIQUE NAME FOR
                                                     EACH OCCURRENCE BY CONCATENATION OF N,R OR C WITH THE NUMBER OF
                                                     OCCURRENCES FOR THE CIRCUIT ELEMENT $,
           GRIDSP = 200                             ... DISTANCE BETWEEN GRID POINTS AT WHICH NODES CAN EXIST $,
           NODE1 = 0                                ... CODE TO INDICATE FIRST NODE IS NOT YET DEFINED $,
           BEGIN.DEVICE.CHAIN = 0                   ... CODE TO INDICATE NO DEVICES HAVE BEEN DEFINED $,
           EMPTY = 0                                ... END OF CHAIN CODE -- NULL POINTER $,

COMMENT PRESET TYPE NUMBERS TO INDICATE BEAD TYPE $,
           NODE = 0 $,
           RESISTOR = 1 $,
           CAPACITOR = 2 $,
           SHORT = 3 $,
           DEVICE.TYPE = 4 $,

COMMENT PRESET BEAD SIZES $,
           SHTSZ = 4                                ... SHORT CIRCUT BEAD SIZE $,
           RESSZ = CAPSZ = 6                        ... SIZE FOR CAPACITOR AND RESISTOR BEADS $,
           NODESZ = 12                              ... SIZE FOR NODE AND DEVICE USAGE BEADS $,
           DEFINITION.SIZE = 11                     ... SIZE FOR DEVICE DEFINITION BEADS $,
           MENU1 = 1                                ... CODE TO INDICATE FIRST SET OF LIGHT BUTTONS SHOULD BE USED $,
           MENU2 = 2                                ... CODE FOR SECOND SET OF LIGHT BUTTONS -- USED ONLY WITH DEVICE
                                                     DEFINITIONS $,
           BUT.TYPE = 5                             ... CODE FOR A LIGHT BUTTON BEAD $,
           BUT.SIZE = 3                             ... LENGTH OF A LIGHT BUTTON BEAD $,
           NEED.PENSEE = 1                          ... CODE TO INDICATE INPUT ITEM MUST BE A PENSEE $,
           NEED.ABPOS = 2                           ... CODE TO INDICATE CURSOR ABSOLUTE COORDINATES REQUIRED $,
           NEED.CHARS = 3                           ... CODE TO INDICATE CHARACTER STRING REQUIRED $,
           NEED.VALUE = 4                           ... CODE TO INDICATE REAL NUMERIC VALUE REQUIRED $,
           END $,

      SWITCH LIGHT.BUTTON = DELET,NEWVAL,HORES,VERES,HOCAP,VECAP,HOSHT,VESHT,ERSON,ERSOFF,REWRITE,EXIT,USE.DEVICE,
       TERM.DEVICE,DEF.DEVICE,EDIT.DEVICE           ... DECLARE ARRAY OF STATEMENT LABELS CORRESPONDING TO PRINCIPAL
                                                     FUNCTIONS OF THE PROGRAM.  'SWITCH' DECLARES LIGHT.BUTTON TO BE A
                                                     A STATEMENT LABEL ARRAY OF 16 ELEMENTS $,
```

```
COMMENT DEFINITION OF PROCEDURE MAKEPICS. PROCEDURES CAN BE DEFINED AT ANY POINT IN AN AED PROGRAM, BUT THEY WILL NOT
        BE EXECUTED UNTIL THEY ARE CALLED. MAKEPICS IS USED IN THE INITIALIZATION PHASE OF THE PROGRAM TO BUILD THE
        PICTORIAL REPRESENTATION OF NODES, RESISTORS, CAPACITORS, AND SHORTS $,

      DEFINE PROCEDURE MAKEPICS(EX,NODEPT,HRES,VRES,HCAP,VCAP,VSHT,HSHT) WHERE POINTER EX,NODEPT,HRES,VRES,HCAP,VCAP,
       VSHT,HSHT TOBE
                                                  ... MAKEPICS DEFINES THE COMPOUND DISPLAY OBJECTS REQUIRED TO
                                                   PLOT ALL THE ELEMENTS TO BE DISPLAYED //
       BEGIN                                      ... FIRST DISPLAY OBJECT IS AN X TO BE DISPLAYED OVER DELETED
                                                   ELEMENTS //
       DEFOBJ()                                   ... BEGIN OBJECT DEFINITION. DEFOBJ HAS NO ARGUMENTS BUT () INDICATES
                                                     THAT IT IS A PROCEDURE CALL. $,
      ADDOBJ(INVIS(LIN(-40,40)))                  ... INVISIBLE LINE WITH X INCREMENT = -40,Y INCREMENT = 40 $,
      ADDOBJ(DOTTED(LIN(80,-80)))                 ... DOTTED LINE WITH X INCREMENT = 80, Y INCREMENT = -80 $,
      ADDOBJ(INVIS(LIN(-80,0))) $,
      ADDOBJ(DOTTED(LIN(80,80))) $,
      ADDOBJ(INVIS(LIN(-40,-40))) $,
      EX = ENDOBJ()                               ... END DISPLAY OBJECT DEFINITION,NAME IS EX $,
      DEFOBJ() $,
      ADDOBJ(INVIS(LIN(5,5))) $,
      ADDOBJ(LIN(0,-10))                          ... VISIBLE LINE WITH X INCREMENT = 0, Y INCREMENT = -10 $,
      ADDOBJ(LIN(-10,0)) $,
      ADDOBJ(LIN(0,10)) $,
      ADDOBJ(LIN(10,0)) $,
      ADDOBJ(INVIS(LIN(-7,20))) $,
      NODEPT = ENDOBJ()                           ... DISPLAY OBJECT FOR A NODE,NAME = NODEPT $,
      DEFOBJ() $,
      ADDOBJ(LIN(60,0)) $,
      ADDOBJ(LIN(10,10)) $,
      ADDOBJ(LIN(20,-20)) $,
      ADDOBJ(LIN(20,20)) $,
      ADDOBJ(LIN(20,-20)) $,
      ADDOBJ(LIN(10,10)) $,
      ADDOBJ(LIN(60,0)) $,
      ADDOBJ(INVIS(LIN(-100,40))) $,
      HRES = ENDOBJ()                             ... DISPLAY OBJECT FOR A HORIZONTAL RESISTOR,NAME = HRES $,
      DEFOBJ() $,
      ADDOBJ(LIN(0,-60)) $,
      ADDOBJ(LIN(10,-10)) $,
      ADDOBJ(LIN(-20,-20)) $,
      ADDOBJ(LIN(20,-20)) $,
      ADDOBJ(LIN(-20,-20)) $,
      ADDOBJ(LIN(10,-10)) $,
      ADDOBJ(LIN(0,-60)) $,
      ADDOBJ(INVIS(LIN(40,100))) $,
      VRES = ENDOBJ()                             ... DISPLAY OBJECT FOR A VERTICAL RESISTOR,NAME = VRES $,
      DEFOBJ() $,
      ADDOBJ(LIN(90,0)) $,
      ADDOBJ(INVIS(LIN(0,20))) $,
      ADDOBJ(LIN(0,-40)) $,
      ADDOBJ(INVIS(LIN(20,0))) $,
      ADDOBJ(LIN(0,40)) $,
      ADDOBJ(INVIS(LIN(0,-20))) $,
      ADDOBJ(LIN(90,0)) $,
      ADDOBJ(INVIS(LIN(-100,40))) $,
      HCAP = ENDOBJ()                             ... DISPLAY OBJECT FOR A HORIZONTAL CAPACITOR,NAME IS HCAP $,
      DEFOBJ() $,
      ADDOBJ(LIN(0,-90)) $,
      ADDOBJ(INVIS(LIN(20,0))) $,
      ADDOBJ(LIN(-40,0)) $,
      ADDOBJ(INVIS(LIN(0,-20))) $,
```

```
        ADDOBJ(LIN(40,0)) $,
        ADDOBJ(INVIS(LIN(-20,0))) $,
        ADDOBJ(LIN(0,-90)) $,
        ADDOBJ(INVIS(LIN(40,100))) $,
        VCAP = ENDOBJ()                          ... DISPLAY OBJECT FOR A VERTICAL CAPACITOR, NAME=VCAP $,
        HSHT = LIN(200,0)                        ... DISPLAY OBJECT FOR HORIZONTAL SHORT ,NAME = HSHT $,
        VSHT = LIN(0,-200)                       ... DISPLAY OBJECT FOR A VERTICAL SHORT,NAME = VSHT $,
        END $,

COMMENT  END TERMINATES THE DEFINITION OF PROCEDURE MAKEPICS. $,
```

```
COMMENT THE EXECUTABLE PART OF THE PROGRAM BEGINS HERE. THE FOLLOWING SECTION OF THE PROGRAM PERFORMS A VARIETY OF
        INITIALIZATION TASKS $,
       INITARDS()                                          ... INITIALIZE GRAPHSYS AND ARDS TO RECEIVE GRAPHIC INPUT $,
       MAKEPICS(EX,NODEPT,HRES,VRES,HCAP,VCAP,VSHT,HSHT) ... CALL PROCEDURE MAKEPICS, WHICH IS DEFINED ABOVE, TO MAKE THE
                                                           DISPLAY OBJECTS REPRESENTING ELEMENTS AND NODES. ARGUMENTS ARE
                                                           POINTERS TO THE DISPLAY OBJECTS GENERATED. THESE OBJECTS CAN
                                                           LATER BE ADDED TO THE DISPLAY FILE USING PROCEDURE PLOT $,
       INITBUT(FBUT,LBUT)                                 ... CALL TO INITBUT, WHICH IS DEFINED BELOW, TO GENERATE DISPLAY
                                                           OBJECTS FOR ALL LIGHT BUTTONS TO BE USED ON BOTH LIGHT BUTTON
                                                           MENUS. POINTER TO THE CORRELATION WORD FOR THE FIRST BUTTON IS
                                                           FBUT AND FOR THE LAST BUTTON IS LBUT. LBUT IS NOW THE LAST ITEM
                                                           IN THE DISPLAY FILE . $,
       END.OF.DISPLAY.FILE = LBUT                         ... USED TO RECORD WHAT IS THE LAST OBJECT IN THE DISPLAY FILE SO
                                                            OBJECTS CAN ALWAYS BE ADDED AT THE END OF THE DISPLAY
                                                           FILE RATHER THAN WITHIN IT. $,
       GET.ITEM(.C. /INDICATE POSITION OF INITIAL NODE OF MAIN CIRCUIT/,INPUT.ITEM,NEED.ABPOS) ... CALL PROCEDURE
                                                           GET.ITEM, WHICH IS DEFINED BELOW, TO GET POSITION WHERE FIRST
                                                           NODE IS TO BE DRAWN BY READING THE CURSOR COORDINATES.
                                                           INPUT.ITEM POINTS TO A BEAD WHICH WILL RECEIVE THE X,Y CURSOR
                                                           COORDINATES $,
       LASTSEE = NODE1 = GETNODE(XX(INPUT.ITEM),YY(INPUT.ITEM)) ... GETNODE FINDS THE NEAREST GRID POSITION TO THE
                                                           CURSOR AND RETURNS A POINTER TO THE NODE BEAD BUILT $,
       BUTTONS(MENU1)                                     ... SETUP FIRST SET OF LIGHT BUTTONS BY REMOVING FROM THE DISPLAY
                                                           FILE THOSE BUTTONS THAT ARE RELEVANT ONLY WHEN DEVICES ARE BEING
                                                           DEFINED. $,
       DISPLAY()                                          ... DISPLAY THE LIGHT BUTTONS AND THE FIRST NODE. $,

COMMENT  PROCEDURE INITBUT,GET.ITEM,GETNODE, AND BUTTONS USED ABOVE ARE DEFINED IN THE NEXT SECTION OF THE
         LISTING.  FOLLOWING THE PROCEDURE DEFINITIONS, THE MAIN PROGRAM IS CONTINUED. $,
       GOTO GET.PENSEE                                    ... TRANSFER TO PORTION OF PROGRAM HANDLING GRAPHIC INPUT. $,
```

```
DEFINE PROCEDURE INITBUT(FBUT,LBUT) WHERE POINTER FBUT,LBUT TOBE
     BEGIN                                          ... INITBUT BUILDS THE LIGHT BUTTON DATA STRUCTURE AND ADDS THE
                                                     LIGHT BUTTONS TO THE DISPLAY FILE //

     DEFINE POINTER PROCEDURE MAKEBUT(NAME,NBUT) WHERE INTEGER NBUT $,
     POINTER NAME TOBE
                                                    ... MAKEBUT CREATES A BEAD FOR A LIGHT BUTTON AND ADDS THE TEXT FOR
                                                    IT TO THE DISPLAY FILE. MAKEBUT IS AN INTERNAL PROCEDURE OF INITBUT //
          BEGIN
          POINTER P $,
          P = FREE(BUT.SIZE)                        ... GET BUTTON BEAD FROM FREE STORAGE $,
          TYPE(P) = BUT.TYPE                        ... PUT BUTTON TYPE INTO BEAD $,
          NUMBER(P) = NBUT                          ... BUTTON NUMBER TO BE RETURNED AS CODE WHEN A PENSEE OCCURS ON
                                                     THE BUTTON $,
          MAKEBUT = PENSNS(PLOT(TEXT(NAME),P))      ... ADD TEXT OF LIGHT BUTTON TO DISPLAY FILE AS PEN SENSITIVE
                                                     WITH NAME P, THE POINTER TO THE CORRELATION WORD FOR THE BUTTON
                                                     AND ALSO THE POINTER TO THE BUTTON BEAD. $,

