December, 1968

Report Nos. ESL-R-356
MAC-TR-56

AN INTEGRATED HARDWARE-SOFTWARE SYSTEM FOR COMPUTER GRAPHICS IN TIME-SHARING

by

D. E. Thornhill
R. H. Stotz
D. T. Ross
J. E. Ward

Electronic Systems Laboratory
Department of Electrical Engineering

Project MAC
545 Technology Square

Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

ABSTRACT

This report describes the ESL Display Console and its associated user-oriented software systems developed by the M.I.T. Computer-Aided Design Project with Project MAC. Console facilities include hardware projection of three-dimensional line drawings, automatic light pen tracking, and a flexible set of knob, switch, and push-button inputs. The console is attached to the Project MAC IBM 7094 Compatible Time-Sharing System either directly or through a PDP-7 Computer. Programs of the Display Controller software provide the real-time actions essential to running the display, and communication with the time-sharing supervisor. A companion graphics software system (GRAPHSYS) provides a convenient, high-level, and nearly display-independent interface between the user and the Display Controller. GRAPHSYS procedures allow the user to work with element "picture parts" as well as "subpictures" to which "names" are assigned for identification between user and Controller programs. Software is written mostly in the machine-independent AED-0 Language of the Project and many of the techniques described are applicable in other contexts.

ACKNOWLEDGEMENT

The display hardware and software described in this document has been previously described at various states of development over the past four years in a number of informal memoranda and papers by the authors and by Charles A. Lang, Richard U. Bayles, and John W. Brackett. The present document combines this previous material into a single reference manual, with revision and rewriting where necessary to reflect the present status of the hardware-software system. The contributions of Charles Garman, Harold D. Levin, and Michael F. Brescia, who assisted with parts of the implementation, are gratefully acknowledged.

The respective roles of the two M.I.T. projects which provided support to the Display Group for this work from different contracts are difficult to delineate exactly. The CTSS computer time was entirely provided by Project MAC (ARPA support), whereas the AED facilities used in programming were intimately associated with the work of the Computer-Aided Design Group of the Electronic Systems Laboratory (U.S. Air Force support). Support for hardware acquisitions was about evenly divided between the two, as was support for personnel. This joint support and sharing of resources has created an unusual research environment which has done much to insure the success of these endeavors. It is in this spirit that this document is offered for joint publication.

PREFACE

"On-line computer graphics" and "time-sharing" are fast becoming more than mere catch phrases for exciting and esoteric laboratory curiosities. On-line computer graphics in time-sharing is now passing from the laboratory into the day-to-day production environment, bringing with it new and powerful techniques for coupling men and computers into problem-solving teams with capabilities far exceeding those of either men or machines alone. This report describes the completion of the pioneering activities of the Computer Applications Group and the Display Group of the M.I.T. Electronic Systems Laboratory in the development of the general-purpose ESL Display Console and its supporting user-oriented Software Systems. The work was performed as part of the activities of the M.I.T. Computer-Aided Design Project, sponsored by the Manufacturing Technology Laboratory of the United States Air Force, and later, Project MAC, sponsored through the Office of Naval Research by the Advanced Research Projects Agency. In addition to serving as a complete manual for the now essentially stabilized hardware-software system at M.I.T., this report will be useful to others who do not have direct access to the M.I.T. systems by providing a complete and comprehensive in-depth treatment of a successful overall design of a computer graphics system.

Although the chapters of this report provide details and complete coverage of certain aspects, only a small portion of the overall story is told here. Computer graphics is potentially much more than mere picture making, but in order for this potential to be realized, the subjects covered in this report must be augmented by many additional features so that the mechanical aspects of computer graphics are subsumed in the broader context of true graphical language, integrally incorporated into a problem-solving system. Although many of the features described in this report have been heavily influenced by this broader context, and in some places the mode of discussion provides hints concerning our view of these questions, this report is primarily concerned with providing facilities for a certain class of generalized interactive display facilities and interfacing those facilities into the programming environment. The broader questions of making true graphical languages and incorporating these display facilities into a problem-solving environment are not covered here.

The chapters of this report reflect three major problem areas progressing from (1) the real world of physical display generation and real-time on-line control through (2) the time-sharing environment to (3) the interface to the user's program environment. The material of the various chapter sections has appeared in internal memorandum form in various editions over the years of dynamic change of the total system, and sometimes (like the Bible) the historical diversity of the original sources shows in variations in manner of description. For example, the display console was originally directly connected to the Compatible Time-Sharing System on the IBM 7094 computer, with the Display Controller software an integral part of the time-sharing supervisor. Later, a PDP-7 computer was interposed between the display console and the 7094, and many of the Display Controller features were transferred to the PDP-7. Since the operating characteristics of the programs of the Display Controller remain the same, the description presented in Chapter II is oriented toward the original system and does not go into details about the actual system presently in use.

We have learned a great deal in the evolution of the initial system to its current state. Although much of this learning is reflected in various features of the system, a larger number of insights are of a more basic nature and cannot be incorporated into evolutionary changes, which by their nature include design features of ancient origin. Therefore, concurrently with the final stabilization of the initial system as described in this report, we have been carrying out various new investigations to provide a fresh start for a still more penetrating attack on the fundamental requirements for a next-generation system. Various portions of this new research are now completed and others are being written or designed. Although the flavor of the new system will be much the same as that of the system described in this report, the internal workings will be different in many fundamental respects in order to achieve a degree of independence of display characteristics, computer characteristics, time-sharing environment characteristics, and problem application area which is impossible within the framework of the original system design. Hopefully this new attack will result in a total system which will provide a stable interface between graphic language application systems and the physical devices on which such systems are implemented. Then heavy investments may be made in software developments with confidence that drastic hardware changes may be accommodated with only routine bootstrapping of the software. It is our firm opinion that the fundamentals of interfacing user applications with physical display and computing hardware systems must have this degree of stability before the potential of on-line man-machine problem-solving with mixed verbal and graphical languages can be fully realized. Our experience to date indicates that although it is a difficult and challenging assignment, these goals are eminently realizable.

A key ingredient in both the old and the new systems in achieving problem-, display-, operating system-, and computer-independence is in the use of the AED (Automated Engineering Design or Algol Extended for Design) Approach and the AED Systems developed by the M.I.T. Computer-Aided Design Project. The generalized software development techniques of the AED Approach and the many unique, efficient, and powerful features of the AED-0 Language have played a central role throughout these developments and will be increasingly important in our newer work. Although the subroutine calling conventions of AED are compatible with other languages, so that the display interface system may be called from various foreign environments, AED is strongest in the problem-structuring areas where other languages are inadequate, and those areas are of great importance in successful graphics applications. As AED becomes available on more and more types of computers, these advanced graphics systems will also be available to a growing class of users. Although AED is completely general-purpose and is not restricted to graphics applications, we anticipate that graphics will be an important component in furthering wider adoption by industry of the total spectrum of AED techniques and facilities.

Douglas T. Ross

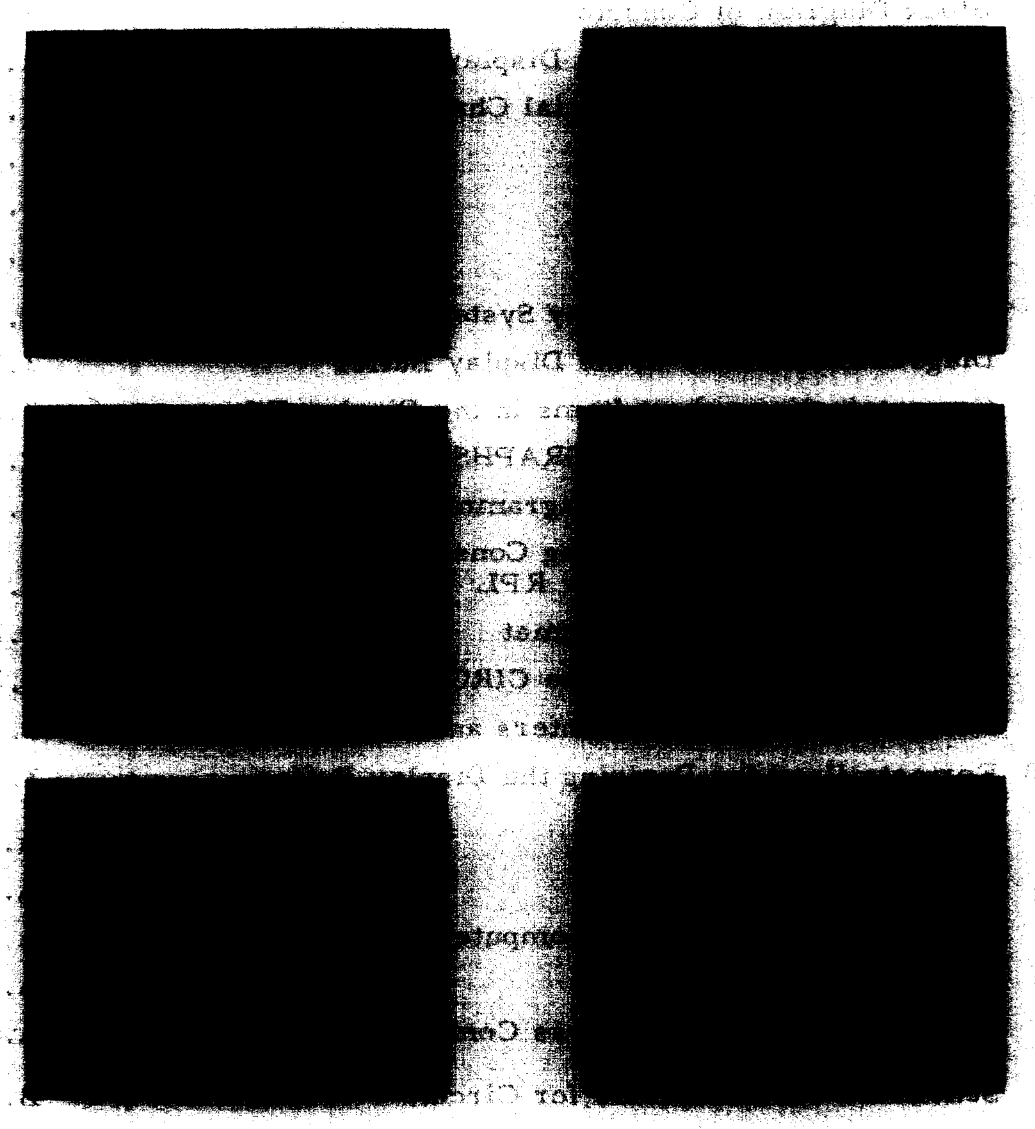
CONTENTS

CONTENTS (Contd.)

CONTENTS (Contd.)

CONTENTS (Contd.)

LIST OF FIGURES



The ESL Console is a specialized computer which automatically converts three-dimensional drawing commands into arbitrary two-dimensional projections. Real-time rotation, translation, and scale change are possible even in time-sharing. Light-pen tracking is fully automatic, and either picture elements or character information may be displayed.

CHAPTER I

OPERATING MANUAL FOR THE ESL DISPLAY CONSOLE

A. INTRODUCTION

The Electronic Systems Laboratory Display Console, shown in Fig. 1.1, was designed and built in 1963. The purpose of the unit is to provide a direct, fast, computer-controlled display plus a flexible set of input devices including a light pen. It was designed with special attention to the needs of Computer Aided Design under the restriction of a time-sharing system, but its flexibility makes it a useful tool for many other applications as well. The Console is an outgrowth of the Manual Intervention System which was connected to the Co-operative Computing Laboratory's IBM 709 computer for several years.

The command format of the Display is based on a 36-bit computer word because of its original connection to the IBM 7094. The PDP-7 provides the 36-bit words as two sequential 18-bit words.

The basic interface between the Display and the Display Controller computer meets the specifications of the Direct Data Interface of the 7094 as described in the IBM special features bulletin for RPQ M90976. When the display was connected directly to the 7094, it was via this Interface to the Direct Data Channel. A single, specially built channel for the PDP-7 matches the IBM specifications. This channel connects the IBM Channel to the PDP-7 and the PDP-7 to the Display. The result of this connection is that to the display the PDP-7 looks like a 7094, and to the 7094 it looks like the terminal side of the Direct Data Interface.

Because the Console is a research tool, it will be subject to modification and change. As these modifications are made, programming details may change. Users of the Console are advised to ensure that they are up to date with the latest revisions of this manual.

B. SYSTEM DESCRIPTION

1. The Console Hardware

The Display Console is shown in block diagram form in Fig. 1.2. A portion of the Console is a Type 330 Incrementing Display, made by the Digital Equipment Corporation of Maynard, Mass. to M.I.T.

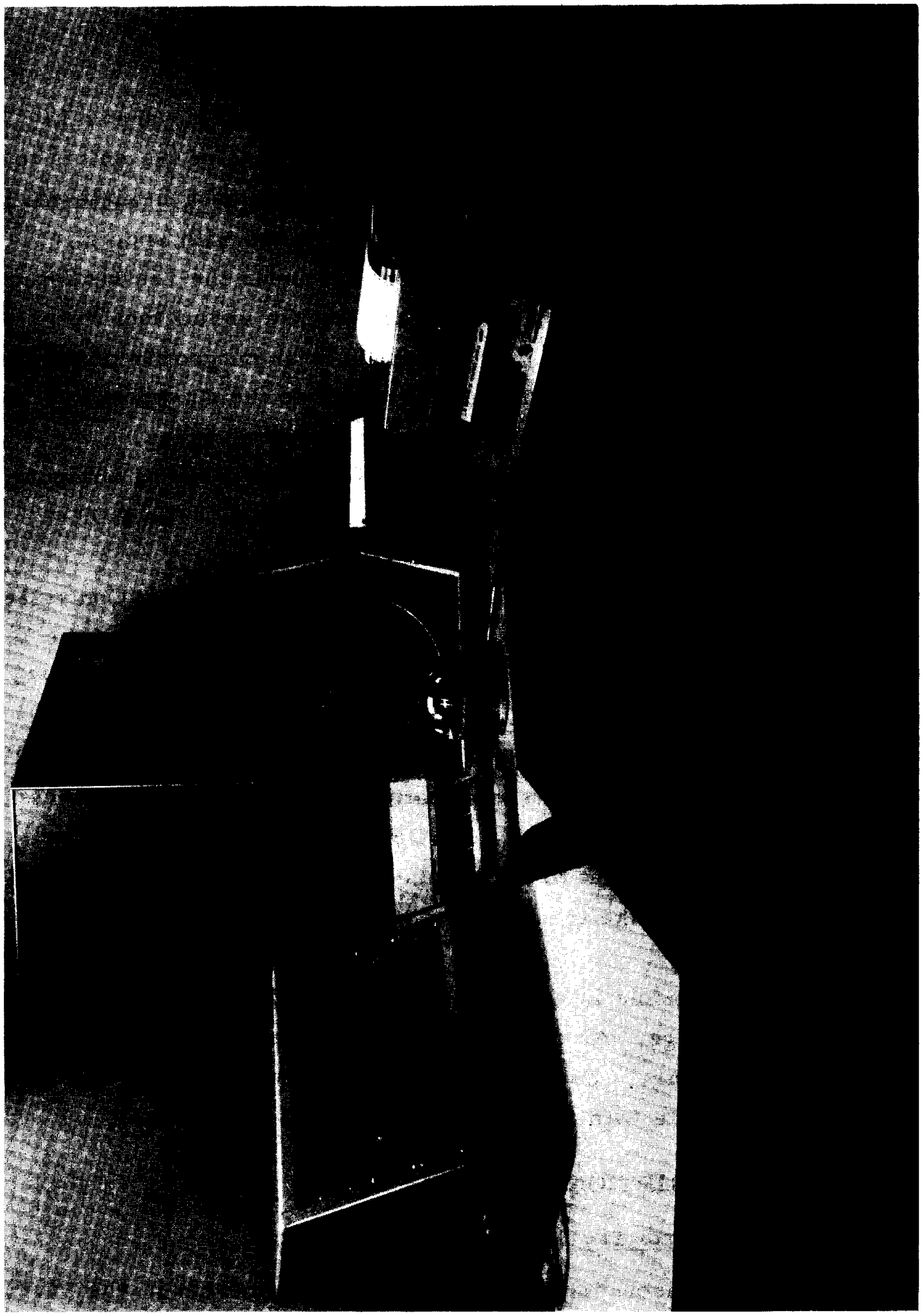


Fig. 1.1 ESL Display Console

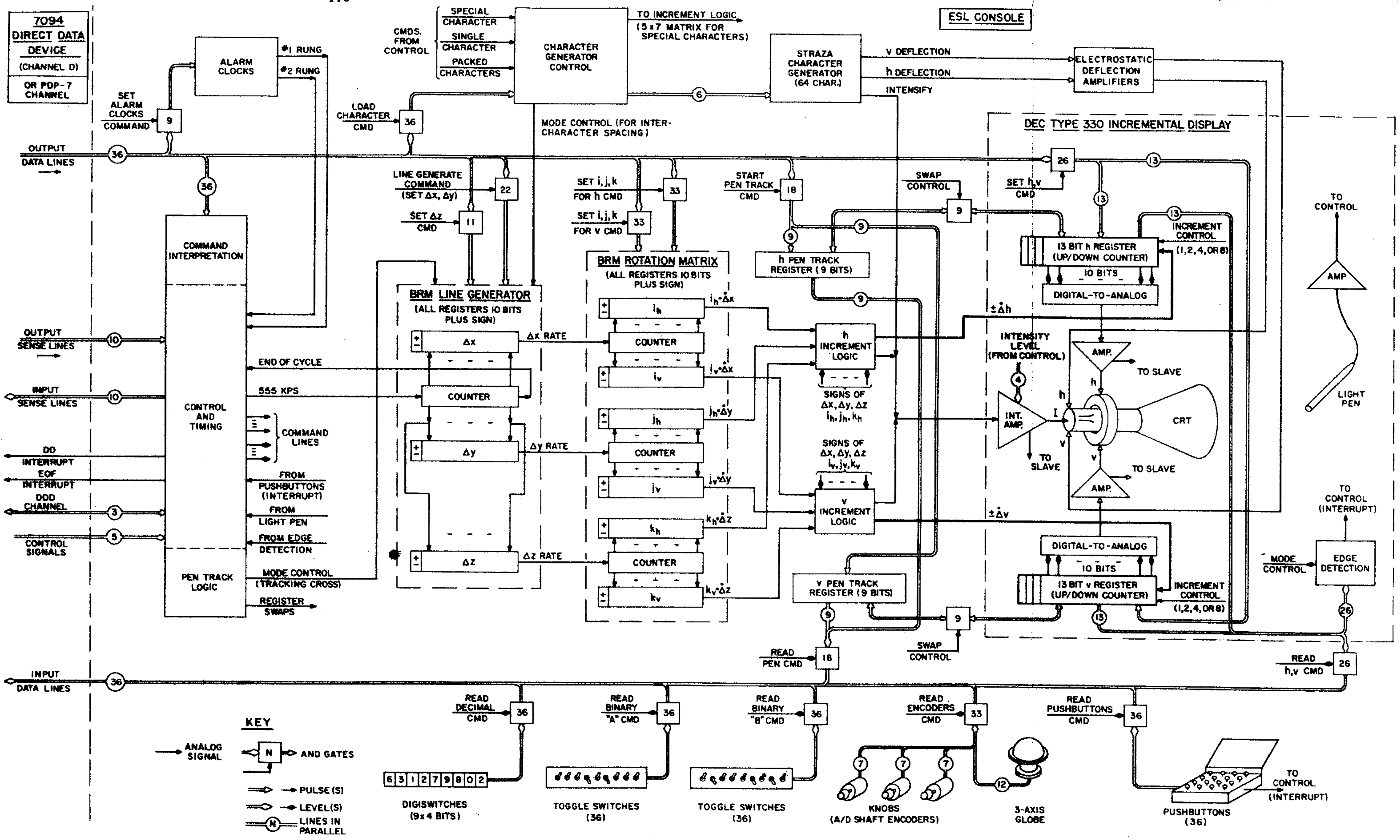


Fig. 1.2 Block Diagram of Console

specifications. This consists of the magnetic-deflection cathode-ray tube with housing, table, power, and deflection amplifiers; the digital-to-analog converters; the digital registers which contain the h and v coordinates of the beam position; and a number of special controls. The scope has 1,024 unique horizontal positions and a similar number of vertical positions, thus providing over one million discrete points that can be specified. It has an active surface 9 3/8 inches on a side.