          END $,

     FBUT = PLOT(SETPT(250,500))                    ...  POINT  AT X=250, Y=500 WHERE BUTTON TEXT WILL BEGIN $,
     MAKEBUT(.C. /DELETE/,1)                        ... WRITE BUTTON HAVING TEXT 'DELETE' AND A BUTTON CODE OF 1 $,
     PLOT(SETPT(250,440)) $,
     MAKEBUT(.C. /CHANGE VALUE/,2) $,
     PLOT(SETPT(250,380)) $,
     MAKEBUT(.C. /HORIZ RESISTOR/,3) $,
     PLOT(SETPT(250,320)) $,
     MAKEBUT(.C. /VERT RESISTOR/,4) $,
     PLOT(SETPT(250,260)) $,
     MAKEBUT(.C. /HORIZ CAPACITOR/,5) $,
     PLOT(SETPT(250,200)) $,
     MAKEBUT(.C. /VERT CAPACITOR/,6) $,
     PLOT(SETPT(250,140)) $,
     MAKEBUT(.C. /HORIZ SHORT/,7) $,
     PLOT(SETPT(250,80)) $,
     MAKEBUT(.C. /VERT SHORT/,8) $,
     PLOT(SETPT(250,20)) $,
     MAKEBUT(.C. /AUTO ERASE ON/,9) $,
     PLOT(SETPT(250,-40)) $,
     MAKEBUT(.C. /AUTO ERASE OFF/,10) $,
     PLOT(SETPT(250,-100)) $,
     MAKEBUT(.C. /ERASE AND REDRAW/,11) $,
     PLOT(SETPT(250,-160)) $,
     MAKEBUT(.C. /QUIT/,12) $,
     PLOT(SETPT(250,-220)) $,
     MAKEBUT(.C. /ATTACH DEVICE/,13) $,
     PT.TERM = PLOT(SETPT(250,-280)) $,
     BT.TERM = MAKEBUT(.C. /TERMINATE/,14) $,
     PT.DEFINE = PLOT(SETPT(250,-340)) $,
     BT.DEFINE = MAKEBUT(.C. /DEFINE DEVICE/,15) $,
     PT.EDIT = PLOT(SETPT(250,-400)) $,
     BT.EDIT = MAKEBUT(.C. /EDIT DEVICE/,16) $,
     LBUT = PLOT(SETPT(250,-460))                   ... ALL BUTTONS ARE IN THE DISPLAY FILE BETWEEN OBJECTS FBUT AND
                                                     LBUT $,

     END $,
```

```
DEFINE PROCEDURE GET.ITEM(MSG,ITEM,CODE) WHERE POINTER MSG,ITEM $,
INTEGER CODE TOBE
                                              ... GET.ITEM WILL PRINT THE MESSAGE POINTED TO BY MSG AND WILL
                                               LOOK FOR AN INPUT ITEM OF THE TYPE GIVEN BY CODE. IF ONE IS
                                               FOUND, ITEM WILL BE THE POINTER TO IT UNLESS IT IS A NUMERIC
                                               VALUE, IN WHICH CASE ITEM WILL CONTAIN THE VALUE. THE ACTUAL
                                               INPUT IS HANDLED BY THE GRAPHSYS SYSTEM ROUTINE GETITM. //
 BEGIN
BOOLEAN FIRST.TIME $,
POINTER RBUFF $,
INTEGER INPUT.TYPE,STRING,REAL.VAL,ABPOS,PENSEE,BUTTON.PUSH $,
POINTER PROCEDURE ITMBUF $,
PROCEDURE ERROR,ITMERR $,
INTEGER PROCEDURE GETITM $,
PRESET
     BEGIN                                    ... PRESET CODES FOR ALLOWABLE INPUT ITEMS //
      FIRST.TIME = TRUE $,
     REAL.VAL = 2 $,
     STRING = 3 $,
     BUTTON.PUSH = 64 $,
     ABPOS = 65 $,
     PENSEE = 67 $,
     END $,
IF FIRST.TIME
THEN BEGIN
     RBUFF = ITMBUF(10,5) $,                 ... ITMBUF OBTAINS AN INPUT BUFFER FROM FREE STORAGE OF A SIZE
                                               SUFFICIENT TO HANDLE ITEMS OF UP TO 10 MACHINE WORDS AND PENSEES
                                               INVOLVING SUBPICTURES TO A DEPTH OF 5 CALLS. RBUFF IS A POINTER
                                               TO THE BUFFER //
      FIRST.TIME = FALSE $,
     END $,
START $   IF CODE EQL NEED.PENSEE             ... CALLING PROGRAM WANTS A PENSEE //
     THEN BEGIN
     INPUT.TYPE = GETITM(MSG,ITEM,RBUFF,ITMERR) $, ... CALL THE GRAPHSYS INPUT ROUTINE TO PRINT THE
                                               MESSAGE AND TO READ THE NEXT INPUT ITEM. ITMERR IS A PROCEDURE
                                               CALLED IF AN ILLEGAL ITEM IS FOUND. THE USER CAN PROVIDE HIS OWN
                                               VERSION OR USE, AS THIS PROGRAM DOES, A STANDARD SYSTEM VERSION.
                                               ITMERR COULD PERFORM ANY DESIRED ACTION, INCLUDING CALLING AN
                                               EDITOR TO EDIT THE BUFFER CONTENTS. INPUT.TYPE WILL BE SET TO
                                               CODE RETURNED BY GETITM TO SPECIFY WHAT IT FOUND //
      IF INPUT.TYPE EQL BUTTON.PUSH          ... ONE OF THE BUTTONS ON THE GRAPHIC INPUT DEVICE WAS PUSHED,
                                               INDICATING A GRAPHIC INPUT ITEM IS TO FOLLOW //
     THEN BEGIN
          INPUT.TYPE = GETITM(0,ITEM,RBUFF,ITMERR)   ... 0 AS FIRST ARGUMENT MEANS NO MESSAGE IS TO BE
                                               PRINTED TO PROMPT USER //
           IF INPUT.TYPE EQL PENSEE
          THEN GOTO RETURN $,                ... EXIT FROM PROCEDURE //
           END $,
     GOTO BAD.INPUT $,                       ... EXPECTED TYPE OF INPUT WAS NOT SUPPLIED //
      END $,
IF CODE EQL NEED.ABPOS                       ... CALLING PROGRAM WANTS A CURSOR POSITION //
THEN BEGIN
     INPUT.TYPE = GETITM(MSG,ITEM,RBUFF,ITMERR) $,
     IF INPUT.TYPE EQL BUTTON.PUSH
     THEN BEGIN
          INPUT.TYPE = GETITM(0,ITEM,RBUFF,ITMERR) $,
          IF INPUT.TYPE EQL ABPOS
          THEN GOTO RETURN $,
          END $,
     GOTO BAD.INPUT $,
     END $,
```

```
            IF CODE EQL NEED.CHARS                    ... CALLING PROGRAM WANTS A CHARACTER STRING //
            THEN BEGIN
                 INPUT.TYPE = GETITM(MSG,ITEM,RBUFF,ITMERR) $,
                 IF INPUT.TYPE EQL STRING
                 THEN GOTO RETURN
                 ELSE GOTO BAD.INPUT $,
                 END $,
            IF CODE EQL NEED.VALUE                    ... CALLING PROGRAM WANTS A NUMERIC VALUE //
            THEN BEGIN
                 INPUT.TYPE = GETITM(MSG,ITEM,RBUFF,ITMERR) $,
                 IF INPUT.TYPE EQL REAL.VAL
                 THEN GOTO RETURN
                 ELSE GOTO BAD.INPUT $,
                 END
            ELSE BEGIN                                ... PROCEDURE DOES NOT RECOGNIZE CODE PROVIDED BY CALLING
                                                       PROGRAM. SET OUTPUT ITEM POINTER TO BE EMPTY //
                  ITEM = 0 $,
                 GOTO RETURN $,
                 END $,
BAD.INPUT $ ERROR(4) $,                               ... GIVE THE USER AN ERROR MESSAGE AND ALLOW HIM TO TRY AGAIN.
                                                       THIS SECTION SHOULD BE EXPANDED IN A USEFUL SYSTEM TO GIVE THE
                                                       USER A BETTER IDEA OF WHAT THE PROGRAM EXPECTS AND WHAT HE DID
                                                       WRONG. //
             GOTO START $,
            END $,

       DEFINE PROCEDURE ERROR(PAR) WHERE INTEGER PAR TOBE
            BEGIN
            POINTER MSG $,
            PROCEDURE GOUT $,
            IF PAR EQL 1
            THEN MSG = .C. /ATTEMPT MADE TO ATTACH ELEMENT TO NONEXISTENT LEAD OF A DEVICE...REQUEST IGNORED/ $,
            IF PAR EQL 2
            THEN MSG = .C. /REFERENCE MADE TO UNDEFINED DEVICE... REQUEST IGNORED/ $,
            IF PAR EQL 3
            THEN MSG = .C. /NO DEVICE ORIENTATION SPECIFIED...TRY AGAIN/ $,
            IF PAR EQL 4
            THEN MSG = .C. /BAD INPUT TYPE...TRY AGAIN/ $,
            GOUT(MSG) $,
            GOTO RETURN $,
            END $,
```

```
  COMMENT DEFINITION OF PROCEDURE GETNODE, WHICH IS USED ABOVE $,

        DEFINE POINTER PROCEDURE GETNODE(XXX,YYY) WHERE INTEGER XXX,YYY TOBE
            BEGIN

  COMMENT GETNODE RETURNS A POINTER TO NODE OR DEVICE USAGE BEAD AT (XXX,YYY) , CREATING A NODE IF NECESSARY. A DEVICE USAGE
          BEAD IS CREATED ONLY BY REPLACING A PREVIOUSLY DEFINED NODE $,
              P = PREV = NODE1                          ... START SEARCHING NODE CHAIN AT FIRST NODE, NODE1. PREV RECORDS
                                                         PREVIOUS NODE ON CHAIN $,
AGAIN $       IF P EQL EMPTY                            ... IS THIS END OF CHAIN. IF SO, NODE DOES NOT EXIST AT THIS GRID
                                                         COORDINATE, SO MUST CREATE ON THERE //

              THEN GOTO GET $,
              IF X(P) EQL XXX AND Y(P) EQL YYY          ... SEE IF THERE IS ALREADY A NODE LOCATED AT THE GRID POSITION
                                                         WHERE NODE IS NEEDED //

              THEN BEGIN
                   GETNODE = P                          ... IF NODE OR DEVICE IS FOUND, GETNODE HAS AS ITS VALUE A
                                                         POINTER TO THE NODE BEAD OR DEVICE USAGE BEAD $,

                   NODE.CREATED = FALSE                 ... SET FLAG TO INDICATE BEAD WAS FOUND ON NODE/DEVICE CHAIN. $,
                   GOTO RETURN $,
                   END $,
              PREV = P                                  ... MOVE TO NEXT BEAD ON CHAIN, UPDATE P AND PREV $,
              P = NEXT(PREV) $,
              GOTO AGAIN $,
GET $         GETNODE = P = FREE(NODESZ)                ... GET A NODE BEAD FROM FREE STORAGE $,
              NODE.CREATED = TRUE $,
              TYPE(P) = NODE                            ... SET TYPE COMPONENT TO VALUE FOR NODE $,
              NEXT(PREV) = P                            ... PUT NEW NODE AT END OF NODE CHAIN. $,
              NEXT(P) = EMPTY $,
              X(P) = XXX                                ... STORE AWAY NODE COORDINATES $,
              Y(P) = YYY $,
              U(P) = D(P) = L(P) = R(P) = EMPTY         ... INDICATE NEW NODE HAS NO ATTACHMENTS $,
              LEADS(P) = 0                              ... INITIALIZE NUMBER OF LEADS $,
              NAME(P) = NAMENODE()                      ... GET A NAME FOR THE NODE $,
              DEFOBJ()                                  ... DEFINE A DISPLAY OBJECT FOR THE NODE, USING THE NODE SYMBOL
                                                         PREVIOUSLY DEFINED AS NODEPT, AND HAVING THE USER'S NAME, WHICH
                                                         WAS JUST CREATED $,

              ADDOBJ(SETPT(XXX,YYY)) $,
              ADDOBJ(CPYOBJ(NODEPT)) $,
              ADDOBJ(TEXT(NAME(P))) $,
              NODE.PIC = ENDOBJ() $,
              END.OF.DISPLAY.FILE = CREATED.NODE = PENSNS(PLOT(NODE.PIC,P,END.OF.DISPLAY.FILE)) ... PLOT THE OBJECT
                                                         REPRESENTING THE NODE AS PEN SENSITIVE AT THE END OF THE DISPLAY
                                                         FILE, WITH NAME P $,
              END $,
```