A unique feature first used in the Type 330 is the ability of the beam position registers (called the h and v registers, for horizontal and vertical position) to count up and down at high speed. This permits lines to be drawn by introducing strings of pulses into the h and v registers and intensifying (unblanking) after each pulse. The current plotting rate in the incrementing mode is 1.8 μsec per point.

Another unusual feature ordered by M.I.T. in the Type 330 scope is three extra bits on the high-order end of the h and v registers (for a total of 13 bits). This allows h and v to be incremented off the edge of the screen without having the line appear coming on at the opposite edge (wrap around). The active (visible) scope surface is described by 2^{10} horizontal and a similar number of vertical positions. This area is called the <u>Scope Field.</u> The three extra bits allow the console to process lines on a field with sides eight times as large as the Scope Field. This larger field is referred to as the <u>Total Console Field</u>. There are programmable interrupts available to alert the computer when the edge of the Scope Field has been crossed or when the edge of the large Total Console Field has been crossed. These interrupts are described in detail in a later section.

The center of the scope has the binary address 0 000 000 000 000 for h and v. The right hand edge of the Scope Field has the horizontal value 0 000 111 111 111 ($= 2^9 - 1$) while the left hand edge is 1 111 000 000 000. Similarly, the upper edge of the visible section has the vertical value 0 000 111 111 111 while the lower edge is the ones complement of this.

The portion of the Display Console built by ESL contains the driving logic for the Type 330, the interface to the 7094 Direct Data Channel (or PDP-7 channel), a section to interpret commands from the Display Controller and controls to perform the function called for by the command. Also, a scheme for automatic picture rotation in three dimensions has been built into the Console. Therefore, two sets of coordinate axes are referred to. The first set is the axes of the scope itself which are identified as "horizontal (h), vertical (v), and depth (d)." The second set is the axes in which a line is specified by the computer, i.e., before it is rotated to the h, v, and d axes. Coordinates in these axes are referred to as "x, y, and z".

The present line generator consists of three Binary Rate Multipliers (BRM), which produce three pulse trains with rates proportional to the Δx, Δy, and Δz values of Line Generate commands. These pulses are processed through a rotation Matrix. Here the Δx rate is the input to a BRM pair which generates two new pulse trains $\dot{\Delta} x \cdot i_h$ and $\dot{\Delta} x \cdot i_v$. The Δy pulses produce a similar pulse train pair $\dot{\Delta} y \cdot j_h$ and $\dot{\Delta} y \cdot j_v$, and Δz generates $\dot{\Delta} z \cdot k_n$ and $\dot{\Delta} z \cdot k_v$. The $\dot{\Delta} x \cdot i_h$, $\dot{\Delta} y \cdot j_h$, and $\dot{\Delta} z \cdot k_h$ pulse trains are combined in the increment logic block to be the h register input pulse train. Similar logic creates the v register input pulse train. The values of i_h, i_v, j_h, etc., are loaded by the computer, and by suitably choosing their values all the lines produced by x, y, and z may be transformed into a rotated axonometric projection of themselves. Since lines are called out incrementally, independent of starting point, an entire picture built up from a connected series of these lines is subject to whatever rotation the computer calls for. Also moving the starting h, v location for a picture made up of connected lines moves the entire picture. The rotation matrix is used to change the size of displays by applying a common scale factor to the i_h, i_v, j_h, etc.

Pen tracking in the Console is completely automated; that is, except for initiating or halting tracking, the entire operation of generating the cross, computing the new center, and repositioning the cross, is done in the hardware. A pen-track cross is generated once every five milliseconds and requires about 200 microseconds to complete. When it is time to generate the cross, the command logic prevents acceptance of the next word from the Data Channel and swaps the contents (ten least significant bits) of the h and v registers with the Pen Track Registers. This saves the current display location, and puts the old pen position in the h and v registers. A block of logic associated with the line generator then causes it to produce the tracking cross, store the information as to which points the light pen has seen (producing error vector components), and add these components to the h and v registers to update the pen position. The registers then swap back again, and the command logic is allowed to proceed. The pen-track logic is unaffected by the rotation matrix. The Display Controller may read the Pen Track Registers at any time except during a tracking cycle, although reading at the end of a display frame is recommended.

Characters are generated by either of two separate systems. A Straza Symbol Generator, which is a separate unit, interprets six-bit character codes to select one of 64 stored characters which match those of the KSR-35 Teletype. Diodes wired on the character cards specify a sequence of incremental deflections for up to 16 points in a 15 by 16 matrix. The Straza unit produces analog deflection signals for extra h and v high-speed "diddle plates" on the CRT.

The Special Character Generator, constructed by ESL, allows programmed symbols. Special logic causes it to step the scope beam through a 5 by 7 raster, and the intensification of each of the 35 points of the raster is controlled by a corresponding bit in a word from the computer. Thus, a single Special Character requires a full word to specify it. In either character generator, four character sizes are available, and space between characters (which is controlled by the line generator), is entirely programmable in size and direction. Also, both character generators by-pass the rotation matrix, i.e., characters always remain upright.

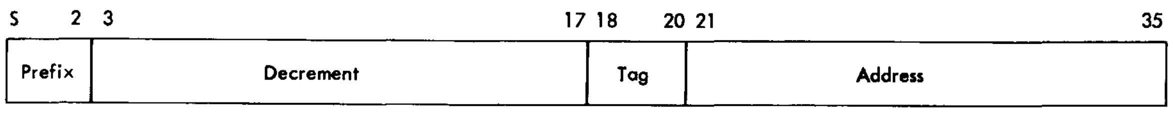
A computer controlled movie camera was added to the display in 1965. The camera is made by D. B. Milliken Co., Arcadia, California. The computer controls frame advance and shutter, but it can not back up the film.

Two alarm clocks are provided for display timing. These are set by program to ring (i.e., cause a Channel Interrupt) at any desired interval from 50 microseconds to 128 milliseconds. Other sources of Interrupt are: a push button press; the light pen seeing an active line, point, or character; and an "edge detect" (a line crossing the edge of the display field or the console field). The "pen see" or "edge detect" interrupts can be individually inhibited by program.

Input facilities provided at each console station are a CTSS Teletype, a light pen, a bank of nine decimal switches (Digi-switches), two banks of 36 toggle switches, 36 push buttons, three 7-bit shaft encoders and a 3-dimensional rate-control joy stick (crystal ball). The computer can read the settings of these switches and controls whenever it pleases. There is no hardware relation between any of these controls and any Console output function. They are entirely interpreted by the computer program. For example, although usually the crystal ball (3-D joy stick) is most effective for controlling rotation, the computer can interpret it any way the programmer desires.

2. Mode of Operation from the Computer

The Display Console interprets parallel 36-bit words from the Data Channel as commands. The Prefix Field of the word (sign bit and bits 1 and 2) categorizes the type of command. The Tag Field (bits 18, 19, and 20) further specifies certain commands. The Address and Decrement Fields, and for certain commands the Tag Field, contain the particular parameters for the command.



Section D contains a detailed description of the action taken upon each command; Tables 1.1 and 1.2 in Section G summarize the command format.

Commands to be transmitted to the Console are arranged in the computer memory in groups of consecutive locations called display lists. Since the Data Channel can directly access words stored in this manner, the only time the computer's attention is required is when a new list must be started (or an old list started over). Each command "steals" memory cycles from the computer (one cycle from the 7094 or two from the PDP-7).

When the Display Console accepts an output word from the Data Channel it responds with a Direct Data Demand (DDD) pulse, interprets the command and commences performing the operation called for (e.g., drawing a specified straight line). The DDD pulse releases the Data

Channel to fetch the next output word pair, i.e., the next command. The Display Console will ignore this new command until it has completed the previous one and the Data Channel will hang up waiting for the DDD pulse. When one command is completed, the Console immediately processes the next awaiting command, etc. Each output word from the Data Channel is looked upon as a new command by the Console except that the Packed and special character commands put the Console in a special mode whereby it processes successive output words as characters. This mode of operation, which requires an escape provision, is discussed in detail in the section on Character Generation, (Section D.9).

The Console also can input to the Data Channel 36-bit words which can be the contents of an internal register, or the settings of toggle switches etc. There are ten Output Sense Lines provided by the Data Channel which are used for selection of these inputs, and also control other Console functions. The Channel Interrupt Bit and the ten Input Sense Lines allow the Console to signal special conditions to the computer. In general the Input Sense Lines identify the source of the interrupt. Table 1.3 in Section G depicts the Sense Line bit assignments. Their use is amplified in Section E.

3. Second Operator's Station

A second operator's station was added to the system in 1964. This unit (called the "Slave" unit) contains a duplicate set of manual inputs and a scope which is driven in parallel with the "Master" cathode-ray tube, but which has a separate intensification control. Depending on the state of a pair of control flip-flops which are set by a control command, a picture being generated appears on either the Master, the Slave, both, or neither. Although called Master and Slave, the stations are in fact equivalent in capability, and two operators can work with different displays by time-sharing the console display generating system.

C. SPECIFICATIONS AND PERFORMANCE

1. Computer Interface

Matches IBM Direct Data Channel
36 data bits In

36 data bits Out

10 sense lines In (direct to CPU)

10 sense lines Out (direct from CPU)

1 Channel Interrupt (or Direct Data Interrupt)

1 End of File Interrupt

8 Control Signal Lines

DEC logic levels (0 and -3 volts) out and in. Requires DEC/7094 level conversion unit (see ESL Memo 9474-M-1), if connected to the 7090 Direct Data Channel.

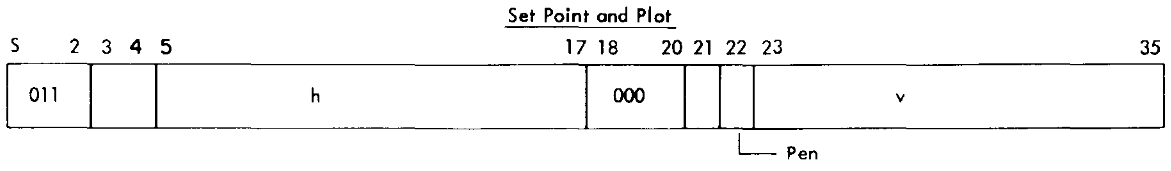
2. Internal Specifications

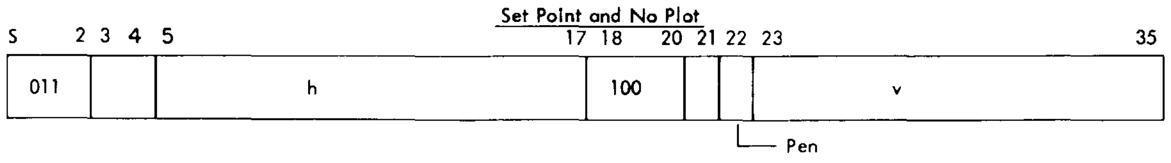
Input Power	115 ± 10 volts, 60 cycles, single phase at 25 amps.
Active Scope Size	9 3/8 inches by 9 3/8 inches containing 1024 points by 1024 points.
Memory	None (operates from display lists stored in the computer memory, and accessed through the Data Channel Connection).
Clock Rate	Normal 555.55 KC (1.8 μsec between clocks). Slow 69.44 KC (14.4 μsec between clocks).
Line Plotting	
Line Plotting Rate	a point each clock (1.8μs). This point can be a step of 0, 1, 2, 4, or 8 scope increments in ±h and in ±v (0, 1, 2 for lines to be rotated).
Line Length	0 to 1023 increments in ±x and in ±y without magnification. 0 - 2046 increments by steps of 2 in ±x and ±y with magnification. Thus, Δx and Δy require 10 bits plus sign each.
Random Point Plotting	
Point Plotting Rate	a point every 40 microseconds
Point Plotting Range	2^{13}=8192 horizontal and vertical positions. Of this only 2^{10}=1024 will appear on the scope. h=0, v=0 is center of screen.

Straza Symbol Generator	
Symbol Code	6 bits to produce one of 64 symbols matching KSR-35 Teletype.
Symbol Size	0.10, 0.14, 0.20, or 0.28 inches high.
Symbol Plotting Rate	a character every 12 μsec.
	Intercharacter spacing is done with the line generator and requires an additional nine μsec.
Special Character Generator	
Symbol Code	35-bit code to produce any symbol on a 5 x 7 dot matrix.
Symbol Size	0.14, 0.28, or 0.56 inches high.
Symbol Plotting Rate	a character every 72 μsec.
Alarm Clocks	Programmable real time interrupt between 50 microseconds and 6.4 milliseconds by 50 microsecond increments (fast clock), or between 1 millisecond and 128 milliseconds by 1 millisecond increments (slow clock).

D. COMMANDS

1. Point Plotting



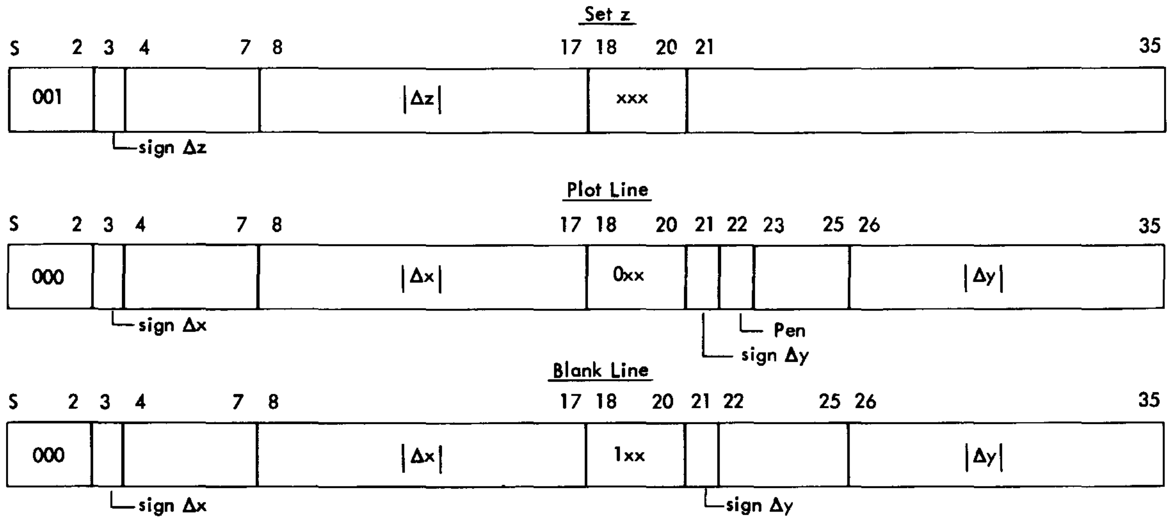


Individual points can be specified and plotted in a manner identical to the DEC PDP Type 30 Scope. This is done by setting h and v (the registers which directly drive the position coils on the CRT) and calling for plotting (IBM bit 18=0). The electron beam can be positioned without plotting by giving the Set Point command with IBM bit 18=1.

The beam can be positioned to any spot in the Total Console Field. As seen in Table 1.2 (Section G), h is specified by the least significant bits of the decrement portion and v by the least significant bits of the address portion of the 36-bit command. Both h and v are specified as one's complement values.

Bit 22 of this command gives override control on the Light Pen sensitivity. If bit 22 is set to a ONE, this Set Point will be insensitive to the Light Pen, regardless of the Pen Enable bit of the previous Set C command.