```
DEFINE PROCEDURE BUTTONS(MENU) WHERE INTEGER MENU TOBE
     BEGIN                                          ... THIS PROCEDURE MODIFIES THE GRAPHSYS DATA STRUCTURE TO ADD OR
                                                     DELETE LIGHT BUTTONS DEPENDING UPON WHAT THE USER IS DOING //

      BOOLEAN FIRST.TIME $,
     PRESET FIRST.TIME = TRUE $,
COMMENT  INITIALIZE VARIABLE TO INDICATE THIS IS THE FIRST TIME THE PROCEDURE WAS CALLED SO THAT THE
      INITIAL MENU, WHICH IS CORRECT, WILL NOT BE ALTERED. $,
     IF MENU EQL MENU1                              ... SET OF LIGHT BUTTONS WHEN THE USER IS WORKING AT THE MAIN
                                                     DISPLAY FILE LEVEL //

     THEN BEGIN
          RMV(BT.TERM)                              ... REMOVE TERMINATE BUTTON, AS OPTION IS MEANINGFUL ONLY WHEN A
                                                     DEVICE IS BEING DEFINED $,

          IF FIRST.TIME
          THEN BEGIN
               FIRST.TIME = FALSE $,
               GOTO RETURN $,
               END $,
          PENSNS(PLOT(TEXT(.C. /DEFINE DEVICE/),BT.DEFINE,PT.DEFINE)) ... ADD TO DISPLAY FILE THE TEXT FOR THE
                                                     BUTTON REMOVED WHEN USER ENTERED DEVICE DEFINITION MODE. AT THAT
                                                     TIME THESE OPTIONS WERE NO LONGER RELEVANT. TEXT MUST BE PLOTTED
                                                     IN THE DISPLAY FILE AFTER THE OBJECT WITH NAME PT.DEFINE, THE
                                                     SET POINT FOR THE BUTTON. THE BUTTON HAS THE NAME BT.DEFINE $,
          PENSNS(PLOT(TEXT(.C. /EDIT DEVICE/),BT.EDIT,PT.EDIT)) $,
          GOTO RETURN $,
          END $,
     IF MENU EQL MENU2                              ... SET OF LIGHT BUTTONS USED WHEN DEFINING OR EDITING A DEVICE. //
     THEN BEGIN
          PENSNS(PLOT(TEXT(.C. /TERMINATE/),BT.TERM,PT.TERM)) $,
          RMV(BT.DEFINE)                            ... THESE BUTTONS ARE NOT RELEVANT AT THIS TIME AS DEVICES CAN
                                                     NOT BE DEFINED WHILE EDITING OR DEFINING A DEVICE $,

          RMV(BT.EDIT) $,
          END $,
     END $,
```

```
  COMMENT MAIN EXECUTABLE PORTION OF PROGRAM FOLLOWS. THE FOLLOWING SECTION IS THE BASIC LOOP OF THE PROGRAM WHICH
          ALTERS THE SCREEN DISPLAY AND WHICH RECIEVES GRAPHICAL INPUT $,

REDRAW $ IF REDRAW.MODE                                    ... REGENERATE PICTURE IF NECESSARY //
         THEN BEGIN
REWRITE $     ERASER()                                     ... ERASE SCREEN, LIGHT BUTTON TO ERASE AND REDRAW TRANSFERS HERE. $,
              IF EDITING.DEFINITION OR DEFINITION.MODE     ... IF EDITING OR DEFINING A DEVICE, SUBPICTURE DEFINITION IS
                                                           BEING ALTERED AND ONLY REMAINDER OF DEFINITION SHOULD BE
                                                           DISPLAYED, NOT THE ENTIRE DISPLAY FILE //
              THEN BEGIN
                   IF EDITING.DEFINITION
                   THEN DISPLAY(DEVICE)                    ... PREVIOUSLY COMPLETED DEFINITION IS BEING EDITED, SO CALL THE
                                                           SUBPICTURE FOR THE DEVICE. DEVICE IS THE NAME OF A SUBPICTURE
                                                           CALL //
                   ELSE DISPLAY(DEVICE,END.OF.SUBPIC.DEF)  ... SUBPICTURE HAS NOT BEEN TERMINATED IF IN DEFINITION MODE
                                                           SO DEVICE IS THE  NAME OF THE FIRST OBJECT IN THE
                                                           SUBPICTURE DEFINITION. THE DISPLAY FILE BETWEEN THIS OBJECT AND
                                                           THE CURRENT END OF THE SUBPICTURE DEFINITION WILL BE DISPLAYED. $,
                   DISPLAY(FBUT,LBUT)                      ... ALL THE LIGHT BUTTONS IN THE DISPLAY FILE BETWEEN THE OBJECT
                                                           WITH NAME FBUT AND THE OBJECT WITH NAME LBUT WILL BE DISPLAYED. $,
                   END
              ELSE DISPLAY()                               ... DISPLAY WITH NO ARGUMENTS WILL CAUSE THE WHOLE DISPLAY FILE
                                                           TO BE OUTPUT $,
              END $,

GET.PENSEE $ GET.ITEM(0,INPUT.ITEM,NEED.PENSEE)            ... THE FOLLOWING LOOP  CONTINUOUSLY
                                                           ANALYZES AND PROCESSES USER INPUT $,
         NBUT = ISBUTT(INPUT.ITEM)                         ... ISBUTT DETERMINES IF A LIGHT BUTTON HAS BEEN POINTED TO WITH
                                                           THE GRAPHIC INPUT DEVICE. IF IT HAS, AN INTEGER 'BUTTON NUMBER'
                                                           IS RETURNED. IF NOT, A CODE OF 0 IS RETURNED TO INDICATE SOME
                                                           OTHER TYPE OF OBJECT WAS SEEN.  ISBUTT IS DEFINED BELOW $,
         IF NBUT EQL NOTBUT
         THEN BEGIN
              XSEEN = XX(INPUT.ITEM)                       ... GET FROM BEAD BUILT BY GET.ITEM THE X POSITION OF THE GRAPHIC
                                                           INPUT DEVICE $,
              YSEEN = YY(INPUT.ITEM) $,
              LASTSEE = OBJECT.NAME(INPUT.ITEM)            ... LASTSEE IS SET TO THE NAME OF INPUT.ITEM SEEN, I.E., THE
                                                           FIRST ITEM IN THE DISPLAY FILE HAVING A PEN-SENSITIVE AREA
                                                           WITHIN WHICH THE CURSOR COORDINATES LIE . THE NAME IS A POINTER
                                                           TO THE CORRELATION WORD OF THE INPUT.ITEM SEEN, WHICH WILL BE
                                                           THE FIRST WORD IN SOME DATA STRUCTURE BEAD, PERHAPS REPRESENTING
                                                           A NODE, RESISTOR, ETC. $,
              GOTO GET.PENSEE                              ... GO BACK AND GET A PENSEE ON A BUTTON IN ORDER TO DETERMINE
                                                           WHICH OPERATION TO PERFORM ON THE OBJECT TO WHICH THE USER JUST
                                                           POINTED $,
              END
         ELSE GOTO LIGHT.BUTTON(NBUT)                      ... GO TO STATEMENT LABEL CORRESPONDING TO BUTTON NUMBER, WHICH
                                                           IS AN INDEX IN LIGHT.BUTTON, THE ARRAY OF STATEMENT LABELS $,


         DEFINE INTEGER PROCEDURE ISBUTT(PTR) WHERE POINTER PTR TOBE
              BEGIN                                        ... THE PROCEDURE CHECKS IF THE OBJECT CORRELATED WITH THE PENSEE
                                                           IS A LIGHT BUTTON. ARGUMENT IS A POINTER TO A BEAD CONTAINING
                                                           THE PENSEE INFORMATION //
               POINTER OBSN $,
              OBSN = OBJECT.NAME(PTR) $,
              IF TYPE(OBSN) EQL BUT.TYPE                   ... IS TYPE CODE THAT FOR A LIGHT BUTTON //
              THEN ISBUTT = NUMBER(OBSN)                   ... RETURN BUTTON NUMBER STORED IN LIGHT BUTTON BEAD //
              ELSE ISBUTT = NOTBUT                         ... CODE INDICATES PENSEE WAS NOT ON A LIGHT BUTTON $,
              END $,
```

```
  COMMENT REMAINDER OF MAIN PROGRAM PROCESSES LIGHT BUTTONS $,
DELET $  IF TYPE(LASTSEE) EQL NODE OR TYPE(LASTSEE) EQL DEVICE.TYPE ... PROCESS LIGHT BUTTON TO DELETE NODE,
                                                                        DEVICE OR ELEMENT //
         THEN BEGIN                                                    ... DELETE A NODE BY DELETING ALL ITS ATTACHED ELEMENTS FIRST,
                                                                        AND CHECKING THEIR INTERFACING NODES IN TURN. //
              DELET.ELEMENT(U(LASTSEE))                                ... DELET.ELEMENT CHECKS  IF POINTER IS EMPTY BEFORE DELETING. $,
              DELET.ELEMENT(D(LASTSEE)) $,
              DELET.ELEMENT(L(LASTSEE)) $,
              DELET.ELEMENT(R(LASTSEE))
                                                                       ... NODE TO BE DELETED WILL BE DELETED THIS TIME, AT LEAST, SINCE
                                                                        ALL ATTACHED ELEMENTS WILL HAVE PREVIOUSLY BEEN DELETED. $,
              END
              ELSE DELET.ELEMENT(LASTSEE)                              ... DELETE AN ELEMENT POINTED TO BY THE GRAPHIC INPUT DEVICE $,


         DEFINE PROCEDURE DELET.ELEMENT(ELEMENT) WHERE POINTER ELEMENT TOBE
                IF ELEMENT EQL EMPTY OR ELEMENT EQL FORBIDDEN.LEAD      ... SEE IF ELEMENT IS VALID //
                THEN GOTO RETURN
                ELSE BEGIN
                     DELETE(ELEMENT,FROMNODE,TONODE)                    ... DELETE THE ELEMENT.  FROMNODE AND TONODE ARE RETURNED BY DELETE,
                                                                        INDICATING THE NODES BETWEEN WHICH THE ELEMENT WAS ATTACHED.
                                                                        DELETE ALSO UPDATES THOSE ADJACENT NODE BEADS BY
                                                                         UPDATING THE NUMBER OF LEADS AND UPDATING
                                                                        THE POINTERS TO THE DELETED ELEMENT TO BE EMPTY. $,
                     IF LEADS(FROMNODE) EQL 0                          ... SEE IF  NODE FROM WHICH ELEMENT WAS ATTACHED HAS NO LEADS NOW //
                     THEN RMVNOD(FROMNODE)                             ... IF SO, REMOVE IT $,
                     IF LEADS(TONODE) EQL 0
                     THEN RMVNOD(TONODE) $,
                     END $,



  COMMENT DEFINITION OF PROCEDURE DELETE, WHICH IS CALLED ABOVE $,

         DEFINE PROCEDURE DELETE(ELEMENT,FROMNODE,TONODE) WHERE POINTER ELEMENT,FROMNODE,TONODE TOBE
              BEGIN

  COMMENT DELETE DELETES THE ELEMENT GIVEN, UPDATING DATA STRUCTURE $,
              FROMNODE = FROM(ELEMENT) $,
              TONODE = INTO(ELEMENT) $,
              RMV(ELEMENT)                                              ... REMOVE THE REPRESENTATION OF THE ELEMENT FROM THE DISPLAY
                                                                        FILE.  ELEMENT STILL POINTS TO THE BEAD $,
              IF NOT REDRAW.MODE                                        ... IS USER IN AUTO ERASE MODE. IF SO, PICTURE MUST BE REDRAWN
                                                                        AFTER DELETION //
              THEN BEGIN
                   XSEEN = (X(FROMNODE)+X(TONODE))/2                    ... THE OBJECT EX, WHICH IS A BIG X, WILL BE PLOTTED CENTERED
                                                                        OVER THE ELEMENT DELETED IF PICTURE IS NOT TO BE REDRAWN. $,
                   YSEEN = (Y(FROMNODE)+Y(TONODE))/2                    ... XSEEN AND YSEEN ARE AT THE MIDDLE OF THE ELEMENT SEEN, THE
                                                                        PLACE WHERE THE X IS TO BE PLOTTED $,
                   DEFOBJ()                                             ... DEFOBJ IS A PROCEDURE WHICH BEGINS THE DEFINITION OF A
                                                                        COMPOUND DISPLAY OBJECT $,
                   ADDOBJ(SETPT(XSEEN,YSEEN))                           ... ADD TO THE OBJECT BEING DEFINED A POINT, GENERATED BY
                                                                        PROCEDURE SETPT, AT X=XSEEN, Y=YSEEN $,
                   ADDOBJ(CPYOBJ(EX))                                   ... ADD TO THE OBJECT A COPY OF THE STANDARD 'X' USED TO DENOTE
                                                                        DELETIONS. $,
                   P = ENDOBJ()                                         ... ENDOBJ TERMINATES THE COMPOUND OBJECT AND RETURNS A POINTER
                                                                        TO THE OBJECT WHICH WAS JUST BUILT. $,
                   DISPLAY(PTR=PLOT(P)) $,
                   RMV(PTR)                                            ... REMOVE THE X FROM THE DISPLAY FILE AS IT SHOULD NOT APPEAR
                                                                        WHEN THE PICTURE IS REGENERATED WITH THE ELEMENT REMOVED $,
```