2. Line Generation



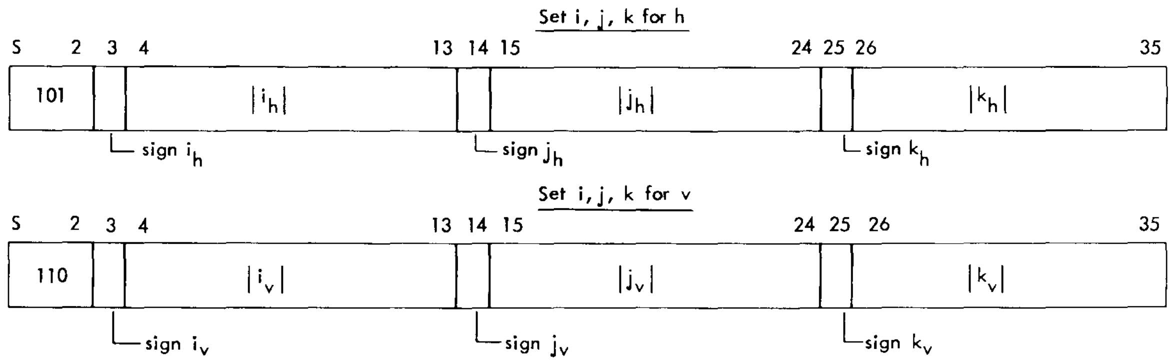
The Line Generator is made of three Binary Rate Multipliers sharing a common counter register. It generates Δx, Δy, and Δz pulse rates based on values set into it by the computer. The line will be begun at whatever point the beam was left from the previous command. Lines are actually drawn by the Plot Line (or Blank Line) command. If the line to be drawn contains a z component, a Set z command must precede the Plot Line (or Blank Line). The second command is necessary even if the line has only a z component ($\Delta x = \Delta y = 0$). Completion of the Plot Line (or Blank Line) command zeros the z register.

The number of x increments desired is placed in the decrement field and the number of y increments in the address portion of the Plot Line command. Z is put into the decrement of the Set z command.

All three are sign-magnitude numbers with the sign in the most significant bit of the 15-bit field and the magnitude in the least significant 10 bits. If the line is to be visible, bit 18 of the Plot Line command should be a zero. A blank line is drawn by making bit 18 a one. Blank lines are useful in repositioning the beam without breaking the continuity of a drawing. This point will be clarified in the discussion of the Rotation Matrix. Since the line generators contain 10 bits (plus sign), the longest possible line that can be drawn with normal dot spacing (1/100 inch steps) is 1023 increments (9 3/8 inches) in both x and y, the diagonal of the Scope Field. Dot spacing can be increased by a C Control command as discussed in Section D.7.

Lines are not restricted to the Scope Field, but can be drawn on any portion of the Total Console Field. A special programmable interrupt is available to alert the computer when a line is drawn which crosses the edge of the Scope Field (the portion of the Total Console Field which is visible) or when it leaves the Total Console Field entirely. An Edge Detection (ED) flip flop is provided which is set by bit 22 of the C Control command. When ED is ON (bit 22 of the last C Control command was a ONE), lines crossing the Scope Field edge in either direction will cause the interrupt. When ED is off (bit 22 of the last C Control command was a ZERO) the interrupt will occur only if the line leaves the Total Console Field. Bit 22 of the Plot Line Command gives override control on the Light Pen sensitivity in the same manner as it does for Set Point Commands.

3. Rotation Matrix



A three-dimensional rotation matrix has been included in the ESL Display Console, which works with the three-dimensional line generator.

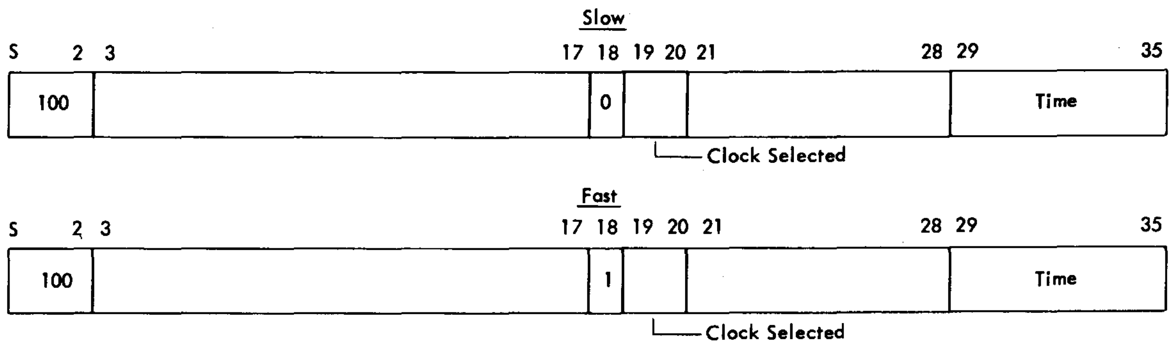
The rotation matrix units consist of three sets of 10 bit BRM pairs which have as inputs, the $\dot{\Delta}x$, $\dot{\Delta}y$, and $\dot{\Delta}z$ pulse-train outputs of the line generator. Thus, the three line generator outputs are each multiplied by a corresponding pair of numbers placed in the Rotation Matrix registers. If these numbers are the components of the unit vector relating the x, y, and z co-ordinate system to the h, v system, rotation will occur. For example, the BRM pair driven by the $\dot{\Delta}x$ output multiplies Δx by the vector cosines of the horizontal and vertical axes. Its outputs are $\dot{\Delta}x \cdot i_h$ and $\dot{\Delta}x \cdot i_v$ which are pulse trains whose rates are proportional to the horizontal and vertical components of Δx. In a similar manner the BRM pair driven by the $\dot{\Delta}y$ output of the line generator provides output $\dot{\Delta}y \cdot j_h$ and $\dot{\Delta}y \cdot j_v$ proportional to the horizontal and vertical components of Δy. The third BRM pair, driven by the $\dot{\Delta}z$ output of the generator, produces $\dot{\Delta}z \cdot k_h$ and $\dot{\Delta}z \cdot k_v$ proportional to the horizontal and vertical components of Δz.

The hardware automatically performs the summing of $\dot{\Delta}x \cdot i_h$, $\dot{\Delta}y \cdot j_h$ and $\dot{\Delta}z \cdot k_h$ into the horizontal incremental input, and the summing of $\dot{\Delta}x \cdot i_v$, $\dot{\Delta}y \cdot j_v$ and $\dot{\Delta}z \cdot k_v$ into the vertical incremental input. Thus, if the proper values are stored into its six data registers ($i_h, i_v, j_h, j_v, k_h, k_v$), the Rotation Matrix will automatically rotate whatever lines are specified in the x, y, and z axes into the horizontal (h) and vertical (v) axes of the scope.

It should be recognized that there is no inherent equipment limitation that requires the values placed in the Rotation Matrix data registers to be the proper vector cosines. The only limitation is that the numbers must all be less than ONE. The closest to ONE that can be set into these registers is 1777_8 which represents the number 1023/1024. The nature of the BRM is such that it is the 1024th pulse that will be lost. Since the longest line that can be drawn is 1023 increments, all steps will be plotted and no error will occur. It should be noted, however, that round-off errors from the BRM's do exist and picture distortions are visible as a picture is rotated.

There is no equipment in the console to determine the correct values for the Rotation Matrix settings. These are set solely by the computer with the two commands listed in Tables 1.1 and 1.2 as "Set i, j, k for h" and "Set i, j, k for v." In addition a command "Set i, j, k for d" has been reserved for possible inclusion of equipment to compute the depth component of lines. The data registers each accept ten bits of magnitude plus a sign bit. The binary point is just to the left of the most significant digit (e.g., i_{v9}, see Table 1.2, Section G).

4. Alarm Clocks



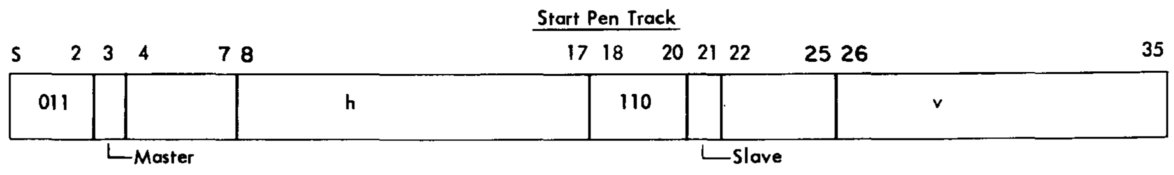
Two programmable Alarm Clocks are provided to give the computer a real-time interrupt source. There is provision in the order code for inclusion of two more Alarm Clocks as specified by bits 19 and 20 of the command. At present only 00 and 01 are active. The clocks are 7-bit counters which are preset by the last seven bits of the address field of the "Set Alarm Clock" command and are driven by either a 50-microsecond or a 1-millisecond clock oscillator. Bit 18 determines which clock source is to be used.

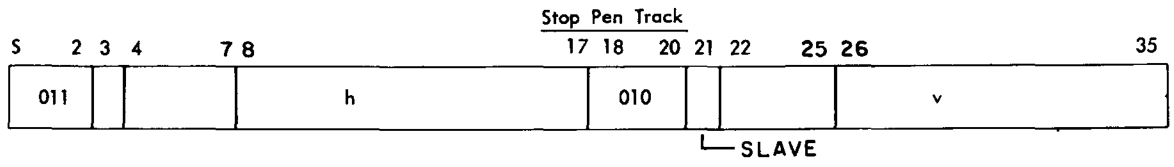
The 50-microsecond clock provides delays of up to 6.4 milliseconds. The 1-millisecond clock provides delays up to 0.128 seconds. Longer delays must be programmed by a succession of shorter ones. Because the clock oscillators are asynchronous, it is impossible to predict the accuracy of the clock closer than one oscillator cycle. Thus, if a clock is set to 67 and the 1-millisecond clock is specified, an interrupt will occur somewhere between 67 and 68 milliseconds after the command is accepted. A clock set to 0 milliseconds will ring in less than 1 millisecond.

When an alarm clock counter completes its cycle a Direct Data Interrupt will occur. To identify the source of the interrupt, Input Sense Line 8 is turned on for Alarm Clock 00 and Input Sense Line 9 is turned on for Alarm Clock 01. (See Table 1.3, Section G)

A SENSE LINE OUTPUT Reset of an Alarm Clock will turn it off so that no further interrupts will occur due to the clock. If such a Reset is given while the Alarm is ticking, an interrupt will occur, but no Alarm Clock flag will be set. If the Reset is given when the flag is already up, no interrupt will occur. If an Alarm Clock command is given to an already ticking Alarm, it will reset to the new time. An interrupt will occur at this time, but no flag will be set.

5. Light Pen Track





In its application in Computer-Aided Design Studies, the ESL Console is used a great deal for tracking the movements of the light pen. To lessen the pen tracking burden on the computer, an automatic tracking feature has been added to the Console, making use of the line generator for generating the tracking cross. There are separate pen-track registers for tracking on the Master and Slave independently.

A Pen Track command to start tracking on the Master (bit 3=ONE) causes the Master Pen Track Register to be loaded with the horizontal and vertical position specified by the decrement and address fields respectively. The hardware will generate a tracking cross and maintain its up-to-date position automatically every five milliseconds. Each Pen Track cross requires about 200 microseconds, thus pen tracking uses four percent of display time for one pen.

The computer may read the contents of the Pen Track Registers, whenever it desires, in the same manner that it reads the other input information available, as explained in a later paragraph. An important restriction exists however. Reading the Pen Track Register for the Master causes loss of the current h, v register contents, so it is best read after completion of a display list.

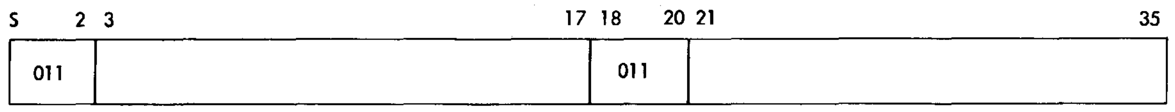
Pen Tracking on the slave is initiated by a Pen Track command with bit 21=ONE. It operates in an identical manner to Master Pen

Tracking, except that reading of the Slave Pen Track Register does not cause loss of the h, v registers. If both Master and Slave Pen Tracking generation are taking place, each takes 200 microseconds out of five milliseconds, so only 9.2 milliseconds out of ten is left for display time.

Since pen tracking cannot occur outside the Scope Field, the Pen Track Registers contain only ten bits each. Thus, the highest-order three bits of the h and v position read in from Pen Track Registers are extraneous, and are liable to contain false information. In order to interpret the pen position correctly as being on the Scope Field, these bits should be masked by the program and set to match the fourth-most significant bit. Also, since the present Pen Tracking circuits have a maximum resolution of two scope increments, the least significant bits of the h and v positions read in are meaningless and should be ignored.

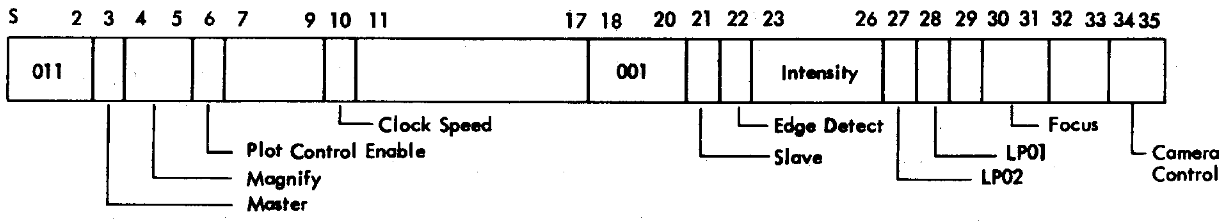
To halt pen tracking, a Pen Track command with a ZERO in bit 18 should be given. The tracking cross will disappear and the Pen Track Registers will contain the address and decrement of the Stop Tracking command.

6. End of File



This command merely sets up a pulse on the End of File Interrupt line into the Data Channel. This is a separate interrupt line from the Direct Data Interrupt, as discussed in IBM Special Features Bulletin for the Direct Data Channel (RPQ M90976). The End of File signal stops the Data Channel from sending further commands. The value of the End of File command is that it provides a way to terminate the Display List processing by the Data Channel without requiring that the number of output words in the list be counted ahead of time. The Address and Decrement fields are ignored by the Console, but can be read by the computer following an End-of-File interrupt. The present Display Controller uses this feature to provide several types of transfers useful in subroutining picture data.

7. Set C Control Word



Within the Display Console there are a number of special control flip-flops which affect various aspects of the display. This command will set these flip-flops if the appropriate bits (as shown in Table 1.2, Section G) are ONE's.

The meaning of each of these special controls is discussed below:

1. Master Bit 3. If the Plot Control Enable (Bit 6) is ONE, Bit 3 sets the Master Plot Control flip-flop. If Bit 6 is ZERO, the logical "and" of Bit 3 with the Master Plot Control flip-flop sets the Master flip-flop.

   When the Master flip-flop is ONE, normal plotting on the Master Scope will occur; when it is ZERO, the Master Scope will be blank.

2. M_1, M_0 (Magnify). Bit 4, 5. These bits control the size of the increment taken in the h and v registers for all lines (including spacing lines after characters).

3. Plot Control Enable. Bit 6. When this bit is a ONE, Bits 3, 21, 27 and 28 are used to set the Master Plot Control flip-flop, Slave Plot Control flip-flop, Master Light Pen Control flip-flop and Slave Light Pen Control flip-flop respectively. When it is a ZERO, the logical "and" of each of the flip-flops above with its corresponding bit in the Set C word will set the Master, Slave, LP01, and LP02 flip-flops respectively.

   The purpose of the Plot Control Enable is to allow users to produce display programs which will run on either the Master, the Slave, or both. In order to do this the user would specify that his picture should appear on both scopes. The Display Controller uses the Plot Control Enable to assure that the user's display affects only the scope(s) to which he is assigned, thus preventing interference between two users. A user display should never include a Set C word with a ONE in the Plot Control Enable bit.

4. Slow Clock. Bit 10. This bit controls the setting of a flip-flop (SLOW) which controls whether the basic console clock operates at its normal rate (a pulse every 1.8 sec), or at a slower rate (a pulse every 14.4 μsec) for driving remote direct view storage tubes. At one time several of these tubes were connected to the ESL Console as an experiment. These have since been removed so Slow Clock serves no function now.

5. Slave. Bit 21. This bit sets the Slave Plot Control flip-flop if Bit 6 is ONE, or the logical "and" of this bit with the Slave Plot Control flip-flop sets the Slave flip-flop if Bit 6 is ZERO. The Slave flip-flop controls the slave scope as the Master flip-flop controls the Master scope. (See Bit 3) Both Master and Slave flip-flops may be on simultaneously if identical displays are desired.

6. ED. Bit 22. This bit sets the Edge Detection flip-flop which determines whether to create a Direct Data Interrupt upon crossing the Scope Field edge (ED=ONE) or upon crossing the Total Console Field edge (ED=ZERO).

7. I_3 through I_0. Bits 23 - 26. These are the Intensity Control flip-flops. Considering Bit 26 as the least significant bit, larger numbers cause brighter displays.

8. LP02. 27. If Plot Control Enable (Bit 6) is a ONE, this bit sets the Slave Light Pen Control flip-flop. If Bit 6 is a ZERO, the logical "and" of Bit 27 with the Slave Light Pen Control flip-flop sets the LP02 flip-flop. When set to ONE, the LP02 flip-flop enables Light Pen No. 2 (Slave Pen) to respond to points and lines drawn on the scope. It has no effect on Pen Tracking with Light Pen No. 2. If set to a ZERO this control can be used to exclude desired parts of the display from being seen by the Light Pen.

9. LP01. Bit 28. Similar to LP02, except it is for Light Pen No. 1 (Master Pen).

10. F_1, F_2. Bits 30, 31. These flip-flops eventually will affect the focusing of the beam, but are not presently connected.

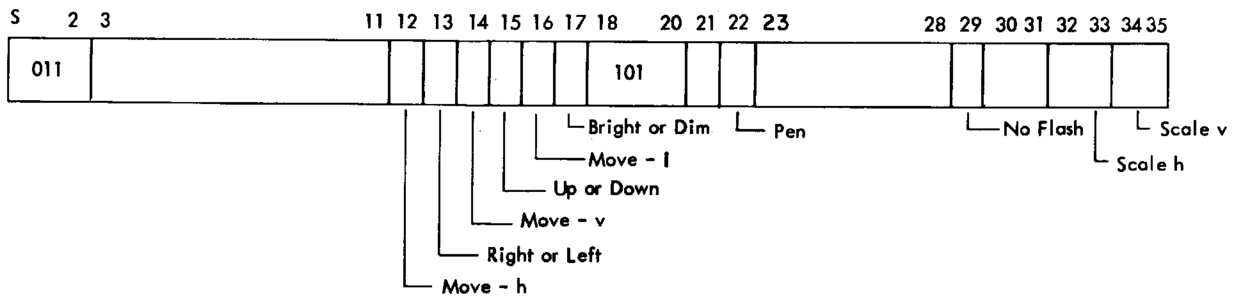
11. Camera Control. Bits 34, 35. These bits provide signals to control the operation of a D. B. Milliken movie camera. The bits are decoded as shown.