```
          END $,
     I = IF TYPE(ELEMENT) EQL SHORT THEN SHTSZ ELSE RESSZ  ... DETERMINE SIZE OF ELEMENT BEAD TO BE RETURNED TO
                                                               FREE STORAGE $,
     IF R(FROMNODE) EQL ELEMENT                          ... ELEMENT IS HORIZONTALLY CONNECTED //
     THEN BEGIN
          FRET(I,ELEMENT)                                ... RETURN ELEMENT TO FREE STORAGE $,
          R(FROMNODE) = EMPTY                            ... FIX NODE BEADS TO WHICH ELEMENT WAS ATTACHED TO INDICATE
                                                          ELEMENT HAS BEEN DELETED $,
          L(TONODE) = EMPTY $,
          END
     ELSE BEGIN                                          ... ELEMENT IS VERTICALLY CONNECTED. SINCE ELEMENTS ARE DEFINED
                                                          ONLY TO THE RIGHT OR BELOW THEIR FROMNODE, WE NEED CHECK ONLY
                                                          THESE TWO POSSIBILITIES //
          FRET(I,ELEMENT) $,
          D(FROMNODE) = EMPTY $,
          U(TONODE) = EMPTY $,
          END $,
     LEADS(FROMNODE) = LEADS(FROMNODE)-1                 ... REDUCE COUNT OF NUMBER OF ELEMENTS ATTACHED TO NODE $,
     LEADS(TONODE) = LEADS(TONODE)-1 $,
     FIXEND()                                            ... SYSTEM KLUDGE $,
     END $,


DEFINE PROCEDURE RMVNOD(NODE) WHERE POINTER NODE TOBE
     BEGIN

COMMENT RMVNOD REMOVES A NODE FROM THE DATA STRUCTURE AND THE DISPLAY FILE $,
     IF NODE EQL NODE1
     THEN GOTO RETURN $,
     P = NODE1 $,
HERE $ PREV = P                                          ... SEARCH NODE CHAIN FOR NODE $,
     P = NEXT(PREV) $,
     IF P EQL EMPTY
     THEN BEGIN
          GOUT (.C. /NODE SPECIFIED NOT ON NODE CHAIN /) $,
          GOTO GET.PENSEE $,
          END
     ELSE IF P NEQ NODE
          THEN GOTO HERE $,
     NEXT(PREV) = NEXT(P)                                ... REMOVE NODE FROM NODE CHAIN $,
     RMV(P)                                              ... REMOVE NODE FROM THE DISPLAY FILE $,
     XSEEN = X(P) $,
     YSEEN = Y(P) $,
     FRET(NODESZ,P)                                      ... RETURN NODE BEAD TO FREE STORAGE $,
     FIXEND() $,
     IF NOT REDRAW.MODE
     THEN BEGIN
          DEFOBJ() $,
          ADDOBJ(SETPT(XSEEN,YSEEN))                     ... PLOT A LARGE X OVER THE NODE BEING DELETED $,
          ADDOBJ(CPYOBJ(EX)) $,
          P = ENDOBJ() $,
          DISPLAY(PTR = PLOT(P)) $,
          RMV(PTR) $,
          END $,
     END $,
```

```
  COMMENT ALLOW VALUE TO BE CHANGED IF ITEM SEEN WAS RESISTOR OR CAPACITOR $,
NEWVAL $ IF TYPE(LASTSEE) EQL RESISTOR OR TYPE(LASTSEE) EQL CAPACITOR
        THEN BEGIN
             GET.ITEM(.C. /TYPE NEW VALUE/,RVAL,NEED.VALUE) ... PROMPT USER AND GET A NEW VALUE $,
             VAL(LASTSEE) = RVAL                            ... STORE NEW VALUE IN ELEMENT BEAD $,
             END $,
        GOTO GET.PENSEE $,

  COMMENT DEFINE HORIZONTAL ELEMENTS $,
HORES $  THIS = FREE(RESSZ) $,                              ... ADD HORIZONTAL RESISTOR. START BY GETTING RESISTOR BEAD FROM
                                                                FREE STORAGE $,
         TYPE(THIS) = RESISTOR                              ... AND SET TYPE COMPONENT $,
         ITEM = HRES                                        ... ITEM TO BE PLOTTED IS PICTURE OF HORIZONTAL RESISTOR. HRES IS
                                                                A POINTER TO THE DISPLAY OBJECT REPRESENTING THE RESISTOR $,
HORIZ $  FROM(THIS) = FROMNODE = NEARNODE(LASTSEE,XSEEN,YSEEN) ... PROCESS HORIZONTAL ELEMENTS. START BY GETTING POINTER TO
                                                                NEAREST NODE OR DEVICE TO CURSOR AT XSEEN,YSEEN USING NEARNODE,
                                                                WHICH IS EXPLAINED BELOW.  THE 'FROM' COMPONENT OF THE ELEMENT BEAD
                                                                IS SET TO POINT TO THE NODE OR DEVICE RETURNED. $,
         INTO(THIS) = TONODE = GETNODE(X(FROMNODE)+GRIDSP,Y(FROMNODE)) ... FIND NODE OR DEVICE AT END OF ELEMENT OR
                                                                CREATE A NODE THERE $,
         IF R(FROMNODE) EQL FORBIDDEN.LEAD OR L(TONODE) EQL FORBIDDEN.LEAD ... CHECK IF ELEMENT REQUIRES A CONNECTION TO
                                                                A FORBIDDEN LEAD OF A DEVICE. //
         THEN BEGIN
              ERROR(1)                                      ... USE ERROR OUTPUT ROUTINE DEFINED ABOVE TO PRINT ERROR MESSAGE
                                                                CORRESPONDING TO CODE OF 1 $,
              IF TYPE(THIS) EQL SHORT                       ... START OVER BY GETTING ANOTHER PENSEE ON A LIGHT BUTTON, BUT
                                                                FIRST RETURN TO FREE STORAGE BEADS CREATED BUT WHICH COULD NOT
                                                                BE ATTACHED //
              THEN FRET(SHTSZ,THIS)
              ELSE FRET(RESSZ,THIS)                         ... RESISTORS AND CAPACITORS ARE THE SAME SIZE $,
              IF NODE.CREATED                               ... CHECK IF A NODE WAS CREATED AT THE RIGHT-HAND END OF THE
                                                                NEW ELEMENT. SINCE ELEMENT CAN NOT BE ATTACHED,
                                                                THE NODE WILL HAVE NO LEADS AND SHOULD BE DELETED.
                                                                NODE.CREATED IS SET WITHIN PROCEDURE NEARNODE. $,
              THEN BEGIN
                   RMV(CREATED.NODE)                        ... REMOVE FROM DISPLAY FILE $,
                   FRET(NODESZ,CREATED.NODE)                ... AND FROM DATA STRUCTURE $,
                   FIXEND() $,
                   END $,
              GOTO GET.PENSEE $,
              END $,
         IF R(FROMNODE) NEQ EMPTY                           ... IF SOMETHING IS ALREADY IN POSITION WHERE NEW ELEMENT IS TO
                                                                BE ATTACHED, DELETE IT AND SET FLAG SO PICTURE WILL BE REDRAWN
                                                                IF IN AUTO ERASE MODE. //
         THEN BEGIN
               DELETE (R(FROMNODE),FROMNODE,TONODE) $,
               ELEMENT.DELETED = TRUE $,
               END $,
         R(FROMNODE) = THIS                                 ... MAKE NODE OR DEVICE BEADS POINT TO NEW ELEMENT $,
         L(TONODE) = THIS $,
ALL $    LEADS(FROMNODE) = LEADS(FROMNODE)+1                ... COME HERE FOR ALL ELEMENTS AND INCREASE ELEMENT COUNT ON
                                                                NODES OR DEVICES $,
         LEADS(TONODE) = LEADS(TONODE)+1 $,
         IF TYPE(THIS) NEQ SHORT                            ... IF 'THIS' POINTS TO A RESISTOR OR CAPACITOR BEAD , GET A NAME
                                                                FOR IT BY CALLING PROCEDURE NAMEELEM //
         THEN BEGIN
              VAL(THIS) = 0                                 ... SET VALUE TO BE 0 INITIALLY $,
              NAME(THIS) = NAMEELEM(TYPE(THIS)) $,
              END $,
         DEFOBJ()                                           ... BEGIN OBJECT DEFINITION. OBJECT WILL INCLUDE A POINT COMMAND
                                                                TO DRAW THE ELEMENT AT THE CORRECT POSITION, THE PICTURE OF THE
```

```
                                                    ELEMENT AND THE TEXT OF THE USER'S NAME,IF ANY. $,
 ADDOBJ(SETPT(X(FROMNODE),Y(FROMNODE)))            ... ADD A SET POINT AND THEN THE PICTURE OF THE ELEMENT $,
 ADDOBJ(CPYOBJ(ITEM)) $,
 IF TYPE(THIS) NEQ SHORT
 THEN ADDOBJ(TEXT(NAME(THIS)))                     ... ADD USER'S NAME FOR ELEMENT $,
 ELEM.PIC = ENDOBJ()                               ... TERMINATE THE OBJECT DEFINITION . ELEM.PIC IS POINTER TO
                                                    OBJECT $,
 END.OF.DISPLAY.FILE = ADDED.ELEMENT = PENSNS(PLOT(ELEM.PIC,THIS,END.OF.DISPLAY.FILE)) ... PLOT OBJECT, 'NAME'
                                                    IS THIS.  NAME IS POINTER TO CORRELATION WORD. NOTE THAT
                                                    DISPLAY 'NAME' IS NOT USER'S NAME, BUT A POINTER TO THE BEAD
                                                    DESCRIBING THE ELEMENT . OBJECT IS PLOTTED AT THE END OF THE
                                                    DISPLAY FILE $,
IF REDRAW.MODE AND  ELEMENT.DELETED                ... IF IN AUTOERASE MODE, AND AN ELEMENT WAS DELETED WHILE
                                                   ADDING A NEW ELEMENT, PICTURE MUST BE REDRAWN //
 THEN BEGIN
      ELEMENT.DELETED = FALSE $,
      GOTO REWRITE $,
      END
 ELSE BEGIN

      IF NODE.CREATED
      THEN DISPLAY(CREATED.NODE,ADDED.ELEMENT)     ... DISPLAY NODE AND ELEMENT JUST CREATED. NOTE THAT PROCEDURE
                                                    PLOT DOES NOT TRANSMIT DISPLAY COMMANDS TO THE TERMINAL , BUT
                                                    ONLY ADDS OBJECTS TO THE GRAPHSYS DATA STRUCTURE. PROCEDURE
                                                    DISPLAY MUST BE CALLED TO SEND COMMANDS //
      ELSE DISPLAY(ADDED.ELEMENT) $,
      GOTO GET.PENSEE $,
      END $,
```

```
  COMMENT DEFINE VERTICAL ELEMENTS. COMMENTS ARE MUCH THE SAME AS FOR HORIZONTAL ELEMENTS IN THE PRECEEDING SECTION $,
VERES $  TYPE(THIS = FREE(RESSZ)) = RESISTOR              ... NOTE USE OF PHRASE SUBSTITUTION $,
         ITEM = VRES $,
VERT $   FROM(THIS) = FROMNODE = NEARNODE(LASTSEE,XSEEN,YSEEN) $,
         INTO(THIS) = TONODE = GETNODE(X(FROMNODE),Y(FROMNODE)-GRIDSP) $,
         IF D(FROMNODE) EQL FORBIDDEN.LEAD OR U(TONODE) EQL FORBIDDEN.LEAD
         THEN BEGIN
              ERROR(1) $,
              IF TYPE(THIS) EQL SHORT
              THEN FRET(SHTSZ,THIS)
              ELSE FRET(RESSZ,THIS) $,
              IF NODE.CREATED
              THEN BEGIN
                   RMV(CREATED.NODE) $,
                   FRET(NODESZ,CREATED.NODE) $,
                   FIXEND() $,
                   END $,
              GOTO GET.PENSEE $,
              END $,
         IF D(FROMNODE) NEQ EMPTY
         THEN BEGIN
              ELEMENT.DELETED = TRUE $,
              DELETE(D(FROMNODE),FROMNODE,TONODE) $,
              END $,
         D(FROMNODE) = THIS $,
         U(TONODE) = THIS $,
         GOTO ALL $,
HOCAP $  TYPE(THIS = FREE(CAPSZ)) = CAPACITOR          ... HORIZONTAL CAPACITOR $,
         ITEM = HCAP $,
         GOTO HORIZ $,
VECAP $  TYPE(THIS = FREE(CAPSZ)) = CAPACITOR          ... VERTICAL CAPACITOR $,
         ITEM = VCAP $,
         GOTO VERT $,
HOSHT $  TYPE(THIS = FREE(SHTSZ)) = SHORT              ... HORIZONTAL SHORT $,
         ITEM = HSHT $,
         GOTO HORIZ $,
VESHT $  TYPE(THIS = FREE(SHTSZ)) = SHORT              ... VERTICAL SHORT $,
         ITEM = VSHT $,
         GOTO VERT $,
ERSON $  REDRAW.MODE = TRUE                            ... SET AUTO ERASE MODE SWITCH SO PICTURE WILL BE RETRANSMITTED
                                                        AFTER EVERY DELETION $,
         GOTO GET.PENSEE $,
ERSOFF $ REDRAW.MODE = FALSE $,
         GOTO GET.PENSEE $,
EXIT $   CHNCOM(0)                                     ... EXIT FROM PROGRAM TO TIMESHARING SYSTEM COMMAND LEVEL $,