34	35	Action
0	0	No Action
0	1	Hang up Display Console for 125 msec.
1	0	Close shutter. Do not advance film.
1	1	Advance film (film will stop in open shutter position) and hang up Display Console for 125 msec.

The purpose of these controls is to synchronize the movie camera operation to the display frames. If a Set C command with bits 34 and 35 both ONE is provided at the beginning of a display file, the display is hung up for 125 ms while the camera advances a frame and stops with its shutter open. When the 125 ms times is up the display is released to process the display file and expose the film. The next cycle of the display file will repeat the operation. To stop filming, the state of the Set C command should be switched to bit 34 ONE, bit 35 ZERO. This will cause the shutter to close without advancing the film, thereby preventing overexposure of the last frame taken.

If bits 34 and 35 are both ZERO no special action is taken, thus Set C commands can appear in the display file without affecting the camera.

8. Set F Control Word



The F control word allows access to the increment inputs to the h and v registers directly. In this way any sort of display generator can be simulated. The controls available are:

1. Move-h. Bit 12. A ONE causes an increment to occur in the horizontal direction of whatever magnitude has been called for in the Scale h_1, h_0 flip-flops (bits 32, 33 of this command).

2. Right or Left. Bit 13. This control determines the direction of any horizontal step called for by move-h. If a ONE, the step will be taken to the left. A ZERO calls for the step to be to the right.

3. Move-v. Bit 14. Similar to Move-h, but in the vertical direction. Its magnitude is controlled by the Scale v_1, v_0 flip-flops (bits 34, 35 of this command).

4. Up or Down. Bit 15. This determines the direction of any vertical step called for by Move-v. A ONE causes steps to be taken downward.

5. Move-I. Bit 16. Similar to Move-h, but it increments or decrements the Intensity Register. The step is always just one unit.

6. Bright or Dim. Bit 17. This determines whether Move-I commands intensify or dim the beam. A ONE in bit 17 causes the beam to get dimmer.

7. Pen. Bit 22. This bit gives override control on the Light Pen Sensitivity in the same manner it does for Line and Set Point commands.

8. No Flash. Bit 29. This controls whether the point called for by this command is to be plotted or not. A ONE inhibits plotting.

9. Scale h_1, h_0, v_1, v_0. Bits 32-35. These flip-flops determine the size of the increments that are to be taken in h and v. The following table applies:

Horizontal Step Size	Bit 32 h_1	Bit 33 h_0	Vertical Step Size	Bit 34 v_1	Bit 35 v_0
1	0	0	1	0	0
2	0	1	2	0	1
4	1	0	4	1	0
8	1	1	8	1	1

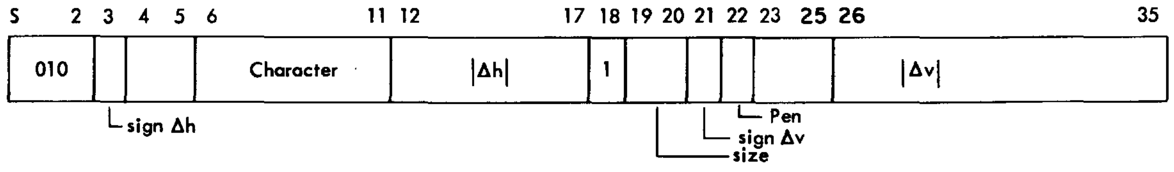
Note that the present beam position can be intensified by an F Word command with no step taken (Move h= Move v = Move I = 0) and plotting (bit 29=0).

9. Character Generation

The Display Console contains two symbol generation systems: a Straza Symbol Generator for producing 64 standard symbols, and an M.I.T. constructed Special Character Generator for creating any pattern on a 5 x 7 matrix for display of nonstandard symbols. In addition, a flexible automatic format control (spacing between characters) is provided by making use of the line generator. Table 1.6 in Section G lists the character code for the Straza Symbol Generator.

There are three modes of operation of the Display Console in generating characters. In the first mode (Unpacked Straza Mode), a single 6-bit character is given with each command. In the Packed Straza Mode, six characters are packed into a single output word. The third mode is the one in which nonstandard symbols are produced using a full 36-bit word to specify each character.

a. Unpacked Mode



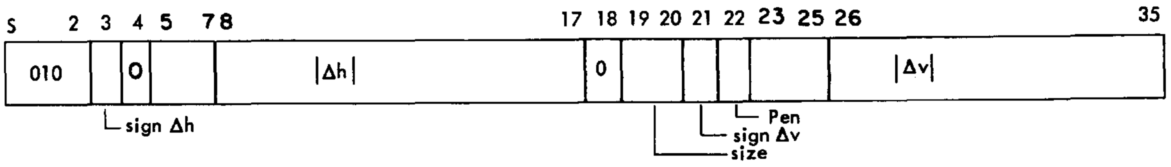
In the unpacked mode, the 6-bit code of the character is held in bits 6 through 11 (see Table 1.2 in Section G). The Δh spacing after the character is contained in bits 12 through 17, with the sign of Δh in bit 3. The Δv spacing is held in the address portion of the word and is ten bits in magnitude with sign in bit 21. The character is drawn by the Straza Character Generator starting at the location specified by the h and v registers, and will appear below and to the left of the starting point. One of the four character sizes is selected by bits 19 and 20 which set the S_1 and S_0 flip-flops.

After the character has been plotted, the line generator is automatically turned on and a blank line is drawn of whatever length specified in the command. This permits spacing to the correct position for the next character. Note that the dot spacing for the blank lines after characters is automatically set to twice that for normal lines, and is further affected by the setting of the Magnify (M_1, M_0) flip-flops. Thus, the largest Δh character spacing is 63 increments of 2 each (1.26 inches) with M_1 and M_0 set to ZERO, 63 increments of 4 each (2.52 inches) if M_0 is ONE and M_1 is ZERO, or 63 increments of 8 each (5.04 inches) if M_1 is ONE and M_0 is ZERO. (Normal spacing for size 0 characters with M_1 and M_2 ZERO is Δh=6, Δv=0.)

It is also important to note that both the character generator and line generator outputs bypass the rotation matrix, and thus are unaffected by it. By making the character spacing not rotate it is possible to associate written text with the end point of a line and have the text follow the end point as the line is rotated around on the scope. Note that if a block of several lines of text is attached in this manner, the spacing from the end of one line of text to the start of the next should be done by generating a blank character with appropriate -Δh, -Δv to produce a carriage return which will not be affected by rotation. If a line generate command were used instead, this line would be affected by the rotation matrix and text line 2 would rotate about the end of text line 1.

Bit 22 gives override control on Light Pen sensitivity for this character in the same manner it does for Set Point and Line Commands.

b. Packed Mode



This mode for character generation was included to allow more dense packing of characters for standard textual output. It is made to conform to the Long Word format (.BCQ.) of the AED Compiler.

The Packed Character Generator command sets the Line Generator to the character spacing desired and puts the Console into a special mode in which it processes succeeding words as blocks of six characters in the format shown below. The first character (bits S through 5) of the first word following the command specifies the number of characters to follow it (i.e., how many characters are contained in the Long Word), and whether or not this is the terminating Long Word. Bits 1 through 5 give the number of characters (up to 30) and bit S designates whether to terminate the packed character processing mode (bit S = ZERO) after this Long Word, or to treat the succeeding word as a new Long word (bit S = ONE).

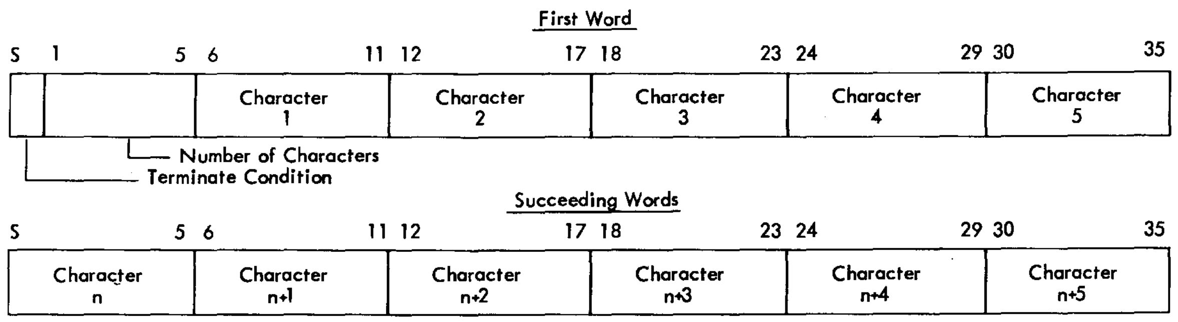


Figure 1.3 illustrates a sample display list for the packed character generate mode. The Console will display the 17 characters of the

	PACKED CHARACTER GENERATE COMMAND					
1st Long Word	110001	Character 1	Character 2	Character 3	Character 4	Character 5
	Character 6	Character 7	Character 8	Character 9	Character 10	Character 11
	Character 12	Character 13	Character 14	Character 15	Character 16	Character 17
2nd Long Word	100001	Character 1				
3rd Long Word	000110	Character 1	Character 2	Character 3	Character 4	Character 5
	Character 6					
	NEXT COMMAND					

Fig. 1.3 Sample Packed-Character Display List

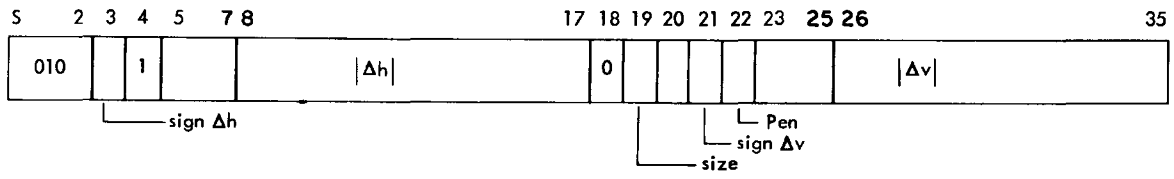
First Long Word, the one character of the second Long Word and the six characters of the third Long Word in a string, each succeeding character spaced by the amount Δh, Δv as called for in the command.

If the Scope Field Edge is crossed by the line generator during character spacing and the Edge Detect flip-flop (ED) is ON, an interrupt is generated identical to that described under Line Generation. No such interrupt will occur if the character itself spills over the edge.

Bits 19 and 20 select the size as in Unpacked Mode.

Bit 22 gives override control on the Light Pen sensitivity in the same manner it does for Line and Set Point commands.

c. Special Character Mode



This mode of character generation permits the creation of any symbol desired on a matrix array of dots. A Special Character Generator command loads the Line Generator with Δh, Δv information for spacing <u>after</u> the array is displayed, then puts the Display Console into a mode in which it processes each succeeding word as 35 bits to be converted into a 5 x 7 array of dots. The 36th bit (bit S) is used to determine whether to terminate the mode (bit S=ZERO) or to process the next word as another array (bit S=ONE). The format of the array is shown in Fig. 1.4.

Start → X	6	12	18	24	30
	25	19	13	7	1
	31	2	8	14	20
	15	9	3	32	26
	21	27	33	4	10
	5	34	28	22	16
	11	17	23	29	35
				End →	X

Fig. 1.4 Assignment of Bits in Special Character

To intensify any of the points of the array the appropriate bit of the data word is set to a ONE. Dot spacing is normally 2 in this mode, but by setting the SIZ 1 flip-flop ON (Bit 19=1), this can be expanded to 4. These can be expanded again by a factor of two by setting the Magnify flip-flop (M_0) ON. M_0 also affects the line drawn after the character, but the SIZ 1 flip-flop does not. Although 5 x 7 is the standard size in this mode, it is easy to build up larger, more complicated symbols by blocking groups of these arrays together.

These symbols bypass the rotation matrix, but not the h, v register. Note that since h and v are altered, the beam is not repositioned to its starting point at the end of a character. As shown in Fig. 1.4, the starting point is one position to the left of the upper left-hand corner of the character, and the beam will end one position below the lower right-hand corner. In spacing the beam for the next character, this repositioning must be accounted for. (For size 0 characters, $\Delta h=1$, $\Delta v=7$ will provide normal spacing.) Character spacing is still twice normal size in this mode. The rather strange bit pattern comes about because of the way the matrix is generated, and, because the data register and shifting logic used for the special characters are the same as those used for the packed Straza Characters. To form the matrix, the line generator increments across the top row from left to right, drops down one row, and increments from right to left, etc. For each increment, bits 1 through 35 of the data word are rotated left six places and bit position 1 is tested to decide whether to intensify. In making up bit patterns for special characters, it is suggested that a chart such as Fig. 1.4 will be helpful.

Bit 22 gives override control on the Light Pen sensitivity in the same manner it does for Line and Set Point Commands.

E. INPUT TO THE COMPUTER

The Display Controller computer is able to read into its memory the h and v registers of the Display Console, the Pen Track registers, and the data from the various devices on the control panel, which are described in the next section. To accomplish this, the computer first selects the input source by presenting Sense Lines with four Sense Lines coded to the desired unit. Table 1.4 indicates the existing input sources and their codes for selection. Table 1.5 indicates the bit positions of the input words from h, v registers, the shaft encoders and crystal ball, and the Pen Track registers. The push buttons and switches are labeled as to their bit assignments. Selecting an input source causes the Console to put the data on the input lines to the Data Channel. The computer must then read the Data Channel to input a single word.

Read Control of the Console has been made independent of Write Control. It is thus possible to read information from the console into the computer while the console is hung in some output condition, such as an Edge Crossing. This permits the computer to read h, v to determine the exact location of the crossing and then release the Console by means of a Reset Flags to finish the display list.

Identical facilities are provided at the master and slave stations, with separate selection codes for read in to the computer; thus the following discussions apply either to the master or the slave stations.

1. Control Panel

Figure 1.5 illustrates the control panel provided for input to the computer. At the bottom of the panel are the three 7-bit binary Shaft Encoders. There is no internal relation between the position of these knobs and anything else in the Display Console, and their interpretation is handled purely by program. There is no interrupt caused when these are moved, and if the program is concerned with the position of these knobs, it must sample their contents often enough to make sense out of the data. There is no indication of the direction that the knob is turning, other than whether the number presented is getting larger or smaller. Note that since the code is continuous (modulo 2^7), the sampling must be sufficiently rapid to insure the shaft does not rotate more than 180^o between samples. Normally 30 times per second is a satisfactory sampling rate.

To the right of the knobs are located two switches marked BEAM OFF and POWER and a button marked RESET. The BEAM OFF switch originally blanked the beam so the operator could protect the screen from being burned by an improper program (one that continuously intensifies the same spot). It was found that by keeping the intensity level adjusted properly this switch function was not necessary so it was disconnected. Since then, the switch has been wired to override the no-plot control, to allow display of all lines, blank or plotted, on screen or off. This has proved useful for debugging purposes. The RESET button resets the active control flip-flops of the Console. In this state the Console is ready to process a new command. The computer is not alerted in any way when this button has been activated. The POWER switches on the Control Panels are disconnected.

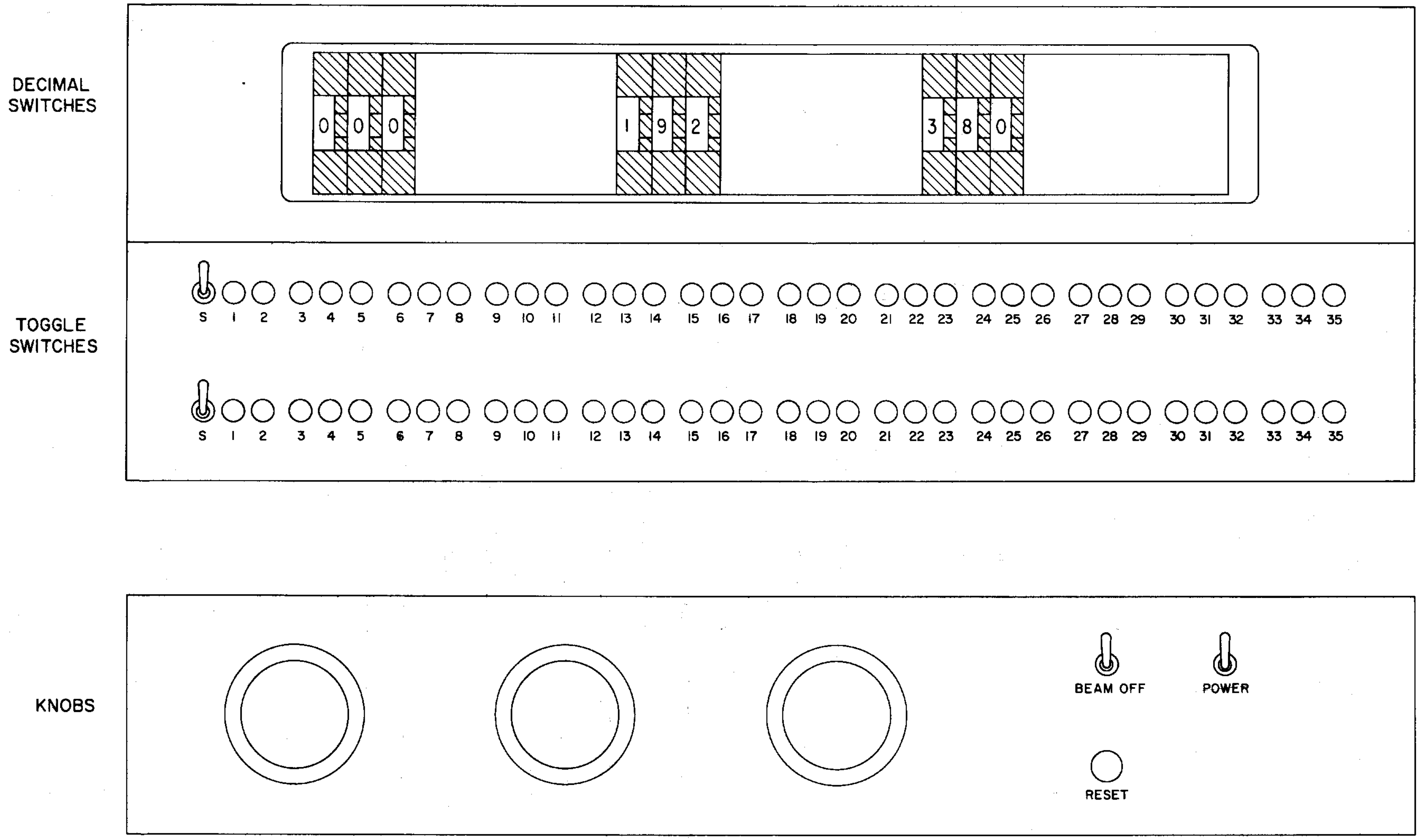

DECIMAL SWITCHES
0 0 0
1 9 2
3 8 0
TOGGLE SWITCHES
S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
KNOBS
BEAM OFF
POWER
RESET

Fig. 1.5 Control Panel

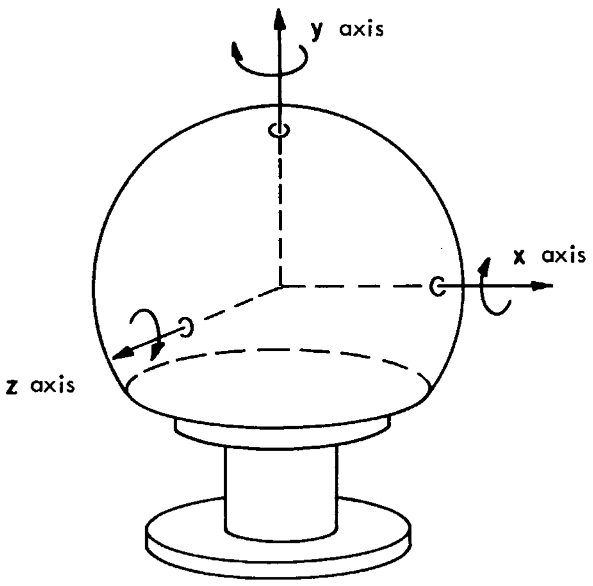
Behind the panel on the Master, a Sonalert buzzer has been wired to the WHO flip-flop to give the operator an active alarm. The WHOA flip-flop was chosen because a continuous ONE state for this flip-flop usually indicates a "computer hung-up" condition. The function of this flip-flop is described later.