  COMMENT REMAINDER OF MAIN PROGRAM PROCESSES DEVICE DEFINITIONS, THE EDITING OF DEVICES AND THE USAGE OF DEVICES $,
```

```
  COMMENT PROCESS LIGHT BUTTON TO DEFINE A DEVICE $,
DEF.DEVICE $ DEFINITION.MODE = TRUE $,
        DEVICE = DEFINITION.POINTER = FREE(DEFINITION.SIZE) ... GET A DEVICE DEFINITION BEAD $,
        DEFSUB(DEVICE)                                  ... BEGIN SUBPICTURE DEFINITION. NAME OF SUBPICTURE IS 'DEVICE'. $,
        END.OF.DISPLAY.FILE = DEVICE $,
  COMMENT  THE DISPLAY OBJECT TO REPRESENT A DEVICE DEFINITION IS A SUBPICUTURE.  UNTIL A SUBPICTURE IS TERMINATED
     BY A CALL TO PROCEDURE ENDSUB,  THE SUBPICTURE CAN NOT BE DISPLAYED AS A FINISHED ENTITY BY
     A CALL TO DISPLAY WITH THE NAME OF THE SUBPICTURE, I.E. DISPLAY(DEVICE).  THEREFORE WHILE DEFINING A DEVICE THE
     PROGRAM MUST EXPLICITLY KNOW WHAT INFORMATION IN THE DISPLAY FILE IS TO REDISPLAYED WHEN
     THE SCREEN IS ERASED AND THE PICTURE IS TO BE REDRAWN.  IN ADDITION OBJECTS MUST ALWAYS BE ADDED TO
     THE SET OF OBJECTS COMPRISING THE SUBPICTURE DEFINITION RATHER THAN TO THE CIRCUIT WHICH WAS BEING
     DEFINED BEFORE THE USER ENTERED DEVICE DEFINITION MODE.  THEREFORE DEVICE IS A POINTER TO THE
      CORRELATION WORD OF THE FIRST OBJECT IN THE SECTION OF THE DISPLAY FILE FOR THE SUBPICTURE DEFINITION.
     END.OF.DISPLAY.FILE POINTS TO THE CORRELATION WORD OF THE LAST OBJECT ADDED TO THE
     SUBPICTURE DEFINITION. $,
        STORE.NODE = NODE1                              ... PUT AWAY POINTER TO BEGINNING OF MAIN CIRCUIT DATA STRUCTURE. $,
        ERASER() $,
        GET.ITEM(.C. /INDICATE POSITION OF INITIAL NODE FOR DEVICE/,INPUT.ITEM,NEED.ABPOS) $,
        NODE1 = GETNODE(XX(INPUT.ITEM),YY(INPUT.ITEM)) ... NODE1 NOW POINTS TO THE BEGINNING OF THE DEVICE DATA
                                                         STRUCTURE. $,
        BUTTONS(MENU2)                                  ... GENERATE LIGHT BUTTONS FOR USE DURING DEVICE DEFINITION $,
        DISPLAY(FBUT,LBUT) $,
        DISPLAY(NODE1) $,
        IF BEGIN.DEVICE.CHAIN EQL EMPTY
        THEN BEGIN.DEVICE.CHAIN = DEFINITION.POINTER    ... IF THIS IS THE FIRST DEVICE TO BE DEFINED, PUT IT AS FIRST
                                                           DEVICE ON CHAIN OF DEVICE DEFINITIONS //
        ELSE NEXT(LASTDEF) = DEFINITION.POINTER         ... PUT NEW DEVICE DEFINITION BEAD INTO LIST OF DEFINITIONS $,
        NEXT(DEFINITION.POINTER) = EMPTY $,
        NUM(DEFINITION.POINTER) = 0                     ... SET NUMBER OF USAGES TO BE 0 $,
        FSTNODE(DEFINITION.POINTER) = NODE1             ... SET DEVICE DEFINITION BEAD TO POINT TO DATA STRUCTURE FOR
                                                         EQUIVALENT CIRCUIT $,
        GOTO GET.PENSEE                                 ... PROGRAM IS NOW SETUP TO OPERATE WHILE DEFINING A DEVICE AS IF
                                                         IT WAS ON THE MAIN CIRCUIT LEVEL $,
```

```
  COMMENT PROCESS LIGHT BUTTON TO TERMINATE DEVICE EDITING OR DEFINITION $,
TERM.DEVICE $ NODE1 = STORE.NODE                          ... RESET NODE1, THE POINTER TO THE CIRCUIT DATA STRUCTURE, TO BE
                                                           THAT FOR THE MAIN CIRCUIT $,
        IF EDITING.DEFINITION
        THEN BEGIN
             EDITING.DEFINITION = FALSE $,
             RMV(DEVICE)                                  ... REMOVE SUBPICTURE CALL FOR DEVICE FROM DISPLAY FILE $,
             END
        ELSE BEGIN                                        ... DEFINITION IS BEING TERMINATED //
              DEFINITION.MODE = FALSE $,
             GET.ITEM(.C. /SPECIFY DEVICE NAME/,INPUT.ITEM,NEED.CHARS) $,
             NAME(DEFINITION.POINTER) = COPYC(0,0,INPUT.ITEM) ... MAKE A COPY OF THE TEXT OF THE NAME $,
             BOXES(DEFINITION.POINTER)                    ... PROCEDURE BOXES REQUESTS USER TO SPECIFY THE NUMBER OF
                                                           TERMINAL LEADS AND THEIR ORIENTATION $,
             TRMNLS(DEFINITION.POINTER)                   ... PROCEDURE TRMNLS REQUESTS USER TO INDICATE WHICH LEADS IN THE
                                                           DEVICE DEFINITION ARE THE TERMINAL LEADS. THIS IS REQUIRED IN
                                                           ORDER FOR AN ANALYSIS PROGRAM TO SUBSTITUTE THE EQUIVALENT
                                                           CIRCUIT FOR THE DEVICE REPRESENTATION. $,
             LASTDEF = DEFINITION.POINTER                 ... SET LAST DEFINITION ON THE LIST OF DEFINITIONS TO BE THIS
                                                           DEFINITION $,
             ENDSUB()
             END $,                                       ... TERMINATE SUBPICTURE DEFINITION $,
        END.OF.DISPLAY.FILE = LASTOBJ(MAIN.FILE) $,
COMMENT   SET THE END OF THE DISPLAY FILE TO BE THAT FOR THE MAIN DISPLAY FILE BY CALLING PROCEDURE LASTOBJ,
           WHICH FINDS THE LAST OBJECT IN THE MAIN DISPLAY FILE, OR IN A SUBPICTURE DEFINITION. $,
        BUTTONS(MENU1)                                    ... REGNERATE LIGHT BUTTONS FOR WORKING AT MAIN CIRCUIT LEVEL $,
        GOTO REWRITE $,
```

```
  COMMENT  PROCESS LIGHT BUTTON TO USE DEVICE $,
USE.DEVICE $ GET.ITEM(.C. /SPECIFY DEVICE NAME/,INPUT.ITEM,NEED.CHARS) $,
         IF (P = BEGIN.DEVICE.CHAIN) EQL EMPTY
         THEN GOTO NOT.FOUND                          ... START AT BEGINNING OF DEVICE CHAIN AND SEARCH FOR DEVICE
                                                          SPECIFIED //
         ELSE GOTO START $,
REPEAT $ IF (P = NEXT(P)) EQL EMPTY
         THEN BEGIN
NOT.FOUND $    ERROR(2) $,
               GOTO GET.PENSEE $,
               END $,
START $  COMPAR(NAME(P),INPUT.ITEM,REPEAT,FOUND,REPEAT) ... PROCEDURE COMPAR TRANSFERS TO LABEL 'FOUND' IF THE NAME OF A
                                                          DEVICE MATCHES THE INPUT NAME. OTHERWISE IT GOES TO THE LABEL
                                                          'REPEAT' $,
FOUND $  USAGE = LASTSEE $,
         BOX.STYLE = CHOICE(STYLE(P),USAGE,ILLEGAL.ATTACHMENT) ... PROCEDURE CHOICE PRESENTS THE USER WITH THE
                                                          ROTATIONAL CONFIGURATIONS OF THE SYMBOL USED TO REPRESENT THE
                                                          DEVICE DEFINITION TO WHICH P POINTS. USAGE IS A POINTER TO NODE TO
                                                          BE REPLACED BY THE DEVICE. THE SYMBOL SELECTED BY THE USER
                                                          WITHIN CHOICE MUST HAVE LEADS IN THE DIRECTIONS WHERE THE
                                                          NODE PRESENTLY HAS ATTACHMENTS--IF NOT, ILLEGAL.ATTACHMENT IS SET
                                                          TO BE TRUE SO THE SUBSTITUTION OF THE DEVICE CAN NOT OCCUR.
                                                          BOX.STYLE IS A POINTER TO THE DISPLAY OBJECT FOR THE SYMBOL
                                                          TO REPRESENT THE DEVICE $,
         IF ILLEGAL.ATTACHMENT
         THEN BEGIN
              ERROR(1) $,
              GOTO GET.PENSEE $,
              END
         ELSE BEGIN
 COMMENT ALTER THE NODE BEAD TO MAKE IT A DEVICE USAGE BEAD BY RESETTING SEVERAL COMPONENTS $,
              TYPE(USAGE) = DEVICE.TYPE $,
              NUM(P) = NUM(P)+1                       ... NUM IS THE NUMBER OF USAGES OF THE DEVICE $,
              NAME(USAGE) = COPYC(0,0,NAME(P),NUMTOC(NUM(P))) ... PROCEDURE COPYC CONCATENATES TWO CHARACTER STRINGS,THE
                                                          NAME OF DEVICE AS GIVEN WHEN IT WAS DEFINED AND THE NUMBER OF
                                                          USAGES. PROCEDURE NUMTOC CONVERTS A NUMBER TO A CHARACTER STRING. $,
              DEF.PTR(USAGE) = P                      ... SET POINTER TO DEVICE DEFINITION SO THAT AN ANALYSIS PROGRAM,
                                                          IF ONE EXISTED, COULD SUBSTITUTE THE EQUIVALENT CIRCUIT FOR THE
                                                          DEVICE IN CALCULATING THE PROPERTIES OF THE CIRCUIT $,
              DEFOBJ()                                ... DEFINE A DISPLAY OBJECT FOR THE DEVICE CONSISTING OF A
                                                          POINT TO PLOT THE DEVICE AT THE NODE POSITION $,
              ADDOBJ(SETPT(X(USAGE),Y(USAGE))) $,
              ADDOBJ( BOX.STYLE)                       ... AND THE DISPLAY OBJECT FOR THE BOX WITH THE NUMBER OF LEADS
                                                          SPECIFIED BY THE DEVICE STYLE $,
              ADDOBJ(TEXT(NAME(USAGE)))               ... AND THE NAME OF THIS USAGE OF THE DEVICE $,
              BOX.STYLE = ENDOBJ() $,
              PENSNS(RPL(BOX.STYLE,USAGE))            ... REPLACE THE DISPLAY OBJECT FOR THE NODE, WHICH HAS THE NAME
                                                          USAGE,  BY THE DEVICE DISPLAY OBJECT, BOX.STYLE, AND MAKE IT PEN
                                                          SENSITIVE. $,
              GOTO REWRITE                            ... PICTURE WILL BE REDRAWN TO SHOW DEVICE $,
              END $,
```