Above the knobs are two banks of 36 toggle switches, each of which constitutes one input word. These switches provide arbitrary control functions. There is no internal relation between the position of these switches and anything else in the Display Console, and their interpretation is handled purely by program. Like the knobs, these switches are sampled by the computer at whatever rate is set by the program.

Above the toggle switches is a bank of nine binary coded decimal switches. These are mechanically mounted on horizontal slides in such a way that they can be spaced into any grouping. Thus, they can be interpreted as nine individual 10-position switches or three 1000-position inputs or whatever arrangement the user desires. The bit assignment of the switches is shown in Table 1.5.

2. Crystal Ball

The Crystal Ball or "Globe," illustrated in Fig. 1.6, has spring-loaded limited rotation about three axes of rotation, and provides input



FOR EACH AXIS	
CODE $g_3\ g_2\ g_1\ g_0$	MEANING*
1 0 0 0	Highest negative rate
1 1 0 0	Middle negative rate
0 1 0 0	Lowest negative rate
0 0 0 0	Reset Position. Zero rate
0 0 1 0	Lowest positive rate
0 0 1 1	Middle positive rate
0 0 0 1	Highest positive rate

* Considering clockwise rotation as positive

Fig. 1.6 Crystal Ball

codes to the computer for three positions in each direction about each axis. The device is intended to be used as a three-step rate control to provide a natural easy control over the rotation of a three-dimensional object viewed on the screen. The input codes are merely processed through the Console, however, and do not directly effect any console registers. Thus the Crystal Ball can be used in any way that the programmer desires. Figure 1.6 shows the relation between ball position and binary input for each axis (assuming positive rates are for clockwise rotation).

3. Push Buttons

The Push Button Box (Fig. 1.7) is a unit containing 36 push buttons for control of the computer program. Pushing of any button will

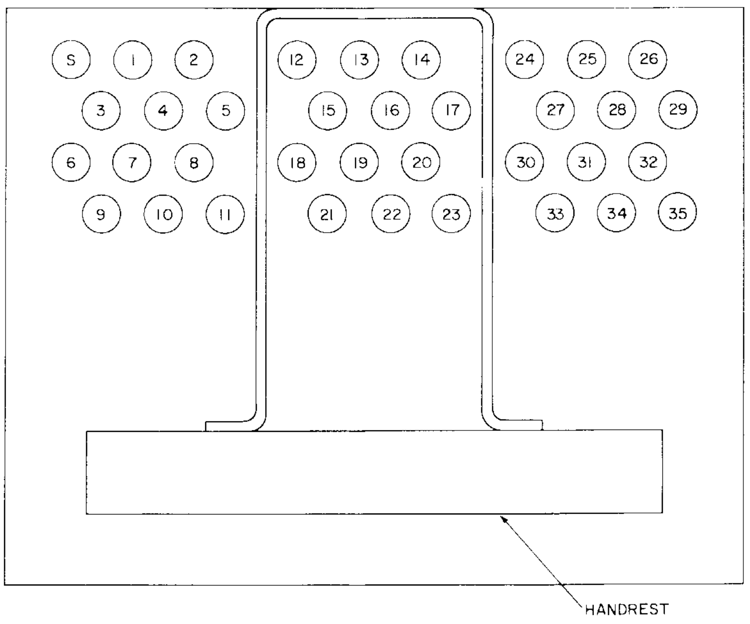


Fig. 1.7 Push-Button Box

cause a Channel Interrupt to occur. The computer should then read in the push buttons to determine which one was pushed. Because there is no buffering of the push button data, it is possible (though not likely) for the computer not to get around to sample the buttons before a button is released. It is, therefore, advisable to give some visible indication on the display that a push button has been recognized. Like the switches and shaft encoders, no wired control has been assigned to any buttons. Their interpretation is controlled strictly by the program.

The Push Button box has been designed for easy use without looking at it, so that the operator's attention is not diverted from the CRT Display. The buttons are arranged like a keyboard and they are designed for very light pressure. A special hand rest is provided with spacing bars separating groups of buttons to aid in locating oneself without looking at the box. A plastic holder is also provided above the box to slip an identification card into, so each user can associate the buttons with the particular uses in his program.

4. Interrupts

The Channel Interrupt is used to call the Central Processor into play when special conditions occur in the Console. The sources of interrupt are as follows:

a. Alarm Clocks. One of the alarm clocks has rung. Input Sense Lines 1 and 2 identify these interrupts.

b. Light Pen. If the light pen is enabled and it sees a point, line or character on the scope, an interrupt is generated and Input Sense Line 6 is set to ONE. In addition, the Display Console is "hung up" in a manner depending on what it was doing. If it is in the process of generating a line, it will halt as soon as the pen responds, leaving part of the line unfinished. If the pen responds to a Set Point, the Console will halt before accepting the next command. If the pen sees a Straza character, the Console will halt just before plotting the blank spacing line to the next character. Special characters work just like lines.

   The use of the light pen interrupt to identify to the computer a particular item of displayed information is an important means of man-machine communication. Set points can be used as "light buttons," i.e., programmable switches. Displayed lines or characters can be altered, moved, or erased. The computer can identify lines in two ways. The exact point on a line seen by the pen can be obtained by reading in the h, v register when this interrupt occurs. If just identification of which line has been seen is enough, the computer can examine the state of data channel and determine the memory address of the next output command. Note that the only way to identify which of six packed characters was seen is to establish each character location and compare it to the h, v register.

Presenting Sense Lines with the Reset Flag bit ON (Line 6) will release the line generator and allow an interrupted line to be completed. If the h, v register word is read in by the computer, bits 3 and 21 contain flags which identify which light pen (Master or Slave) saw the line. Bit 3 flag is set by the Master Pen. Bit 21 flag is set by the Slave Pen. Both are reset by Reset Flags (Output Sense Line 6).

c. Edge of Scope. This interrupt, which is identified by Input Sense Line 7, occurs if the Edge Detect flip-flop (ED) is ON and the Scope Field edge is crossed by a line during Line Generation or spacing for Character Generation. If ED is OFF, it occurs when a line is drawn completely off the Total Console Field. When this interrupt occurs, the Line Generator is hung up in the same way as described in the previous paragraph.

The same two options described for the Light Pen interrupt are available to the programmer after the Edge of Scope interrupt. Thus, the computer can find the exact edge co-ordinates of the line crossing, or just identify which line is the culprit.

d. Push Buttons. When any of the 36 push buttons available on the Master Console for input to the computer is pushed, an interrupt is caused and Input Sense Line 8 is turned ON. The computer can then input these buttons to determine the particular button pushed. The sense line flag is reset by a Reset Flags (Output Sense Line 6).

The slave push buttons cause Input Sense Line 3 to be turned on. This flag is also reset by Reset Flags.

F. INTERRUPT PROCESSING

The programmer should be aware that the processing of Channel Interrupts is fraught with hazards. Some are imposed by the nature of the 7094 Direct Data Channel, others by the characteristics of the Display Console.

Halting the Data Channel in the midst of a display list to process an interrupt can also cause difficulties. For example, it is necessary for the computer to know reliably where to restart the list. If the computer merely stores the channel address to preserve the list address, and then resets the channel to start the list again, it cannot be sure that the Display Console did not accept the command that was being held on the output lines just before resetting the channel. If this occurs, an address is skipped which produces annoying jumps in the display. To avoid this problem the WHOA flip-flop was added to the Console which causes the Console to hang up before accepting the next command. The computer sets the WHOA flip-flop by presenting Sense Lines with Line 5 of the Sense Output word = ONE. By giving such an instruction several cycle times before storing the channel the computer can be sure that the Data Channel address will not be stored just as it is about to change state.

If the Data Channel is stopped on a data word for packed or special characters, the Console is in a condition where it expects the next output word to be a data word, not a command. If the computer were to attempt to output any command at this time, it would be improperly interpreted. For a command to be processed correctly in this situation, the Console must be reset with a Panic Reset before the command is put out. This, however, clears the Console's memory that it is awaiting a character word. Restarting on the character word would foul up the rest of that one pass through the display. The computer must either restart the display back at the beginning or it must be able to identify the stored channel address as being that of a Character Word and restart at the last Set Point Command. Reading the h, v register can determine that this is a Character Word (Bit S = ONE).

G. COMMAND FORMAT, BIT ASSIGNMENTS AND CHARACTER CODES

The following tables summarize the hardware assignments.

Table 1.1

COMMAND FORMAT FOR ESL DISPLAY CONSOLE

7094 Bits							Function	Option
Prefix			Tag					
S	1	2	18	19	20	4		
0	0	1	X	X	X		Set ΔZ component of line	
0	0	0	0	X	X		Plot line $\Delta X, \Delta Y, (\Delta Z)$	Visible
			1	X	X		" "	Invisible
0	1	0	0	S_1	S_0	0	Character Generation	Packed Mode
			0	X	S_0	1	" "	Special Symbol Mode
			1	S_1	S_0		" "	Unpacked Mode
0	1	1	0	0	0		Plot Set Point X, Y	Visible
			1	0	0		" "	Invisible
			0	0	1		Load Control Word C	Set various console parameters
			1	0	1		Load Control Word F	Incremental beam control
			0	1	0		Start Light Pen Track	Master or Slave
			1	1	0		Stop " "	" "
			0	1	1		End of File	Jump Address
			1	1	1		Not Used	
1	0	0	0	A_1	A_0		Set Alarm Clock	Slow
			1	A_1	A_0		" "	Fast
1	0	1	X	X	X		Set i, j, k, for h	
1	1	0	X	X	X		Set i, j, k, for v	
1	1	1	X	X	X		(Reserved to set i, j, k for depth coordinate, d)	

Notes: S_1 and S_0 specify character size (4 sizes available).

A_1 and A_0 specify which of four Alarm Clocks, (only two installed).

X indicates "don't care".

Table 1.2

COMMAND BIT ASSIGNMENTS FOR ESL DISPLAY CONSOLE

7094 Bits	Set Point	Set z	Generate Line	Unpacked Character Generator	Packed Character Generator	Special Character Generator	Set i, j, k for h	Set i, j, k for v	Set C Control	Set F Control	Light Pen Track	Alarm Clock	EOF	7094 Bits
S	0	0	0	0	0	0	1	1	0	0	0	1	0	S
1	1	0	0	1	1	1	0	1	1	1	1	0	1	1
2	1	1	0	0	0	0	1	0	1	1	1	0	1	2
3		Sign z	Sign x	Sign x	Sign x	Sign x	Sign i_h	Sign i_v	Master (1)		Master (1)			3
4					0	1	i_{h9}	i_{v9}	M_1 } Increment Size					4
5	h_{12}						i_{h8}	i_{v8}	M_0					5
6	h_{11}			c_5			i_{h7}	i_{v7}	Plot Control					6
7	h_{10}			c_4			i_{h6}	i_{v6}						7
8	h_9	z_9	x_9	c_3 } Char.	h_9	h_9	i_{h5}	i_{v5}			h_9			8
9	h_8	z_8	x_8	c_2	h_8	h_8	i_{h4}	i_{v4}			h_8			9
10	h_7	z_7	x_7	c_1	h_7	h_7	i_{h3}	i_{v3}	Clock } Normal (0) Slow (1)		h_7			10
11	h_6	z_6	x_6	c_0	h_6	h_6	i_{h2}	i_{v2}			h_6			11
12	h_5	z_5	x_5	h_5	h_5	h_5	i_{h1}	i_{v1}		Move h	h_5			12
13	h_4	z_4	x_4	h_4	h_4	h_4	i_{h0}	i_{v0}		Right (0) Left (1)	h_4			13
14	h_3	z_3	x_3	h_3	h_3	h_3	Sign j_h	Sign j_v		Move v	h_3			14
15	h_2	z_2	x_2	h_2	h_2	h_2	j_{h9}	j_{v9}		Up (0) Down (1)	h_2			15
16	h_1	z_1	x_1	h_1	h_1	h_1	j_{h8}	j_{v8}		Move I	h_1			16
17	h_0	z_0	x_0	h_0	h_0	h_0	j_{h7}	j_{v7}		Bright (0) Dim (1)	*			17
18	Plot (0) or Blank (1)		Plot (0) or Blank (1)	1	0	0	j_{h6}	j_{v6}	0	1	On (1) Off (0)	Fast (1) Slow (0)	0	18
19	0			S_1 } Size	S_1 } Size	S_1 (Size)	j_{h5}	j_{v5}	0	0	1	A_1 } Clock	1	19
20	0			S_0	S_0		j_{h4}	j_{v4}	1	1	0	A_0	1	20
21			Sign y	Sign y	Sign y	Sign y	j_{h3}	j_{v3}	Slave (1)		Slave (1)			21
22	Pen		Pen	Pen	Pen	Pen	j_{h2}	j_{v2}	ED (Edge Detect)	Pen				22
23	v_{12}						j_{h1}	j_{v1}	I_3					23
24	v_{11}						j_{h0}	j_{v0}	I_2 } Intensity					24
25	v_{10}						Sign k_h	Sign k_v	I_1					25
26	v_9		y_9	v_9	v_9	v_9	k_{h9}	k_{v9}	I_0		v_9			26
27	v_8		y_8	v_8	v_8	v_8	k_{h8}	k_{v8}	LP02 (Slave)		v_8			27
28	v_7		y_7	v_7	v_7	v_7	k_{h7}	k_{v7}	LP01 (Master)		v_7			28
29	v_6		y_6	v_6	v_6	v_6	k_{h6}	k_{v6}		No Flash	v_6	t_6		29
30	v_5		y_5	v_5	v_5	v_5	k_{h5}	k_{v5}	F_1 } Focus		v_5	t_5		30
31	v_4		y_4	v_4	v_4	v_4	k_{h4}	k_{v4}	F_2		v_4	t_4		31
32	v_3		y_3	v_3	v_3	v_3	k_{h3}	k_{v3}		s_{h1}	v_3	t_3 } Time		32
33	v_2		y_2	v_2	v_2	v_2	k_{h2}	k_{v2}		s_{h0} } Scale	v_2	t_2		33
34	v_1		y_1	v_1	v_1	v_1	k_{h1}	k_{v1}	Camera	s_{v1}	v_1	t_1		34
35	v_0		y_0	v_0	v_0	v_0	k_{h0}	k_{v0}	Control	s_{v0}	*	t_0		35

*

Note: A blank in any bit position indicates "Don't Care"

Table 1.3
SENSE INPUT AND OUTPUT WORDS

Sense Input Number	IBM Bit	Meaning
1	8	Alarm Clock No. 1 Rung
2	9	Alarm Clock No. 2 Rung
3	10	Push Button 2 (Slave)
4	11	Not Used
5	12	Not Used
6	13	Light Pen Seen
7	14	Edge of Scope
8	15	Push Button 1 (Master)
9	16	Not Used
10	17	Not Used

Sense Output Number	IBM Bit	Meaning	
1	8	Reset Alarm Clock No. 1	
2	9	Reset Alarm Clock No. 2	
3	10	Not Used	
4	11	WHOA	
5	12	Panic Reset	
6	13	Reset Flags	
7	14	Select Register Bit 0	See Table 1.4
8	15	Select Register Bit 1	
9	16	Select Register Bit 2	
10	17	Select Register Bit 3	

Table 1.4

SENSE LINE OUTPUT BITS FOR INPUT SELECTION

Sense Output Number				Device Selected
7	8	9	10	
0	0	0	0	h,v Register
0	0	0	1	Unassigned
0	0	1	0	Push Buttons 1
0	0	1	1	Push Buttons 2
0	1	0	0	Auto Pen Track Register 1
0	1	0	1	Auto Pen Track Register 2
0	1	1	0	Switch Bank 1 B (lower bank)
0	1	1	1	Switch Bank 2 B (lower bank)
1	0	0	0	Shaft Encoders and Crystal Ball 1
1	0	0	1	Shaft Encoders and Crystal Ball 2
1	0	1	0	Switch Bank 1 A (upper bank)
1	0	1	1	Switch Bank 2 A (upper bank)
1	1	0	0	Not Used
1	1	0	1	Not Used
1	1	1	0	Decimal Switches 1
1	1	1	1	Decimal Switches 2