```
  COMMENT PROCESS LIGHT BUTTON TO EDIT DEVICE $,
EDIT.DEVICE $ GET.ITEM(.C. /SPECIFY DEVICE NAME/,INPUT.ITEM,NEED.CHARS) $,
         IF (P = BEGIN.DEVICE.CHAIN) EQL EMPTY
         THEN GOTO NOT.FOUND                                  ... LOOK FOR DEVICE ON DEVICE DEFINITION CHAIN //
         ELSE GOTO TEST $,
R1 $     IF (P = NEXT(P)) EQL EMPTY
         THEN ERROR(2)
         ELSE BEGIN
TEST $        SUB=P
              COMPAR(NAME(P),INPUT.ITEM,R1,F1,R1) $,
F1 $          DEVICE = CALL(P)                                ... GENERATE A DISPLAY OBJECT WHICH IS A SUBPICTURE CALL FOR THE
                                                                  DEVICE DEFINITION $,
              ERASER() $,
              DISPLAY(DEVICE) $,
              BUTTONS(MENU2) $,
              EDITING.DEFINITION = TRUE $,
              END.OF.DISPLAY.FILE = LASTOBJ(SUB) $,
 COMMENT  SET END.OF.DISPLAY.FILE TO BE A POINTER TO THE CORRELATION WORD OF THE LAST OBJECT IN THE SUBPICTURE
      DEFINITION SO OBJECTS CAN BE ADDED TO THE END OF THE SUBPICTURE DEFINITION.  FIRST 2 WORDS OF THE DEFINITION
      BEAD POINTED TO BY P ARE THE SUBPICTURE CORRELATION WORDS. $,
              STORE.NODE = NODE1 $,
              NODE1 = FSTNODE(P)                              ... SET DATA STRUCTURE UPON WHICH THE PROGRAM IS TO OPERATE TO BE
                                                                  THE CIRCUIT CORRESPONDING TO THE DEVICE DEFINITION $,
              DISPLAY(FBUT,LBUT) $,
              END $,
         GOTO GET.PENSEE $,
```

```
COMMENT  THE FOLLOWING PROCEDURES ARE A GROUP OF SMALL SERVICE ROUTINES. $,

       DEFINE PROCEDURE FIXEND TOBE
COMMENT  FIXEND UPDATES THE POINTER TO THE END OF THE DISPLAY FILE TO AVOID A  PROBLEM WHEN THE LAST OBJECT
         IN THE DISPLAY FILE IS DELETED.  PROCEDURE LASTOBJ RETURNS A POINTER TO THE LAST OBJECT
         IN EITHER A SUBPICTURE DEFINITION OR THE MAIN DISPLAY FILE.  IF A SUBPICTURE IS BEING EDITED, SUB POINTS TO
         THE DEVICE DEFINITION BEAD, WHICH BEGINS WITH THE SUBPICTURE CORRELATION WORD.  $,
            BEGIN
            END.OF.DISPLAY.FILE = LASTOBJ(IF EDITING.DEFINITION THEN SUB ELSE MAIN.FILE) $,
            END $,


     DEFINE POINTER PROCEDURE NEARNODE(LASTSEE,XSEEN,YSEEN) WHERE POINTER LASTSEE, INTEGER XSEEN,YSEEN TOBE
            BEGIN
COMMENT NEARNODE GIVES A POINTER TO THE NODE NEAREST TO XSEEN,YSEEN $,
            IF TYPE(LASTSEE) EQL NODE OR TYPE(LASTSEE) EQL DEVICE.TYPE
            THEN NEARNODE = LASTSEE
            ELSE BEGIN
                 FROMNODE = FROM(LASTSEE) $,
                 TONODE = INTO(LASTSEE) $,
                 IF (XSEEN-X(FROMNODE)+Y(FROMNODE)-YSEEN) LES (X(TONODE)-XSEEN+YSEEN-Y(TONODE)) ... WORKS ONLY WHEN
                                                          NODES ARE LAID OUT ON A GRID //
                 THEN NEARNODE = FROMNODE
                 ELSE NEARNODE = TONODE $,
                 END $,
            END $,


       DEFINE PROCEDURE INITARDS TOBE
            BEGIN
            SGNON(1)                              ... CONNECT TO DISPLAY UNIT AND INITIALIZE GRAPHSYS ROUTINES. $,
            ERASER() $,
            PENCHK(TRUE)                          ... PENCHK TELLS GRAPHSYS TO CHECK EACH GRAPHICAL POSITION TO SEE
                                                   IF IT IS IN THE PEN-SENSITIVE AREA OF AN OBJECT $,
            PENDLT(50,50)                         ... PENDLT SETS THE SIZE OF AREA WITHIN WHICH OBJECTS ARE
                                                   PEN-SENSITIVE.  ALL OBJECTS ARE SENSITIVE WITHIN A SQUARE OF
                                                   HALF WIDTH 50 CENTERED AT THE CALCULATED CENTER OF THE OBJECT $,
            END $,
```

```
      DEFINE POINTER PROCEDURE NAMEELEM(TYPE) WHERE INTEGER TYPE TOBE
           BEGIN
COMMENT NAMEELEM GETS A NAME FOR AN ELEMENT AND ADDS IT TO THE BEAD FOR THE ELEMENT $,
           IF TYPE EQL CAPACITOR
           THEN BEGIN
                NUMBER.OF.OCCURRENCES(1) = NUMBER.OF.OCCURRENCES(1)+1 $,
                NAMEELEM = COPYC(0,0,.C. /C/,NUMTOC(NUMBER.OF.OCCURRENCES(1))) ... COPYC CONCATENATES C WITH
                                                    CAPACITOR NUMBER TO GIVE A NAME SUCH AS C5 $,
                END
           ELSE BEGIN
COMMENT  OTHERWISE THE ELEMENT MUST BE A RESISTOR $,
                NUMBER.OF.OCCURRENCES(2) = NUMBER.OF.OCCURRENCES(2)+1 $,
                NAMEELEM = COPYC(0,0,.C. /R/,NUMTOC(NUMBER.OF.OCCURRENCES(2))) $,
                END $,
           END $,


      DEFINE POINTER PROCEDURE NAMENODE TOBE
           BEGIN
COMMENT NAMENODE GIVES A UNIQUE NAME TO EACH NODE. NAME IS 'N' FOLLOWED BY AN INTEGER $,
           NUMBER.OF.OCCURRENCES(0)=NUMBER.OF.OCCURRENCES(0)+1 $,
           NAMENODE = COPYC(0,0,.C. /N/,NUMTOC(NUMBER.OF.OCCURRENCES(0))) $,
           END $,


COMMENT END OF THIS COMPILATION.  PROCEDURES BOXES, CHOICE AND TRMNLS ARE IN THE FOLLOWING COMPILATION. $,
      END FINI
```

```
    BEGIN

COMMENT SEPARATE COMPILATION INCLUDING PROCEDURES FOR DEFINING AND USING DEVICES. PROCEDURES ARE BOXES,
        CHOICE,TRMNLS,SAMPLE.BOX,MAKEBOX $,

    INTEGER NEED.PENSEE $,
    POINTER RBOX,P1,P2,P3,P4,BOX1,BOX2,BOX3,BOX4,BOX5,BOX6,BOX7,BOX8,BOX9,BOX10,BOX11,BOKS1,BOKS2,BOKS3,BOKS4,SBOX1,
     SBOX2,SBOX3,SBOX4,SBOX5,SBOX6,SBOX7,SBOX8,SBOX9,SBOX10,SBOX11,ITEM,MESSAGE,EMPTY,FORBIDDEN.LEAD $,
    PROCEDURE ERASER,ERROR,ADDOBJ,DEFOBJ,DISPLAY,GET.ITEM $,
    POINTER PROCEDURE PENSNS,PLOT,ENDOBJ,RMV,CPYOBJ $,

COMMENT THE FOLLOWING DECLARATIONS ARE FOR COMPONENTS IN NODE AND DEVICE USAGE BEADS AND ARE THE SAME AS IN THE
        MAIN PROGRAM $,
    INTEGER COMPONENT STYLE $,
    POINTER COMPONENT U,D,R,L,TRMNL1,TRMNL2,TRMNL3,TRMNL4 $,
    STYLE $=$ 2                                    ... THIS IS THE STYLE COMPONENT OF A DEVICE DEFINITION BEAD --
                                                    SEE FIG 2 $,
    U $=$ 5                                        ... THESE ARE THE CONNECTION COMPONENTS OF A NODE BEAD -- SEE FIG
                                                    1 $,

    L $=$ 6 $,
    R $=$ 7 $,
    D $=$ 8 $,
    TRMNL1 $=$ 7                                   ... THESE ARE THE COMPONENTS IN A DEVICE DEFINITION BEAD
                                                    INDICATING HOW THE TERMINAL NODES ARE LOCATED IN THE EQUIVALENT
                                                    CIRCUIT. $,

    TRMNL2 $=$ 8 $,
    TRMNL3 $=$ 9 $,
    TRMNL4 $=$ 10 $,

COMMENT DECLARATION OF 'PENSEE' BEAD AS RETURNED BY GET.ITEM. MORE COMPLETE BEAD DECLARATION IS IN MAIN PROGRAM $,
    POINTER COMPONENT OBJECT.NAME $,
    OBJECT.NAME $=$ 2                              ... THE POINTER TO THE CORRELATION WORD -- THE OBJECT NAME-- IS
                                                    RETURNED IN THE THIRD WORD OF A PENSEE ITEM $,

    PRESET
        BEGIN
        EMPTY = 0                                  ... CODE FOR A NULL POINTER $,
        FORBIDDEN.LEAD = -1                        ... CODE TO INDICATE THIS DIRECTION ON A DEVICE CAN NOT HAVE A
                                                    CONNECTION $,
        NEED.PENSEE = 1                            ... CODE TO INDICATE NEXT INPUT ITEM MUST BE A PENSEE $,
        END $,
```

```
DEFINE PROCEDURE BOXES(DEFINITION.PTR) WHERE POINTER DEFINITION.PTR TOBE
     BEGIN                                          ... BOXES DISPLAYS THE FOUR BASIC BOX STYLES AND ALLOWS THE USER
                                                        TO CHOOSE THE ONE APPROPRIATE TO HIS DEVICE //
      BOOLEAN FIRST.TIME $,
     PROCEDURE SAMPLE.BOX,MAKEBOX $,
     PRESET FIRST.TIME = TRUE $,
     IF FIRST.TIME                                  ... SEE IF THIS IS THE FIRST TIME THIS ROUTINE WAS CALLED //
     THEN BEGIN
          MAKEBOX(BOX1,BOX2,BOX3,BOX4,BOX5,BOX6,BOX7,BOX8,BOX9,BOX10,BOX11) $, ... CREATE THE BOX SYMBOLS TO BE
                                                        USED WHEN DEVICES ARE PLOTTED. THERE ARE 4 BASIC BOX STYLES, AND
                                                        11 CONFIGURATIONS WHEN ROTATION IS CONSIDERED //
           SAMPLE.BOX(SBOX1,SBOX2,SBOX3,SBOX4,SBOX5,SBOX6,SBOX7,SBOX8,SBOX9,SBOX10,SBOX11) ... GET DISPLAY
                                                        OBJECTS FOR THE BOXES USED TO REPRESENT DEVICES. THEY INCLUDE A
                                                        POINT TO POSITION THE BOX IN THE SELECTION AREA AT THE BOTTOM
                                                        OF THE SCREEN $,
          FIRST.TIME = FALSE $,
          END $,
     PENSNS(P1 = PLOT(SBOX1))                       ... ADD A PEN SENSITIVE 2 LEAD BOX TO THE DISPLAY FILE WITH NAME
                                                        P1 $,
     PENSNS(P2 = PLOT(SBOX3)) $,
     PENSNS(P3 = PLOT(SBOX7)) $,
     PENSNS(P4 = PLOT(SBOX11)) $,
     DISPLAY(P1,P4)                                 ... SEND DISPLAY FILE BETWEEN OBJECTS P1 AND P4 TO THE TERMINAL
                                                        $,
     GET.ITEM(.C. /POINT TO DEVICE STYLE/,ITEM,NEED.PENSEE) ... GET PENSEE TO INDICATE WHAT STYLE IS DESIRED.
                                                        ITEM IS A POINTER TO THE BEAD DESCRIBING THE PENSEE $,
     IF OBJECT.NAME(ITEM) EQL P1                    ... SEE IF NAME OF OBJECT SEEN IS FIRST BOX STYLE. P1 IS A
                                                        POINTER TO THE CORRELATION WORD FOR THE FIRST BOX STYLE--SEE IF
                                                        IT IS THE SAME AS THE CORRELATION WORD POINTER RETURNED AFTER
                                                        THE PENSEE //
     THEN STYLE(DEFINITION.PTR) = 1                 ... STORE THE CODE FOR THE DEVICE STYLE CHOSEN IN THE DEVICE
                                                        DEFINITION BEAD $,
     IF OBJECT.NAME(ITEM) EQL P2
     THEN STYLE(DEFINITION.PTR) = 2 $,
     IF OBJECT.NAME(ITEM) EQL P3
     THEN STYLE(DEFINITION.PTR) = 3 $,
     IF OBJECT.NAME(ITEM) EQL P4
     THEN STYLE(DEFINITION.PTR) = 4 $,
     RMV(P1,P4)                                     ... REMOVE OBJECTS P1 THROUGH P4 FROM THE DISPLAY FILE $,
     GOTO RETURN $,
     END $,
```

```
DEFINE POINTER PROCEDURE CHOICE(BOX.STYLE,USAGE,ILLEGAL.ATTACHMENT) WHERE INTEGER BOX.STYLE $,
POINTER USAGE $,
BOOLEAN ILLEGAL.ATTACHMENT TOBE
     BEGIN                                          ... CHOICE DETERMINES THE ROTATIONAL CONFIGURATION OF THE BASIC
                                                    BOX STYLE TO BE USED AT A GIVEN PLACE IN THE CIRCUT. BOX.STYLE
                                                    WAS DETERMINED WHEN THE DEVICE WAS DEFINED BY USING THE
                                                    PROCEDURE BOXES. USAGE IS THE POINTER TO THE NODE BEAD BEING
                                                    REPLACED BY THE DEVICE. DEVICES WITH LESS THAN 4 LEADS HAVE A
                                                    CERTAIN SIDE TO WHICH ELEMENTS CAN NOT BE ATTACHED. IF THE NODE
                                                    BEING REPLACED HAS CONNECTIONS IN THOSE DIRECTIONS,
                                                    ILLEGAL.ATTACHMENT IS SET TO BE TRUE TO INDICATE THE USER HAS
                                                    TRIED TO INCORRECTLY CONNECT THE DEVICE. THE ILLEGAL DIRECTIONS
                                                    ARE SET TO THE CODE FORBIDDEN.LEAD SO NO CONNECTIONS WILL LATER
                                                    BE MADE IN THOSE DIRECTIONS //
      MESSAGE = .C. /POINT TO DEVICE STYLE WITH PROPER ORIENTATION/ $,
     ILLEGAL.ATTACHMENT = FALSE $,
START $   IF BOX.STYLE EQL 1                        ... BOX HAS 2 LEADS ON OPPOSITE SIDES //
     THEN BEGIN                                     ... COMMENTS ON THE FOLLOWING SEQUENCE FOR GETTING THE USER TO
                                                    INDICATE WHICH ROTATIONAL ORIENTATION IS CORRECT ARE MUCH LIKE THOSE FOR
                                                    THE SEQUENCE GIVEN IN PROCEDURE BOXES ABOVE //

          PENSNS(BOKS1 = PLOT(SBOX1))               ... ORIENTATION WITH LEADS AT 3 AND 9 O'CLOCK $,
          PENSNS(BOKS4 = PLOT(SBOX2))               ... ORIENTATION WITH LEADS AT 6 AND 12 O'CLOCK $,
          DISPLAY(BOKS1,BOKS4) $,
          GET.ITEM(MESSAGE,ITEM,NEED.PENSEE) $,
          IF OBJECT.NAME(ITEM) EQL BOKS1
          THEN BEGIN
               IF U(USAGE) NEQ EMPTY OR D(USAGE) NEQ EMPTY ... CHECK IF DIRECTIONS IN WHICH BOX HAS NO LEADS
                                                    HAVE BEEN ALREADY USED IN NODE BEAD BEING REPLACED //

               THEN GOTO ERR $,
               CHOICE = CPYOBJ(BOX1)                ... DEVICE IS BEING USED CORRECTLY. SET VALUE OF THE PROCEDURE TO
                                                    BE A POINTER TO A COPY OF THE OBJECT TO BE DISPLAYED TO
                                                    REPRESENT THE BOX $,
               U(USAGE) = D(USAGE) = FORBIDDEN.LEAD ... OTHER DIRECTIONS CAN NEVER BE USED FOR THIS 2 LEAD
                                                    DEVICE. SET THE UP AND DOWN COMPONENTS OF THE DEVICE USAGE BEAD
                                                    TO A CODE INDICATING NO CONNECTIONS CAN BE MADE TO THEM $,

               END
          ELSE IF OBJECT.NAME(ITEM) EQL BOKS4
               THEN BEGIN
                    IF R(USAGE) NEQ EMPTY OR L(USAGE) NEQ EMPTY
                    THEN GOTO ERR $,
                    CHOICE = CPYOBJ(BOX2) $,
                    R(USAGE) = L(USAGE) = FORBIDDEN.LEAD $,
                    END
               ELSE GOTO BADSEE $,
          RMV(BOKS1,BOKS4)                          ... REMOVE OBJECTS BOKS1 TO BOKS4 FROM THE DISPLAY FILE $,
          GOTO RETURN $,
          END $,
     IF BOX.STYLE EQL 2                             ... BASIC CONFIGURATION HAS 2 LEADS AT 12 AND 3 O'CLOCK AND 4
                                                    POSSIBLE ROTATIONAL CONFIGURATIONS, ASSUMING BOTH LEADS ARE
                                                    EQUIVALENT //

     THEN BEGIN
          PENSNS(BOKS1 = PLOT(SBOX3)) $,
          PENSNS(BOKS2 = PLOT(SBOX4)) $,
          PENSNS(BOKS3 = PLOT(SBOX5)) $,
          PENSNS(BOKS4 = PLOT(SBOX6)) $,
          DISPLAY(BOKS1,BOKS4) $,
          GET.ITEM(MESSAGE,ITEM,NEED.PENSEE) $,
          IF OBJECT.NAME(ITEM) EQL BOKS1
          THEN BEGIN
               RBOX = BOX3 $,
               IF L(USAGE) NEQ EMPTY OR D(USAGE) NEQ EMPTY
```

```
          THEN GOTO ERR $,
          L(USAGE) = D(USAGE) = FORBIDDEN.LEAD $,
          GOTO ALLBOX $,
          END $,
     IF OBJECT.NAME(ITEM) EQL BOKS2
     THEN BEGIN
          RBOX = BOX4 $,
          IF R(USAGE) NEQ EMPTY OR D(USAGE) NEQ EMPTY
          THEN GOTO ERR $,
          R(USAGE) = D(USAGE) = FORBIDDEN.LEAD $,
          GOTO ALLBOX $,
          END $,
     IF OBJECT.NAME(ITEM) EQL BOKS3
     THEN BEGIN
          RBOX = BOX5 $,
          IF R(USAGE) NEQ EMPTY OR U(USAGE) NEQ EMPTY
          THEN GOTO ERR $,
          R(USAGE) = U(USAGE) = FORBIDDEN.LEAD $,
          GOTO ALLBOX $,
          END $,
     IF OBJECT.NAME(ITEM) EQL BOKS4
     THEN BEGIN
          RBOX = BOX6 $,
          IF U(USAGE) NEQ EMPTY OR L(USAGE) NEQ EMPTY
          THEN GOTO ERR $,
          U(USAGE) = L(USAGE) = FORBIDDEN.LEAD $,
          GOTO ALLBOX $,
          END $,
     GOTO BADSEE $,
     END $,
IF BOX.STYLE EQL 3                    ... BOX HAS 3 LEADS AT 3,9 AND 12 O'CLOCK AND 4 ROTATIONAL
                                       CONFIGURATIONS, ASSUMING ALL LEADS ARE EQUAL //
THEN BEGIN
     PENSNS(BOKS1 = PLOT(SBOX7)) $,
     PENSNS(BOKS2 = PLOT(SBOX8)) $,
     PENSNS(BOKS3 = PLOT(SBOX9)) $,
     PENSNS(BOKS4 = PLOT(SBOX10)) $,
     DISPLAY(BOKS1,BOKS4) $,
     GET.ITEM(MESSAGE,ITEM,NEED.PENSEE) $,
     IF OBJECT.NAME(ITEM) EQL BOKS1
     THEN BEGIN
          RBOX = BOX7 $,
          IF D(USAGE) NEQ EMPTY
          THEN GOTO ERR $,
          D(USAGE) = FORBIDDEN.LEAD $,
          GOTO ALLBOX $,
          END $,
     IF OBJECT.NAME(ITEM) EQL BOKS2
     THEN BEGIN
          RBOX = BOX8 $,
          IF R(USAGE) NEQ EMPTY
          THEN GOTO ERR $,
          R(USAGE) = FORBIDDEN.LEAD $,
          GOTO ALLBOX $,
          END $,
     IF OBJECT.NAME(ITEM) EQL BOKS3
     THEN BEGIN
          RBOX = BOX9 $,
          IF U(USAGE) NEQ EMPTY
          THEN GOTO ERR $,
          U(USAGE) = FORBIDDEN.LEAD $,
```

```
                    GOTO ALLBOX $,
                    END $,
               IF OBJECT.NAME(ITEM) EQL BOKS4
               THEN BEGIN
                    RBOX = BOX10 $,
                    IF L(USAGE) NEQ EMPTY
                    THEN GOTO ERR $,
                    L(USAGE) = FORBIDDEN.LEAD $,
                    GOTO ALLBOX $,
                    END $,
               GOTO BADSEE $,
               END $,
          IF BOX.STYLE EQL 4                    ... BOX HAS 4 LEADS, AND ONLY ONE CONFIGURATION, ASSUMING ALL
                                                    LEADS ARE EQUIVALENT. TO SIMPLIFY THE PROGRAMMING THE PROGRAM
                                                    ASSUMES ANY LEAD CAN BE EQUALLY WELL ATTACHED IN A GIVEN
                                                    DIRECTION, I.E., THERE IS ONLY 1 ORIENTATION FOR A DEVICE WITH 4
                                                    LEADS. THIS IS OBVIOUSLY UNREALISTIC, BUT IT REDUCES THE NUMBER
                                                    OF ORIENTATIONS TO BE CHECKED WHEN A DEVICE IS BEING USED. IN A
                                                    USEFUL SYSTEM EACH LEAD WOULD HAVE A DESIGNATION AND THIS WOULD BE
                                                    USED IN SPECIFYING THE DESIRED ORIENTATION. //
          THEN BEGIN
               CHOICE = CPYOBJ(BOX11) $,
               GOTO RETURN $,
               END $,
BADSEE $  ERROR(3)                              ... USER POINTED TO SOMETHING OTHER THAN THE SAMPLE BOXES $,
          GOTO START $,
ALLBOX $  CHOICE = CPYOBJ(RBOX) $,
          RMV(BOKS1,BOKS4) $,
          GOTO RETURN $,
ERR $     ILLEGAL.ATTACHMENT = TRUE $,
          RMV(BOKS1,BOKS4) $,
          GOTO RETURN $,
          END $,
```

```
DEFINE PROCEDURE TRMNLS(DEFINITION.POINTER) WHERE POINTER DEFINITION.POINTER TOBE
     BEGIN                                    ... TRMNLS DETERMINES WHICH NODES IN THE EQUIVALENT CIRCUIT FOR A
                                               DEVICE ARE TERMINAL NODES AND STORES THIS INFORMATION IN THE
                                               DEVICE DEFINITION BEAD. THE NUMBER OF NODES IS DETERMINED BY THE
                                               BOX STYLE WHICH WILL HAVE BEEN SET BY A PRIOR CALL TO PROCEDURE
                                               BOXES. //
      TRMNL1(DEFINITION.POINTER) = EMPTY       ... SET EACH TERMINAL LEAD POINTER TO BE EMPTY $,
     TRMNL2(DEFINITION.POINTER) = EMPTY $,
     TRMNL3(DEFINITION.POINTER) = EMPTY $,
     TRMNL4(DEFINITION.POINTER) = EMPTY $,
     GET.ITEM(.C. /POINT TO FIRST TERMINAL OF DEVICE/,ITEM,NEED.PENSEE) $,
     TRMNL1(DEFINITION.POINTER) = OBJECT.NAME(ITEM) ... STORE POINTER TO TERMINAL NODE SEEN INTO THE DEVICE
                                               DEFINITION BEAD $,
     GET.ITEM(.C. /POINT TO SECOND TERMINAL OF DEVICE/,ITEM,NEED.PENSEE) $,
     TRMNL2(DEFINITION.POINTER) = OBJECT.NAME(ITEM) $,
     IF STYLE(DEFINITION.POINTER) GRT 2
     THEN BEGIN
          GET.ITEM(.C. /POINT TO THIRD TERMINAL OF DEVICE/,ITEM,NEED.PENSEE) $,
          TRMNL3(DEFINITION.POINTER) = OBJECT.NAME(ITEM) $,
          IF STYLE(DEFINITION.POINTER) GRT 3
          THEN BEGIN
               GET.ITEM(.C. /POINT TO FOURTH TERMINAL OF DEVICE/,ITEM,NEED.PENSEE) $,
               TRMNL4(DEFINITION.POINTER) = OBJECT.NAME(ITEM) $,
               END $,
          END $,
     GOTO RETURN $,
     END $,
```

```
DEFINE PROCEDURE SAMPLE.BOX(SBOX1,SBOX2,SBOX3,SBOX4,SBOX5,SBOX6,SBOX7,SBOX8,SBOX9,SBOX10,SBOX11) WHERE POINTER
 SBOX1,SBOX2,SBOX3,SBOX4,SBOX5,SBOX6,SBOX7,SBOX8,SBOX9,SBOX10,SBOX11 TOBE
     BEGIN                                          ... SAMPLE.BOX CREATES THE DISPLAY OBJECTS USED IN PROCEDURE
                                                     CHOICE. EACH OBJECT IS A BOX AND A POINT TO POSITION THE BOX
                                                     IN THE CORRECT PLACE WITHIN THE SELECTION AREA AT THE BOTTOM OF
                                                     THE SCREEN //