Note: Master = 1, Slave = 2

Table 1.5

DATA INPUT BIT ASSIGNMENTS

7094 Bits	h, v Registers	Shaft Encoders		Decimal Digiswitches		Light Pen 1 Register	Light Pen 2 Register
S	Char. Cmd.			$d9_3$			
1				$d9_2$	9th (leftmost) Digiswitch		
2				$d9_1$			
3	LP1S	g_{x3}		$d9_0$			
4		g_{x2}	Crystal Ball x axis	$d8_3$			
5	h_{12}	g_{x1}		$d8_2$			
6	h_{11}	g_{x0}		$d8_1$	8th		
7	h_{10}	g_{y3}		$d8_0$			
8	h_9	g_{y2}	Crystal Ball y axis	$d7_3$		PT1 h_9	PT2 h_9
9	h_8	g_{y1}		$d7_2$		PT1 h_8	PT2 h_8
10	h_7	g_{y0}		$d7_1$	7th	PT1 h_7	PT2 h_7
11	h_6	g_{z3}		$d7_0$		PT1 h_6	PT2 h_6
12	h_5	g_{z2}	Crystal Ball z axis	$d6_3$		PT1 h_5	PT2 h_5
13	h_4	g_{z1}		$d6_2$		PT1 h_4	PT2 h_4
14	h_3	g_{z0}		$d6_1$	6th	PT1 h_3	PT2 h_3
15	h_2	$e_{\ell 6}$		$d6_0$		PT1 h_2	PT2 h_2
16	h_1	$e_{\ell 5}$		$d5_3$		PT1 h_1	PT2 h_1
17	h_0	$e_{\ell 4}$		$d5_2$		*	*
18		$e_{\ell 3}$	left encoder	$d5_1$	5th		
19		$e_{\ell 2}$		$d5_0$			
20		$e_{\ell 1}$		$d4_3$			
21	LP2S	$e_{\ell 0}$		$d4_2$			
22		e_{c6}		$d4_1$	4th		
23	v_{12}	e_{c5}		$d4_0$			
24	v_{11}	e_{c4}		$d3_3$			
25	v_{10}	e_{c3}	center encoder	$d3_2$			
26	v_9	e_{c2}		$d3_1$	3rd	PT1 v_9	PT2 v_9
27	v_8	e_{c1}		$d3_0$		PT1 v_8	PT2 v_8
28	v_7	e_{c0}		$d3_3$		PT1 v_7	PT2 v_7
29	v_6	e_{r6}		$d2_2$		PT1 v_6	PT2 v_6
30	v_5	e_{r5}		$d2_1$	2nd	PT1 v_5	PT2 v_5
31	v_4	e_{r4}		$d2_0$		PT1 v_4	PT2 v_4
32	v_3	e_{r3}	right encoder	$d1_3$		PT1 v_3	PT2 v_3
33	v_2	e_{r2}		$d1_2$	1st (rightmost) Digiswitch	PT1 v_2	PT2 v_2
34	v_1	e_{r1}		$d1_1$		PT1 v_1	PT2 v_1
35	v_0	e_{r0}		$d1_0$		*	*

* The present Pen-tracking circuits are such that the smallest step is equal to two scope increments. These bits are reserved for PT h_0 nad PT v_0 if the smallest tracking step is changed to one scope increment.

TABLE 1.6 STRAZA SYMBØL GENERATØR CØDE ASSIGNMENT

ØCTAL CØDE	SYMBØL
00	0
01	1
02	2
03	3
04	4
05	5
06	6
07	7
10	8
11	9
12*	~
13	=
14	'
15*	_
16*	@
17*	&
20	+
21	A
22	B
23	C
24	D
25	E
26	F
27	G
30	H
31	I
32*	;
33	.
34	)
35	:
36*	↑
37*	"
40	-
41	J
42	K
43	L
44	M
45	N
46	Ø
47	P
50	Q
51	R
52*	%
53	$
54	*
55*	←
56*	?
57*	!
60	(SPACE)
61	/
62	S
63	T
64	U
65	V
66	W
67	X
70	Y
71	Z
72*	>
73	,
74	(
75*	]
76*	[
77*	<

* Note: This Code differs from the BCD Code used on CTSS.

CHAPTER II

THE DISPLAY CONTROLLER

A. INTRODUCTION

Application of highly interactive computer graphics to large, complex design problems imposes stringent requirements for both real-time response and extensive computation and storage facilities. If a sophisticated graphics console is to provide acceptable interaction with a user, certain real-time actions are necessary:

1. The picture must be maintained on the screen at all times by executing a display file of commands to the display unit sufficiently often to minimize flicker.
2. There must be very fast response to interrupts (e.g., light pen "see", button push).
3. Execution of real-time programs must be provided at regular intervals in order that certain transformations on the screen (e.g., rotation of a three-dimensional picture) appear to be taking place continuously.

The computation facility which provides these services will be called the Display Controller.

In addition to these real-time requirements, the user also needs occasional extensive computation for manipulating his data structure, for analysis or for simulation. Finally, he needs mass storage to retain his data and programs. However, it is uneconomic for a single on-line user to operate a computer with facilities which he needs only occasionally. A large time-shared computer such as the IBM 7094 at Project MAC meets this need for computation and storage in an economic manner.

In 1964 the ESL Console was attached directly to the 7094 over a high-speed data channel, the Display Controller actions above being provided by that computer as well as the time-sharing services. Experience showed that it was not reasonable for the time-shared 7094 to provide the real-time services, so in 1967 a PDP-7 was acquired as a Display Buffer Computer to store the picture and to provide these services. These two systems and their corresponding user interfaces are described in this chapter.

B. DIRECT CONNECTION TO THE 7094 TIME-SHARING SYSTEM

1. General Operation

The time-sharing system supervisor resides permanently in one 32K core of the 7094 known as A-core. User programs are swapped into the 32K B-core according to a scheduling algorithm, run for a certain period of time, and swapped out again. The user's program, therefore, has only intermittent access to the machine; so it can not be guaranteed that at any given time his program will be in core. The Display Controller services stated above must be available at any time, whether the user's program is in core or not. Therefore these have been provided as a part of the A-core time-sharing Supervisor, called the DSCOPE module. The three basic functions of the DSCOPE Display Controller are:

1. To store the display file (a sequenced list of display commands) and send them to the console every 30 ms. to maintain the picture.

2. To record all real-time events or attentions. These are placed in a "first-in first-out" list known as the attention queue. Attentions occur when a button is pushed or when the light pen sees a picture part. These attentions are available to the user whenever his program is in B-core.

3. To perform a limited number of real-time functions which currently include rotation, translation, and magnification of pictures under control of the crystal ball or shaft encoders, and the ability to make pictures follow the light pen. The specifications of real-time program operation desired are called real-time instructions. Ideally the user should be able to write his own real-time programs and pass them into A-core for operation, but system security and core space problems make this impossible.

DSCOPE provides a single memory area called the display buffer to be used for storing the display file, the attention queue, and the real-time instructions. The allocation and organization of this buffer is done by the user's B-core program.

DSCOPE presently provides facilities for two independent display console users, or for one user with a display buffer twice as large. In the case of two users, it alternates displaying the two pictures. The Display Controller assures that the picture appears on the appropriate console(s). All user references to the display buffer are in terms of relative locations, starting at zero, so that the same user program may be run on either console without modification.

2. Graphical Output - The Display File

The user's display file (sequence of display commands) must start at relative location zero in the display buffer. When he first logs on to the Display Console, the first word of the buffer is set to an "end of display file" command. The user may then build up his own display file by transmitting display commands into the display buffer starting at location zero displacing the "end of display file" command. In addition to the standard console commands, the display commands include various transfer commands which allow the user to perform display transfers within the display buffer, and to perform subpicture calls and returns. The format and description of these commands are given in Section 4.i.

There are no restrictions on what data may be placed in the display buffer. It is solely the user's responsibility to assure that his display file ultimately terminates with the "end-of-display file" command, that subpictures return properly, etc. If the user wishes to display his picture at either console, the Set C commands should always specify both consoles. The Display Controller will assure that the picture appears only on the console(s) to which the user is signed on. It will also initialize the console with normal (unity) rotation and scale, full intensity, light pen not enabled, and a set point at the center, before starting each frame of the user's picture. The user should not include alarm clock, tracking cross, or "plot control enable" Set C commands in his display file as they will cause erratic behavior which cannot be controlled by the Display Controller.

3. Console Inputs

There are two types of inputs available to the user. The passive inputs, including toggle switches, digi-switches, and analog inputs (globe, knobs) may be read at any time by specific calls on the Display Controller. The occurrence of an active input, light pen see or button push, causes information concerning this action to be placed in an attention queue. When the user's program is brought into core, he may request that the earliest attention in the queue be transmitted into his own core. A user who expects attentions must allocate space in his display buffer for an attention buffer in which the queue may be stored. For present time-sharing response, a length of 100 words generally is adequate. Attentions which occur when the buffer is full are lost. In requesting an attention, the user may also specify that if none are available, his program should be put into "input wait" status until one is available.

Five push-buttons have been assigned certain meaning by the Display Controller:

Button	Function
S	Is an on/off switch for light pen tracking with the tracking cross.
1	Starts a "rubber band line". If the user is tracking, a set-point is placed at the current pen position from which a line will be drawn to the current pen position every 30 ms. Subsequent pushes of button 1 terminate the present line and start a new one.
2	Ends a "rubber band line". Terminates the present line and detaches the tracking cross from the line. Each termination of a line by button 1 or 2 is accompanied by an appropriate attention.
3	Turns the picture on and off.
4	The movie camera may be attached to button 4 by a call to the Display Controller. It may be used as an on/off switch, or as a signal to take a specified number of frames.

4. The DSCOPE Interface

a. Introduction. The DSCOPE Interface is the means by which the user's program in B-core has access to the facilities of the Display Controller module in A-core. It accepts a list of commands sequentially stored in user core and interprets these commands to perform actions within itself. There are presently eight different operations, one of which contains a subset of about twenty suboperations. In essence, the user constructs a program which the controller will run interpretively. The "hardware" which the user sees consists of a Pseudo-ACcumulator (PAC) which he may load, store, and shift at his will and a Control Counter (CC) which is used to specify lengths of blocks for transmission or allocation. Block transfer operations are provided which transmit data across core boundaries either into or out of the display buffer. The burden of maintaining a correct display buffer is placed on the user. Steps have been taken within the Controller to insure that one user is completely separate from the other in order to avoid user conflict by inadequately debugged programs. Any error detected in the running of the display file for a user (e.g., protection violation) results in that display file being terminated immediately. Any error in the interpretive running of the DSCOPE command list results in the display file being reset to an effectively empty one, for the controller cannot insure the integrity of a file so terminated. In the case of a command list error, there is a state word from which the user can determine the type of error, and the value of the DSCOPE call will be the number of the command in the command list which caused the error.

b. Initialization Operations. The user may sign on to the Display Console with the LOGON operation. Its argument is the number of consoles desired (one or two). Signing on to two consoles means that the user may plot on either or both scopes, and the double size display buffer is available to him. LOGOFF detaches the user from the console(s), turns off his picture, and frees the console for other users.

The user may specify the relative location of the start of attention buffer in the display buffer with the SAB (set attention buffer) operation, after first loading its length into the CC (control counter).

Similarly, the CC and the RLT (real-time) operation specify the length and start of the real-time buffer which contains the real-time instructions specifying real-time actions desired by the user. There is a separate CAMERA operation which attaches the synchronized movie camera to push button 4 as an on/off switch, and may specify that the camera be run for a number of frames. CLATN clears out any waiting attentions in the attention buffer. CLEAR resets display file to empty, stops tracking, and removes any partially completed rubber band line, and may also call CLATN. TRACK turns the tracking cross on or off.

c. Transferring Information. Three block transfer operations are available. XMTBA transfers information from B-core to A-core (the display buffer), XMTAB from A-core to B-core, and XMTAA from one A-core location to another. For each of these, the CC specifies the number of words to be moved, and the command specifies FROM and TO addresses. All A-core addresses are relative locations in the display buffer.

d. The Pseudo-ACcumulator (PAC) and the Control Counter (CC). The operations LAC, DAC, SHA allow the user to load, deposit, and shift the pseudo-accumulator. It may be loaded from either core, and stored into either core. It is possible to store only the address, tag, or decrement of the PAC. The LCC operation loads the control counter.

e. Reading Inputs. The ATTN operation transmits the first item on the attention queue to the user. It allows the user to specify whether or not he wants to go into input wait if no attention is available. The formats of attentions are given in Section 4.j.

The passive inputs may be read at any time by the operations DIGI, TOGA, TOGB, ANALOG, CROSS. The exact meaning of the values of these inputs is given in the hardware description.

f. Calling Sequence. The DSCOPE supervisor entry is called with the following sample sequences:

```
(AED-0)       NERR = DSCOPE(COMLST,STATE) $,

(MAD)         NERR = DSCOPE. (COMLST(N),STATE)

(FORTRAN)     NERR = DSCOPE(COMLST(N),ISTATE)

(FAP)         TSX  DSCOPE,4
              TXH  COMLST
              TXH  STATE
              STO  NERR
              CLA  STATE
              TMI  ERROR
NERR          BSS  1
STATE         BSS  1
SIZE          BSS  1
DISPLA        OCT  ....                     Display Commands
              OCT  ....
              OCT  ....
              .... ....
              OCT  377777377777             End of file
 DSZ          EQU  *-DISPLA
 COMLST       OPR  1,,LOGON                 Logon with super-
                                            visor
              DAC  SIZE,0,1                 Save Display space
                                            size
```

```
LCC      DSZ              Set Control Counter
XMTBA    DISPLA,,0        Move display to dis-
                          play space location
                          zero
...
...
OPR      ,,END            End of command list
```

where <u>COMLST</u> is the start of the command list (forward-stored array; therefore COMLST(N) for MAD and FORTRAN) and <u>STATE</u> is a word reflecting the user's state at the completion of the commandlist. (See Section 4. k for the interpretation of the bits in this word.) NERR will be the number of the command in COMLST (starting at zero) which caused the error, whenever there is an error.

g. <u>Display Operations</u>. The general format of the command words to DSCOPE is shown below.

S 2	3 17	18	19 20	21 35
OP	D	IND	C	A

The three bit <u>OP</u> specifies one of the eight operations. The OPR set uses the <u>D</u> field as an extension of the OP code. The <u>IND</u> bit indicates that the instruction is to be indirectly addressed. If IND is on when DSCOPE expects a display buffer address, the B-core location is used for the indirect addressing and the final result is interpreted as the display buffer address. The configuration bits <u>C</u> select options on several operations.

h. <u>Detailed Description of DSCOPE Commands</u>. There follows a detailed description of the operations. All references to locations in user core are checked for legality (before and after indirect addressing) as are the limits of the <u>XMT</u> class commands. Any error in the relocation and protection of these addresses results in an error with <u>PROTB</u> bit set in the state word. All display buffer (A-core) address references refer to relative locations in the user's display buffer starting at zero. Any error will return with the <u>PROTA</u> bit set. Where the symbol "(<u>IND</u>)" is used, it means that if the indirect bit is present, indirect addressing will occur.

OPR(0) - Operate

The operate class is used to encode a number of different functions. These are specified by the decrement (D) of the command. These are the legal D codes and their functions:

(0) - END. Signifies the end of the command list.

(1) - LOGON. The address (IND) contains the number of consoles desired by this program (one or two). If the number of consoles requested is not available, an error return is made with the FULL bit set in the state word. On a successful LOGON the user pseudo-accumulator (PAC) is set to the length of the display buffer. If the user is already logged on, he will be logged off and on again. LOGON performs the additional functions CLEAR and CLATN to obtain a complete reset of display status.

(2) - LOGOFF. This operation logs the user off and returns his allocated consoles to the pool. If he is the only user, the console is removed from service and further interrupts are ignored. The address, IND, and configuration are ignored.

(4) - SAB. The address (IND) specifies the location in the user's display buffer where the attention buffer will begin. The length of the attention buffer is set by the user's control counter (CC). If the user does not establish a long enough buffer for his expected rate of interaction, the buffer will overflow, and further attentions will be lost until he makes space in the buffer by calling ATTN. If the address is zero, the current attention buffer will be destroyed and no further attentions will be accepted.

(5) - ATTN. The address (IND) specifies the B-core address of a forward-stored array where the earliest attention is to be returned. If there are no attentions present and the configuration bits are zero, the user will leave working status and be put into input wait. If there are no attentions and configuration bits are nonzero, the Display Controller will return the first word as a zero indicating that the attention buffer was empty, and the user program will continue to run. For the format of the attentions see Section 4.j.

(8) - CLEAR. This command performs the following functions: (1) resets the display file to nil; (2) stops tracking and wipes out anything in the "rubber-band" buffer; and (3) if the address is zero, also resets any attentions in the attention buffer (CLATN).

(9) - <u>CLATN</u>. This command clears out any waiting attentions present in the attention buffer.

(10) - <u>REALT</u>. This command indicates to the Display Controller that various real-time actions are desired. The address (<u>IND</u>) points to a starting location in the display buffer of an array which contains the real-time instructions. Each instruction specifies what action is desired on what display commands with what triggers. The length of this buffer is specified by the contents of the control counter (<u>CC</u>). The available functions and the formatting of real-time instructions are detailed in Section 4.ℓ. An address of zero terminates all real-time actions and frees the space occupied by the real-time buffer.

(11) - <u>PICTUR</u>. The address (IND) is used to turn the picture on or off. If it is nonzero the picture will be run; if zero, stopped.

(12) - <u>TRACK.</u> The address (IND) will turn the tracking cross on if nonzero or off if zero. If the configuration bits (19 and 20) are zero the cross will appear in the upper right hand corner of the screen; otherwise the pseudo-accumulator (PAC) specifies the h coordinate (bits 8-17) and the v coordinate (bits 26-35) for the "light pen track" command.

(13) - <u>CAMERA</u>. The address (IND) contains the count of frames to be taken each time button 4 is pushed. If the configuration bits (19 and 20) are zero, the camera will be started on, taking the number of frames specified, then stopping, repeating on each push of button 4. If button 4 is pushed while the camera is running, it is stopped and the frame count is reset. A count of zero turns the camera off and disconnects button 4 from it.