      POINTER PROCEDURE SETPT $,
     DEFOBJ() $,
     ADDOBJ(SETPT(-410,-520))                       ... ADD A  POINT AT X = -410, Y = -520 $,
     ADDOBJ(CPYOBJ(BOX1))                           ... ADD A COPY OF THE BOX TO THE OBJECT BEING CREATED $,
     SBOX1 = ENDOBJ()                               ... TERMINATE THE OBJECT DEFINITION. SBOX1 IS A POINTER TO THE
                                                     OBJECT, WHICH CAN BE ADDED TO THE DISPLAY FILE BY CALLING
                                                     PROCEDURE PLOT. $,

     DEFOBJ() $,
     ADDOBJ(SETPT(-340,-520)) $,
     ADDOBJ(CPYOBJ(BOX2)) $,
     SBOX2 = ENDOBJ() $,
     DEFOBJ() $,
     ADDOBJ(SETPT(-250,-520)) $,
     ADDOBJ(CPYOBJ(BOX3)) $,
     SBOX3 = ENDOBJ() $,
     DEFOBJ() $,
     ADDOBJ(SETPT(-180,-520)) $,
     ADDOBJ(CPYOBJ(BOX4)) $,
     SBOX4 = ENDOBJ() $,
     DEFOBJ() $,
     ADDOBJ(SETPT(-110,-520)) $,
     ADDOBJ(CPYOBJ(BOX5)) $,
     SBOX5 = ENDOBJ() $,
     DEFOBJ() $,
     ADDOBJ(SETPT(-40,-520)) $,
     ADDOBJ(CPYOBJ(BOX6)) $,
     SBOX6 = ENDOBJ() $,
     DEFOBJ() $,
     ADDOBJ(SETPT(50,-520)) $,
     ADDOBJ(CPYOBJ(BOX7)) $,
     SBOX7 = ENDOBJ() $,
     DEFOBJ() $,
     ADDOBJ(SETPT(120,-520)) $,
     ADDOBJ(CPYOBJ(BOX8)) $,
     SBOX8 = ENDOBJ() $,
     DEFOBJ() $,
     ADDOBJ(SETPT(190,-520)) $,
     ADDOBJ(CPYOBJ(BOX9)) $,
     SBOX9 = ENDOBJ() $,
     DEFOBJ() $,
     ADDOBJ(SETPT(260,-520)) $,
     ADDOBJ(CPYOBJ(BOX10)) $,
     SBOX10 = ENDOBJ() $,
     DEFOBJ() $,
     ADDOBJ(SETPT(350,-520)) $,
     ADDOBJ(CPYOBJ(BOX11)) $,
     SBOX11 = ENDOBJ() $,
     END $,

END FINI
```

```
DEFINE PROCEDURE MAKEBOX(BOX1,BOX2,BOX3,BOX4,BOX5,BOX6,BOX7,BOX8,BOX9,BOX10,BOX11) WHERE POINTER BOX1,BOX2,BOX3
 ,BOX4,BOX5,BOX6,BOX7,BOX8,BOX9,BOX10,BOX11 TOBE
                                        ... MAKEBOX DEFINES THE DISPLAY OBJECTS TO PLOT ALL OF THE
                                         SYMBOLS USED TO REPRESENT DEVICES //
     BEGIN
    POINTER BOX $,
    POINTER PROCEDURE LIN,INVIS $,
    DEFOBJ() $,
    ADDOBJ(INVIS(LIN(10,10))) $,
    ADDOBJ(LIN(0,-20)) $,
    ADDOBJ(LIN(-20,0)) $,
    ADDOBJ(LIN(0,20)) $,
    ADDOBJ(LIN(20,0)) $,
    BOX = ENDOBJ()                      ... DISPLAY OBJECT WITHOUT LEADS,NAME = BOX $,
    DEFOBJ()                            ... BOX WITH 2 LEADS AT POSITIONS DENOTED AS 3 AND 9 O'CLOCK $,
    ADDOBJ(CPYOBJ(BOX)) $,              ... ADD A COPY OF BOX TO COMPOSE A NEW OBJECT WITH LEADS $,
    ADDOBJ(INVIS(LIN(0,-10))) $,
    ADDOBJ(LIN(20,0)) $,
    ADDOBJ(INVIS(LIN(-40,0))) $,
    ADDOBJ(LIN(-20,0)) $,
    ADDOBJ(INVIS(LIN(30,30))) $,        ... REPOSITION BEAM INVISIBLY TO GIVEN POSITION RELATIVE TO THE
                                         BEAM POSITION AT THE BEGINNING OF THE OBJECT //
     BOX1 = ENDOBJ() $,
    DEFOBJ()                            ... BOX WITH 2 LEADS AT 12 AND 6 O'CLOCK $,
    ADDOBJ(CPYOBJ(BOX)) $,
    ADDOBJ(INVIS(LIN(-10,0))) $,
    ADDOBJ(LIN(0,20)) $,
    ADDOBJ(INVIS(LIN(0,-40))) $,
    ADDOBJ(LIN(0,-20)) $,
    ADDOBJ(INVIS(LIN(0,60))) $,
    BOX2 = ENDOBJ() $,
    DEFOBJ()                            ... BOX WITH 2 LEADS AT 12 AND 3 O'CLOCK $,
    ADDOBJ(CPYOBJ(BOX)) $,
    ADDOBJ(INVIS(LIN(20,-10))) $,
    ADDOBJ(LIN(-20,0)) $,
    ADDOBJ(INVIS(LIN(-10,10))) $,
    ADDOBJ(LIN(0,20)) $,
    ADDOBJ(INVIS(LIN(-30,30))) $,
    BOX3 = ENDOBJ() $,
    DEFOBJ()                            ... BOX WITH 2 LEADS AT 12 AND 9 O'CLOCK $,
    ADDOBJ(CPYOBJ(BOX)) $,
    ADDOBJ(INVIS(LIN(-20,-10))) $,
    ADDOBJ(LIN(-20,0)) $,
    ADDOBJ(INVIS(LIN(30,10))) $,
    ADDOBJ(LIN(0,20)) $,
    BOX4 = ENDOBJ() $,
    DEFOBJ()                            ... BOX WITH 2 LEADS AT 6 AND 9 O'CLOCK $,
    ADDOBJ(CPYOBJ(BOX)) $,
    ADDOBJ(INVIS(LIN(-20,-10))) $,
    ADDOBJ(LIN(-20,0)) $,
    ADDOBJ(INVIS(LIN(30,-10))) $,
    ADDOBJ(LIN(0,-20)) $,
    ADDOBJ(INVIS(LIN(0,60))) $,
    BOX5 = ENDOBJ() $,
    DEFOBJ()                            ... BOX WITH 2 LEADS AT 6 AND 3 O'CLOCK $,
    ADDOBJ(CPYOBJ(BOX)) $,
    ADDOBJ(INVIS(LIN(0,-10))) $,
    ADDOBJ(LIN(20,0)) $,
    ADDOBJ(INVIS(LIN(-30,-10))) $,
    ADDOBJ(LIN(0,-20)) $,
    ADDOBJ(INVIS(LIN(0,60))) $,
```

```
BOX6 = ENDOBJ() $,
DEFOBJ()                                    ... BOX WITH 3 LEADS AT 3,9 AND 12 O'CLOCK $,
ADDOBJ(CPYOBJ(BOX)) $,
ADDOBJ(INVIS(LIN(0,-10))) $,
ADDOBJ(LIN(20,0)) $,
ADDOBJ(INVIS(LIN(-40,0))) $,
ADDOBJ(LIN(-20,0)) $,
ADDOBJ(INVIS(LIN(30,10))) $,
ADDOBJ(LIN(0,20)) $,
BOX7 = ENDOBJ() $,
DEFOBJ()                                    ... BOX WITH 3 LEADS AT 6,9 AND 12 O'CLOCK $,
ADDOBJ(CPYOBJ(BOX)) $,
ADDOBJ(INVIS(LIN(-10,0))) $,
ADDOBJ(LIN(0,20)) $,
ADDOBJ(INVIS(LIN(0,-40))) $,
ADDOBJ(LIN(0,-20)) $,
ADDOBJ(INVIS(LIN(-10,30))) $,
ADDOBJ(LIN(-20,0)) $,
ADDOBJ(INVIS(LIN(30,30))) $,
BOX8 = ENDOBJ() $,
DEFOBJ()                                    ... BOX WITH 3 LEADS AT 3,9 AND 6 O'CLOCK $,
ADDOBJ(CPYOBJ(BOX)) $,
ADDOBJ(INVIS(LIN(0,-10))) $,
ADDOBJ(LIN(20,0)) $,
ADDOBJ(INVIS(LIN(-40,0))) $,
ADDOBJ(LIN(-20,0)) $,
ADDOBJ(INVIS(LIN(30,-10))) $,
ADDOBJ(LIN(0,-20)) $,
ADDOBJ(INVIS(LIN(0,60))) $,
BOX9 = ENDOBJ() $,
DEFOBJ()                                    ... BOX WITH 3 LEADS AT 3,6 AND 12 O'CLOCK $,
ADDOBJ(CPYOBJ(BOX)) $,
ADDOBJ(INVIS(LIN(-10,0))) $,
ADDOBJ(LIN(0,20)) $,
ADDOBJ(INVIS(LIN(0,-40))) $,
ADDOBJ(LIN(0,-20)) $,
ADDOBJ(INVIS(LIN(10,30))) $,
ADDOBJ(LIN(20,0)) $,
ADDOBJ(INVIS(LIN(-30,30))) $,
BOX10 = ENDOBJ() $,
DEFOBJ()                                    ... BOX WITH 4 LEADS $,
ADDOBJ(CPYOBJ(BOX)) $,
ADDOBJ(INVIS(LIN(-10,0))) $,
ADDOBJ(LIN(0,20)) $,
ADDOBJ(INVIS(LIN(0,-40))) $,
ADDOBJ(LIN(0,-20)) $,
ADDOBJ(INVIS(LIN(10,30))) $,
ADDOBJ(LIN(20,0)) $,
ADDOBJ(INVIS(LIN(-40,0))) $,
ADDOBJ(LIN(-20,0)) $,
ADDOBJ(INVIS(LIN(30,30))) $,
BOX11 = ENDOBJ() $,
END $,

END FINI
```

CS-TR Scanning Project
Document Control Form

Date : 2/2/96

Report # LCS-TR-63

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 94 (99-IMAGES)
Not to include DOD forms, printer instructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☒ Unknown ☐ Other:_____

Check each if included with document:

☒ DOD Form ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages (by page number): FOLLOWS PAGE #7

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:

(A) IMAGE MAP: (1-94) 1-7, UNNUMBERED BLANK, 9-94 (95-99) SCANCONTROL, DOD, TARGETS (?)

(B) NO TITLE PAGE.

Scanning Agent Signoff:

Date Received: 2/2/96 Date Scanned: 2/2/96 Date Returned: 2/9/96

Scanning Agent Signature: Michael W. Cook

Project MAC — Technical Report Abstract

1. ORIGINATING ACTIVITY	2a. REPORT SECURITY CLASSIFICATION
Massachusetts Institute of Technology Project MAC	UNCLASSIFIED

3. REPORT TITLE

Case Study in Interactive Graphics Programming: A Circuit Drawing and Editing Program for use with a Storage-Tube Display Terminal

4. DESCRIPTIVE NOTES

Technical Report

5. AUTHOR(S)

John W. Brackett, Michael Hammer, Daniel E. Thornhill

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
October 1969	94	10

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER
Office of Naval Research, Nonr-4102(01) b. PROJECT NO. NR-048-189 c. RR 003-09-01	MAC TR-63 9b. OTHER REPORT NO. AD 699-930

10. AVAILABILITY/LIMITATION NOTICES Defense Contractors may obtain from: Defense Documentation Center, Document Service Center, Cameron Station, Alexandria, VA 22314

Others from: Clearinghouse for Federal Scientific and Technical Information (CFSTI) Sills Building, 5285 Port Royal Road, Springfield, VA 22151

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	Advanced Research Projects Agency 3D-200 Pentagon Washington, D. C. 20301

13. ABSTRACT

The concepts involved in building and manipulating a data structure through graphical interaction are presented, using the drawing and editing of electrical circuits as a vehicle. The circuit drawing program was designed to operate on an ARDS storage-tube display terminal attached to the M.I.T. Project MAC IBM 7094 Compatible Time-Sharing System. The graphics software system (GRAPHSYS) developed by the M.I.T. Computer-Aided Design Project was used for dealing with all graphical input and output, and the AED Language of the Project was used in programming. AED System packages for building and manipulating complex data structures are described and their use is illustrated in detail. The report includes flow diagrams and complete listings of the sample circuit drawing and editing system.

14. KEY WORDS

Computer Graphics
Graphic Terminals
Display Software System
On-Line Computers
Computer-Aided Design
Time-Shared Computers
AED Software

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 2/2/1996

M.I.T. Libraries
Document Services