(14) - <u>RDSTAT</u>. Read display status of user. The address (IND) points to the beginning of a forward-stored array in B-core into which the display program parameters will be read as follows:

	0	Length of display buffer
	1	Relative location of start of attention buffer
	2	Length of attention buffer
	3	Relative location of start of real-time buffer
	4	Length of real-time buffer
	5	Camera frame count

The CC contains the maximum number of parameters to be transmitted into the array.

(20) - DIGI. The contents of the decimal switches is read into the user's pseudo-accumulator (PAC).

(21) - TOGA. The contents of the upper bank of toggle switches is read into the user's pseudo-accumulator (PAC).

(22) - TOGB. The contents of the lower bank of toggle switches is read into the user's pseudo-accumulator (PAC).

(23) - ANALOG. The contents of the globe and knobs is read into the user's pseudo-accumulator (PAC).

(24) - CROSS. The position of the tracking cross is read into the user's pseudo-accumulator (PAC). If the user is not tracking, the value read in by CROSS has no meaning.

(30) - TYPE. Reserved for typewriter. Not presently connected.

LAC (1) - Load Accumulator.

This operation loads the user's pseudo-accumulator (PAC) from the address (IND). If the decrement is zero, the address is interpreted as a display buffer location, otherwise B-core.

DAC (2) - Deposit Accumulator

This operation deposits the user's PAC into the address (IND). Core references are as in LAC. In addition, the two configuration bits (19 and 20) are interpreted as follows:

00	store whole word,
01	store address only,
10	store decrement only, or
11	store tag only.

SHA (3) - Shift Accumulator

The PAC is shifted the number of places in the address (IND). If the decrement is nonzero, the shift is made to the left, otherwise to the right. The PAC is a logical accumulator: i.e., the "sign" bit is shifted.

XMTAA (4) - Transmit A to A.

This command is a block transfer of data from one location within the user's display buffer to another. FROM location is specified in the address, TO in the decrement. IND is ignored. The number of words moved is specified by the control counter (CC).

XMTAB (5) - Transmit A to B.

This command is a block transfer from the user's display buffer to B-core. It is similar to XMTAA with the following exception: the B-core address may be indirected by IND.

XMTBA (6) - Transmit B to A.

Similar to XMTAB. The B-core address may be indirected by IND.

LCC (7) - Load Control Counter

The address (IND) is placed in the user's control counter (CC).

i. Transfer Commands in the Display File. The console "End of File" command (Prefix 3, Tag 3) has been coded in address and decrement to provide various transfer commands. All locations are relative display buffer locations. These commands enable the user to have his display file stored other than sequentially in the display buffer, to have subpictures, etc. Full responsibility rests with the user for employing the transfer commands correctly and for assuring that the display file terminates with the "end of display file" command.

1) Unconditional transfer:

3	D	3	0

The next console command will be taken from location D.

2) Subpicture call:

3	D	3	A

The next console command will be taken from A. The location D is placed on a push down stack. If D is zero, the location of the call command +1 is pushed down.

3) Subpicture return:

3	0	3	0

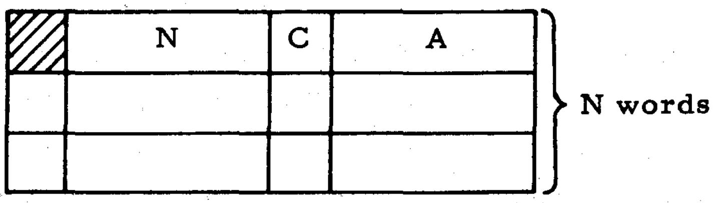
The next console command will be taken from the location specified by the top of the push down stack. If the stack is empty, this will be treated as an end of display file.

4) End of display file:

3	77777	3	77777

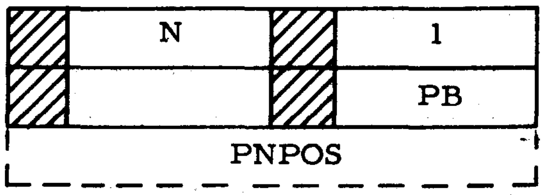
This terminates the display file for this user.

j. Attention Formats. The attention buffer, the location and length of which are specified by the SAB command, contains the attentions until the user's B-core program requests them. Storage in the buffer is managed by a free storage system. Attentions are provided to the user, one at a time, by the ATTN command. Each attention has the following format:



where N is the number of words used to specify this attention and A is the number of the attention. C is used by the supervisor and contains the number of words in the attention display for this attention. The next N-1 words vary with the individual attention numbers.

1) Push-button:



N is the count of this attention (two, or three if tracking). PB is the number of the button pushed and PNPOS is the position of the cross (if tracking).

2) Light-pen see:

▨	N	▨	2	
			LOC	
	H		V	
	CALL		RETURN	
	CALL		RETURN	
	CALL		RETURN	

LOC is the relative location in the display buffer of the display command which produced the light seen by the light pen. H, V are the coordinates of the light which interrupted the display. The next N-3 words contain the pushdown stack of subpicture calls which enables the program to analyze the sequence which caused the trap. CALL is the relative display space location of the sub-picture call. RETURN is used by the supervisor to return from the subpicture. These are ordered outer-most to innermost.

3, 4) Typewriter:

Attention numbers 3 and 4 are reserved for the type-writer which is not presently connected.

5) Line completed:

▨	3	▨	5	
3	H	4	V	BAND
0	ΔX	0	ΔY	

This indicates that a line has been drawn on the screen, via pushbuttons S, 1, and 2. BAND and BAND+1 contain the setpoint and lineplot console commands which produced the rubber band line.

k. State Word Bit Assignments. The STATE word indicates the state of the user at the completion of any particular DSCOPE call. The bits of the left half are error bits, and indicate on an error return what conditions caused the failure. These bits are cleared out on each separate call to DSCOPE. In case of an error, the sign bit of the state word will be set and the accumulator upon return from DSCOPE will contain the relative position in the COMLST (starting at zero) of the command causing the error.

Status bits (right-half):

Bit No.	Octal	
35	1	- User has specified real-time actions (REALT).
31	20	- User is successfully signed on.
30	40	- User is master (0) or slave (1) console.
29	100	- Master console is in use.
28	200	- Slave console is in use.
27	400	- User is in input wait status (Never available to user).
25	2000	- User has specified an attention buffer.
24	4000	- User has a nonempty typewriter buffer. (Not operative)
23	10000	- User's attention buffer is not empty.

Error bits (left-half):

Bit No.	Octal	
0	400000	- Error present in <u>STATE</u> word.
17	1	- Protection error - display buffer. (PROTA)
16	2	- Protection error - core B. (PROTB)
15	4	- ESL Console appears not to be operable.
14	10	- Number of requested consoles not available.
13	20	- Illegal operate class instruction. (ILLOP)
12	40	- Not at time-sharing console authorized to use display.
11	100	- More than 100 Commands specified.

l. <u>Real-Time Instruction Formats</u>. As outlined earlier under the REALT operation in the OPR class, the user specifies the starting location and length of a real-time buffer within his display buffer. Each word in this buffer contains a real-time instruction, and is examined upon the occurrence of any "real-time" change, i.e., a movement of the knobs or globe, or upon other trigger actions specified below. If the "trigger" portion of the instruction matches the trigger code of the action which occurred, a transfer is made to the specified "function" which decodes the remainder of the instruction according to general specifications outlined below and takes the appropriate action.

The format of the real-time instruction word is below:

S 3	4	5	6 16	17 20	21 24	25 35
T		D	TMP	F	S	LOC
4	2		11	4	4	11

T - Four bit trigger code for this instruction

D - The direction of the speed (0 for plus, 1 for minus).

TMP - Starting location of temporary array (if required).

F - Four bit function code.

S - Four bit speed multiplier.

LOC - The display buffer location of the affected display command(s).

The following "trigger" codes are available:

1 - Leftmost knob has changed position

2 - Center knob has changed position

3 - Rightmost knob has changed position

4 - Globe rotated about the x axis

5 - Globe rotated about the y axis

6 - Globe rotated about the z axis

7 - Tracking cross has moved

The following "function" codes are available:

(0 and 8) - No Operation

An operation may be temporarily discontinued by zeroing the tag of the control word (using the store tag option of DAC).

(1) - Rotate about Z axis. LOC in the control word points to a relative position in the user's display buffer which is the start of a pair of rotation matrix commands (Prefix 5 and 6) for the Console. These commands will be altered by the Display Controller as the real-time actions are performed to give the rotation effect. TMP is a pointer to a ten word temporary array in the display buffer (but not in the real-time buffer) which is used for scratch purposes by the programs performing the rotation. In this case the first nine words of the ten word array contain the 3 x 3 matrix required for rotation. The

tenth word is the scale of the drawing for magnification purposes. Each of these numbers is a signed 35-bit binary fraction. Unity is represented by 377777777777(8). It is the user's responsibility to initialize these ten locations to their proper values for correct operation of the real-time rotation and magnification. The correct machine representation of these words for a unity matrix at the beginning of real-time operations would be:

i_h	377777777777
j_h	0
k_h	0
i_v	0
j_v	377777777777
k_v	0
i_d	0
j_d	0
k_d	377777777777
S	377777777777

$$S \cdot \begin{bmatrix} i_h & j_h & k_h \\ i_v & j_v & k_v \\ i_d & i_d & k_d \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} h \\ v \\ d \end{bmatrix}$$

This calculation converts lines from XYZ coordinates to horizontal, vertical, and depth scope coordinates.

(2) - Rotate about X axis. Similar to the rotation about the Z axis. For correct operation in the rotation about any axis for one object, the TMP, LOC, and the initialization of the temporaries should be the same.

(3) - Rotate about Y axis. Similar to rotation about X and Z axes.

(4) - Translate up and down. LOC points to a Set Point command which will be altered to provide translation of correctly structured pictures (i.e., all whose lines are relative to this one Set Point Command). TMP is not used and no temporary space need be assigned.

(5) - Translate right and left. Similar to translate up and down.

(6) - Magnify. LOC points to a pair of rotation matrix commands as in the rotation operations. TMP points to the same type of

temporary array as in rotation. If the user wishes to magnify beyond the original scale, he must include a Set C command in his display file (logically) preceding the rotation commands. A Set C real-time instruction must appear in his real-time buffer preceding the magnify instruction. The Display Controller will change the Set C word to specify double bit spacing and change the scale appropriately. This Set C code applies for all following magnification calls. The unused bit in the real-time instruction is used internally by the Display Controller to control magnification.

(7) - Set C Control. As mentioned above, LOC points to the location in the user's display file of the Set C control command which controls the object under alteration. TMP, S, and D are ignored. If LOC is zero, no larger than original magnification will take place for future magnifications and only scale reduction will be allowed.

(9) - Light Pen Drag. LOC points to a Set Point Command which will be altered to be positioned at the location of the tracking cross when the user is tracking. TMP, S, and D are ignored. This function should be used only with trigger code 7.

C. CONNECTION THROUGH A DISPLAY BUFFER COMPUTER

1. General Operation

The design philosophy for the two-computer system is presented in "The Design and Programming of a Display Interface System Integrating Multi-Access and Satellite Computers." (Appendix A) For purposes of compatibility with old programs, it was decided that the first programming effort would be to provide the DSCOPE Interface on the PDP-7/7094 system. After that interface was completed, work was begun on the new Display Interface System employing minimal executives. These two schemes are both described in the following sections.

2. The DSCOPE Interface

The DSCOPE module was divided into the program interpreter and the real-time operation section. The interpreter was kept on the 7094 and the real-time programs were implemented on the PDP-7. The minimal executive (see Appendix A) for each machine was designed and implemented. The main function of each executive is communication with the other machine. The satellite computer executive also processes interrupts and keeps the machine and display running. As far as the user is concerned, the major difference is the presence of the PDP-7 near the console and the necessity to be sure that it is loaded and running properly.

The changes to the DSCOPE interface presented in Section B are few. The display buffer is, of course, located in the PDP-7, so XMT operations deal with 7-core rather than A-core. Error bits for the state word have been changed to reflect the existence of the PDP-7, as follows:

4	PDP-7 appears not to be operable.
2000	Error sending to PDP-7. Unsuccessful after five tries.
4000	Error receiving from PDP-7.

The DSCOPE interpreter in the 7094 has become a B-core routine rather than the A-core/B-core interface, so it must be called simply with a TSX instruction (not a TIA). This DSCOPE then makes the proper calls on the minimal executive, at least part of which is in A-core, to effect the functions required by the user's program.

3. The Minimal Executive in the 7094

a. General Description. The following description of the minimal executive is provided for completeness. It describes neither the present executive, nor any projected executive (hence the absence of specific details of how to call the MINEX routines). This section should, however, present the essential functions of the executive and a way in which they could be implemented.

As previously mentioned, the minimal executive in the 7094 has the function of communicating with the PDP-7. The basic items communicated are called control blocks and data blocks. A control block consists of three 36-bit words in each of which the left-half is the complement of the right-half for error-checking purposes. The control block sent from the 7094 to the PDP-7 requests some action from the PDP-7 executive, such as receive data, call a routine, etc. A data block consists simply of full 36-bit data words. Transmission of data blocks is checked by preceding them with a control block giving a checksum.

The first word of the control block specifies a function in 3 bits, and an argument in the next 15 bits. The two other 18-bit quantities are arguments. The format of the control block is:

$\bar{F}$	$\bar{A}_1$	F	A_1	
	$\bar{A}_2$		A_2	36-bit words
	$\bar{A}_3$		A_3	

The function code F=0 is reserved for signalling the beginning of data transmission to the PDP-7. In this case A_1 is the address (possibly interpreted) to which the data is to be sent. A_2 is the number of 36-bit words in the data block, and A_3 is the 18-bit one's complement checksum of the data block. Other function codes have meanings according to the specific version of the executive in the PDP-7. The function code assignments used in present systems are given in Section 3. c and 3. d.

b. <u>The MINEX Interface</u>. There are five basic calls on the Minimal Executive:

SC(PTR) - SEND CONTROL BLOCK

The exec reads the three words from the forward-stored array specified by PTR, makes the left-half word be the complement of the right half-word, and sends the control block to the PDP-7 executive. It waits for an OK or a NOGOOD response from the PDP-7. In the case that the control block is no good because of an apparent transmission error, the 7094 exec tries again up to five times. If NOGOOD for some other reason (the control block requests a function which does not exist in the PDP-7), no further attempt is made and an error return is immediately made to the user.

<u>SCD</u>(TO, N, FROM) - SEND CONTROL AND DATA BLOCK

A control block with

$F = 0$

A_1 = TO(15 bits) (Destination in PDP-7 for data)

A_2 = N

A_3 = ones comp. checksum of data block (18 bits)

is sent to the PDP-7 followed by the data block of length N from address FROM in the 7094 (B-core). The exec then waits for the OK or NOGOOD from the PDP-7 and responds as above. Both control and data are sent if there is a retransmission.

<u>RC</u>(PTR) - RECEIVE CONTROL BLOCK

It is assumed that the user has sent some message to the PDP-7 to which it should respond with a three-word control block. A control block is received from the PDP-7 and stored in the three-word forward array specified by PTR. If the left half of each word is not the complement of the right half, the exec requests the PDP-7 to send it again up to 5 times. If it is bad, or if the PDP-7 does not respond within a certain amount of time, an error return is made to the user.

<u>RCD</u>(PTR, TO, NMAX) - RECEIVE CONTROL AND DATA

It is assumed that the PDP-7 is supposed to send a control block and a data block. The control block is assumed to be of the format

F = 0

A_1 = don't care

A_2 = N

A_3 = checksum

The control block is read into the three-word array starting at PTR and, if N is not greater than NMAX, N data words are read into an array starting at the address TO. The exec verifies the checksum, and if it is wrong, requests retransmission from the PDP-7 up to five times. In the case of no message, five errors in transmission, or N greater than NMAX an error return is made to the user.

Note: In connection with the receive calls RC and RCD, since the user in the 7094 sends a message to his PDP-7 program requesting a transmission back, and since transmission of data is directly to user core, the user's program must be guaranteed to remain in core between the two actions. This is accomplished automatically by MINEX which controls the time-sharing supervisor to allow the user to construct a command list (similar to that used by DSCOPE) which is executed to completion before his program is swapped out of core.

EDI (LOC, DISMISS) - ENABLE AND DISMISS INTERRRUPT

The final function needed is provision for messages to the user's 7094 program generated by the PDP-7 program but not by specific request from the 7094. The 7094 program has no way of knowing when any such messages will come, so it cannot anticipate them. A call to EDI sets a location LOC in the user's program to which transfer of control will be made whenever the PDP-7 interrupts to say that it has a message waiting for the 7094. The next time the user's program is run after such an interrupt, MINEX will disable itself for further interrupts, interrupt the user's program, and transfer control to LOC. (MINEX remembers the place from which the user was interrupted as UINT, for later continuation.) At LOC, the user program may then send a control block to the PDP-7 requesting that the waiting message be sent to him. When he has finished processing the message received from the PDP-7, the program at LOC should call EDI again to re-enable for another interrupt. MINEX's response to this EDI call will be to reset its status and re-enable for the next PDP-7 message, then to return to the previously interrupted user activity which it remembers as UINT. If DISMISS is zero in the EDI call, there will be no return to the interrupted activity. (This normally would be used only for the initial enable for PDP-7 messages.) If LOC is zero, messages from the PDP-7 will not be read. If a PDP-7 message becomes available while MINEX is disabled (either the user has not yet enabled, or is currently processing a previous message), the new message will be saved until an EDI call re-enables.

In an alternate scheme which eliminates EDI, MINEX simply places messages from the PDP-7 directly into the user's input buffer along with his teletype input. These messages would be distinguishable from teletype messages. Such messages would have to be of limited length; for long messages, a short message could be sent informing the user of the existence of the long message which he could then request from the PDP-7 by SC and RCD calls.

c. Control Blocks from 7094 to PDP-7. The following function codes have been assigned for control blocks in the PDP-7 executives which have been built so far. Any of them could be changed by a user who is willing and able to write his own sufficiently reliable executive. The meanings of the three arguments for each of the function codes is given.

Name	Function Code		Description and Arguments
XMTB7	(0)		Send data block from 7094 B-core to PDP-7.
		A_1	is the "TO" address in the PDP-7 for the data. It is presently an absolute address.
		A_2	is "N", the number of 36-bit words in the data block.
		A_3	is the one's complement checksum of the 2N 18-bit words in the message.
XMT7B	(1)		Send data block from PDP-7 to 7094 B-core.
		A_1	is not used.
		A_2	is "N" as for Code 0.
		A_3	is the "FROM" address in the PDP-7.
XMT77	(2)		Move data within PDP-7.
		A_1	is "TO" address.
		A_2	is "N" - number of 36-bit words to be moved.
		A_3	is "FROM" address.
			If the two locations of the block overlap, the move routine assures that data is not garbled by moving last word first if necessary.
CALL7	(3)		Call a subroutine in the PDP-7.
		A_1	is the address or "name" of the subroutine to be called.
		A_2	is the first argument (value) for the subroutine.
		A_3	is the second argument (value) for the subroutine.
			The subroutine will be called with the sequence

```
      A2   in the  AC
     JMS   Subroutine
           Pointer to A3
```

The following subroutine "names" (numbers) are defined in the present executive. They perform the DSCOPE functions of the same name and arguments.

0	Sgnoff
1	Sgnon(N)
2	Clear (CLATSW)
3	Clatn
4	Satbuf (LOC, N)
5	Realt (LOC, N)
6	Camera (NOTNOW, N)

The present versions of the executive expect absolute PDP-7 addresses but these could as well be relative addresses in the user's area depending on which machine performs PDP-7 storage allocation in CALL7 "name".

The minimal executive is so structured that the system builder includes with it only such function programs as he requires. For instance, if he does not need the XMT77 function, he does not need to load it; if he then tried to use it, a fatal error would occur.

d. Control Blocks from the PDP-7 to the 7094. The only control block presently assigned precedes data blocks from the PDP-7. The format is

Function Code	Arguments
(0)	A_1 is ignored by MINEX
	A_2 is N, the number of 36-bit words
	A_3 is 18-bit, one's complement checksum

The reason for specifying N is that some messages may be created in the PDP-7, the length of which will not be known to the 7094. To receive such a message, the user program in the 7094 would send a control block (SC) with CALL7 function to request that one of these messages be sent back, then wait to receive control and data back from the PDP-7.

CHAPTER III

GRAPHSYS - A PROGRAMMING SYSTEM FOR THE ESL DISPLAY CONSOLE

A. INTRODUCTION

GRAPHSYS is a set of procedures for programming interactive display consoles on time-sharing systems. The system was originally written for the ESL Display Console connected to a time-shared 7094, but the user interface which it provides has also been used for programming the ARDS (Advanced Remote Display Station) storage tube units. Details of this document concerning specific display facilities are based on the ESL Console; further memoranda will detail the differences which arise in applying GRAPHSYS to other display consoles.

Throughout this chapter, the part of the time-sharing Supervisor and the executives of any satellite computers involved in running the display units will be called the Display Controller, or Controller, for short. The GRAPHSYS procedures, which provide a high-level language for programming the display console, reside in "B-core" of the time-sharing system along with the user's programs. They make calls on the Controller to effect the actions desired. (See Fig. 3.1)

The special problems associated with operating a display console in time-sharing, when the user has only intermittent access to the computer, have determined the division of tasks between GRAPHSYS and the Display Controller. The Controller performs the real-time functions listed in Chapter II.A. It stores the display file (an ordered sequence of display commands) and outputs them to the display unit to maintain the picture; it records attentions (real-time events such as pen "see") as messages for the user in an attention queue; and it performs certain real-time functions (e.g. rotation) as requested by the real-time instructions created by GRAPHSYS in response to user calls.

The Controller provides a single memory area called the display buffer for storing the display file, the attention queue, and the real-time instructions. The allocation and organization of this buffer is one of the tasks performed by GRAPHSYS. The user need only

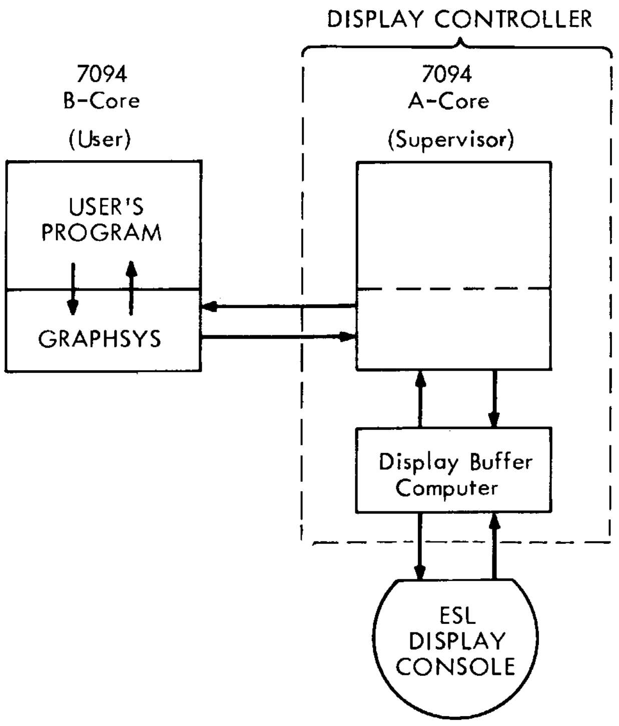


Fig. 3.1 Information Flow in Display System

specify the number of registers to be used for the attention buffer (which contains the attention queue) and the real-time buffer (which contains the real-time instructions). The remaining space stores the display file proper, (and rotation matrix buffers, if the real-time rotation or magnification functions are used). (See Fig. 3.2)

Flexible means are provided by GRAPHSYS for creating and editing a display file. Procedures for adding, removing, and replacing console commands, as well as a "copy" function whereby identical sets of commands may be reproduced within the display file, are included in the package. An important feature is the ability to define subroutine pictures or subpictures. Their definition is analogous to the way computer program subroutines are defined, and each time a display of a subpicture is desired, only a single command (the call command) is added to the display file. No provision is presently made for arguments to subpictures.

Each item added to the display file is assigned a name. All communication between the user and the system about items then

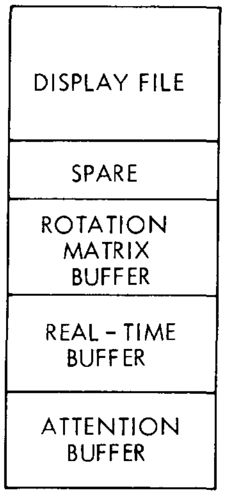


Fig. 3.2 Diagrammatic view of the Display Buffer

takes place in terms of these names, the system automatically performing the required transformations for communication with the Display Controller. The names remain invariant even though the commands which they represent may be moved in the display file. Thus GRAPHSYS provides a form of automatic storage allocation for the display file.

A set of procedures for adding standard picture parts to the display file is also provided. These include arcs of circles, lines, points, set-points, rotation matrix commands, control commands (affecting picture magnification, light pen sensitivity, etc.), and characters. A set point differs from a point in that its position remains fixed during rotations. A picture that is to be rotated consists of a single set point followed by a series of connected incremental vectors. A point is merely a unit length vector, (one scope increment in the plus direction followed by one in the negative direction so that the beam position is unaltered). The command which controls parameters such as intensity, sensitivity to the light pen, etc., is called a Set C command. The control command which increments the beam position directly, and is unaffected by rotation is called a Set F command.

The various facilities of the Display Controller may be called upon via GRAPHSYS. These include signing on and off with one or two consoles, placing the user's program in "input wait" when an attention is requested and there are none, reading the current values of the passive inputs, and requesting operation of the available real-time programs.

The GRAPHSYS procedures are written entirely in the AED-0 (ALGOL Extended for Design) language,[1] and use the AED-0 "Free Storage" System[2] to allocate memory space dynamically for the B-core data structure. If the user employs free storage to store his own data structures in "plex" form, he may call the same free storage procedures, which are automatically loaded with GRAPHSYS.

B. THE DISPLAY FILE, OBJECTS, AND NAMES

The display file is the ordered sequence of display commands which is sent to the display console to produce a picture. Although the console deals only in terms of simple display 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 as a unit, and are to be thought of as an atomic entity, e.g., an arc of a circle consisting of several short, straight-line segments (made by several line-generate commands).

When an object is placed in the display file, it must be given a unique name if it is to be identified again. The user may wish to specify some object on 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 simplest type of unique name which can be assigned to an object is the starting location in the display buffer of its console commands. Every object is, of course, uniquely specified by this number. These locations are chosen by GRAPHSYS, which automatically performs the tedious tasks of allocating space for the display file and organizing it. This is a dynamic process in which, without altering the display sequence, commands may be moved around in the display buffer (e.g., when new commands are inserted into the middle of the display file). Display buffer locations, therefore, do not constitute a suitable naming scheme for the user, since the user could be forced to keep his own data structure continually updated as GRAPHSYS moved objects and thereby changed the names.

The simplest form of invariant name that can be assigned to an object is an integer number. If this type of name were chosen, GRAPHSYS would have to build an array of length equal to the total number of names. The name of an object would be the index for the position in the array containing the actual display buffer locations. In addition, the user would need a similar array given the item in his data structure corresponding to the name. This is a lot of mechanism and requires the existence of large arrays. It would be much simpler if the user and GRAPHSYS had a mutually convenient single name for each object.

The naming scheme which has been employed by GRAPHSYS meets these criteria of uniqueness, invariance, and convenience. GRAPHSYS builds its own data structure (a string of items arranged in display file sequence) in storage supplied by, and thereby known to, the user. Each item in the string constructed by GRAPHSYS represents an object, and specifies its current display buffer location. The register supplied by the user, in which GRAPHSYS stores its information is called the display register of the object. The name of the object is a pointer to (i.e., the absolute address of) its display register. and is called the display pointer name. The user communicates with GRAPHSYS in terms of these names, and the system automatically performs the transformation to and from display buffer location for communication with the Display Controller. (See Fig. 3.3)

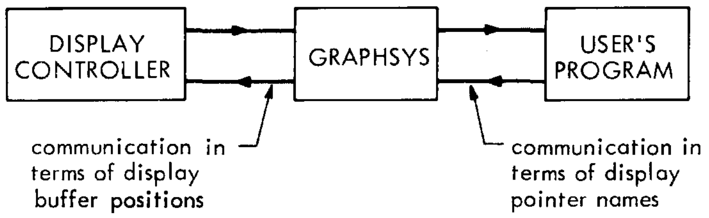

DISPLAY CONTROLLER
GRAPHSYS
USER'S PROGRAM
communication in terms of display buffer positions
communication in terms of display pointer names

Fig. 3.3 Communications about Items in the Display File

This scheme both guarantees a unique name for each object, since the core locations of the display registers are unique numbers, and also eliminates the need for large duplicate arrays, since the display registers can be obtained along with space for user data from free storage[2] as needed. Finally, this scheme provides a mutually convenient name for the user and GRAPHSYS by allowing the user to supply the display register as a part of his own data structure. Figure 3.4 illustrates the combined user-and-GRAPHSYS data structure for a display file containing three objects.

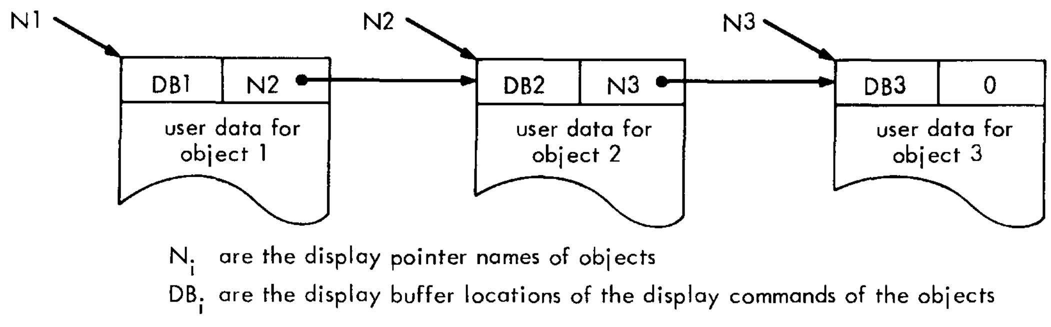

N1
N2
N3
DB1
N2
user data for object 1
DB2
N3
user data for object 2
DB3
0
user data for object 3
N_i are the display pointer names of objects
DB_i are the display buffer locations of the display commands of the objects

Fig. 3.4 The Combined User-and GRAPHSYS Data Structure

Each display register contains the display pointer name of the next object in sequence in the display file. A name of zero signifies the end of the list.

The contents of the display registers are maintained solely by GRAPHSYS; the user does not have to be concerned with the display register string or the display buffer locations.

There are two types of items other than objects which have display registers and display pointer names -- subpicture definitions and subpicture calls. (The process of defining and using subpictures is detailed in Section D.3.) The display pointer for these items must point to the first of <u>two consecutive</u> display registers which are available to the system. The display registers for a subpicture call occur as an item in the middle of a string of displayed objects and/or other calls. The second register specifies the name of the subpicture called. The display registers of a subpicture definition start a separate string linking together the objects (or other calls) of which the subpicture is composed. The second register for the definition specifies the number of calls on the subpicture and forms a string of all subpicture definitions. Additional bits in the display register specify whether the register(s) was supplied by the user or by GRAPHSYS, whether the object is a subpicture call, or definition, or not, and whether or not the object is connected to a real-time program.

C. FORMAT AND CONVENTIONS

Sections D through J describe the various facilities provided by GRAPHSYS. Section K gives a detailed description of the procedures, including their calls, values, and operation. The AED-0[1] calling conventions are used for sample calls on procedures. (The calling sequence in other languages is similar.) Section L presents details of using GRAPHSYS in CTSS and from language other than AED-0. Section M describes the Console Simulator. A sample interactive graphics program using GRAPHSYS is included as Appendix B.

The following conventions are used throughout this manual.

1. Procedure Values

B	Boolean value (True or False).
CC	Console Command (A command for the display hardware).
DPN	Display Pointer Name (15 bits).
N	Number of words.
OB	A 15-bit pointer to an object to be displayed (see Section E).
P	A Pointer.
PI	Passive Input integer value (for devices such as toggle switches).

2. Procedure Arguments

All arguments are of type pointer unless specifically stated.

*	An optional argument that may be omitted.
**	When an optional argument is marked thus, all arguments so marked must either be given or omitted.
0	(zero) The convention used for optional arguments that are to be ignored. E.g., EXAMPLE (ARG1, 0, ARG3) indicates that values for ARG1 and ARG3 only have been given.
#	Integer argument.

≠	Boolean argument. (For AED-0, zero represents False, non-zero (specifically <u>one</u>) represents True.)
+	Real argument (floating point argument).
##	Integer array argument.
__	(underline) Output argument.
- - -	(dashed underline) Input or output argument.

D. THE DISPLAY FILE -- ADDING AND REMOVING CONSOLE COMMANDS

1. General Description

As we have seen the display file is stored in the display buffer provided by the Display Controller. The function of organizing the file and allocating console commands to specific positions in it must be done external to the Controller. GRAPHSYS performs this function automatically when any of the procedures described in this chapter is called. The process is a dynamic one and the stored positions of any particular commands are not constant, although the <u>sequence</u> in which they are sent to the display (specified by the user), remains unaltered. The file organization need never concern the user since the names assigned to items added to the file remain invariant whatever the position of the corresponding commands. An analogy is to be found in CTSS: a user is never concerned with the organization of the disk, since files stored in his tracks may always be referenced by an invariant name.

This section considers those procedures which enable console commands to be added to or removed from the display file. Whereever "NAME" appears as an argument, it is the display pointer name which is being given to the item then being added to the display file. "PNAME", "HERE", and "THERE" are display pointer names of items previously added to the displayfile. An "OBJECT" is any number of console commands added to the file at the same time, which are to be thought of as a single atomic entity, e.g., an arc of a circle consisting of several short straight line segments and built up with line generate commands. "DPN" as the name of a procedure is the display pointer name of the object. It will be equal to "NAME" if the user supplied a name, or will be assigned by the system if not.

Provision is made for defining subroutine pictures, or subpictures. A subpicture definition is represented in the display file by a group of any number of console commands terminated by a special "end" command. To cause a subpicture to be displayed, requires the addition of only a <u>single</u> word to the display file, calling for the subpicture to be displayed at that point in the display sequence. The subpicture call command causes the subpicture definition to be executed, and when the end of the definition is reached, the display sequence resumes following the call command. This facility provides a great saving in display file space for repetitive pictures at the expense of computer time for processing the calls and returns within the display file.

"CALL" refers to a single command added to the display file which causes all the commands of a previously defined subpicture to be sent to the console. An "ITEM" is either an "OBJECT" or a "CALL". Subpicture definitions themselves may contain any number of items, i.e., any number of objects and/or calls.

We now consider the procedures available for creating and editing a display file.

2. <u>The Plot Function</u>

DPN = PLOT(OBJECT)

causes the one or more console commands which OBJECT describes to be added to consecutive positions at the end of the display file. The system will obtain the display register from free storage and return a pointer to this register as the DPN (which should be remembered if the object is to be referenced during later communications between the system and the user.) The user may supply the display register as a part of his own data structure by using the call

DPN = PLOT(OBJECT, NAME)

where NAME is a pointer to a single register which the system may use as the display register for the object. In this case, DPN is set equal to NAME.

If it is desired to insert the object into the display file at some specific position, rather than at the end, the call

DPN = PLOT(OBJECT, NAME, PNAME)

should be used. It will cause the object to be inserted at a position immediately following the commands for item PNAME. PNAME must have been previously assigned by a call to one of the subroutines making additions to the display file (PLOT, CALL, CPY, or RPL). This position refers, of course, to the sequence of the display file and not the physical position of the commands in the display buffer. Thus use of the second name allows editing of the display sequence.

To ignore one optional argument while specifying a later one in the calling sequence, the convention 0 (zero) is used. Hence to insert an object in the display file without giving it a name, write

DPN = PLOT (OBJECT, 0, PNAME)

These several forms of calls on PLOT are summarized using the * optional argument conventions as

DPN=PLOT(OBJECT, NAME*, PNAME*)

3. Subpictures

Subpictures are defined using procedures DEFSUB and ENDSUB which work like BEGIN and END in AED-0, or PL/I. To start defining a subpicture write

DPN = DEFSUB(SNAME*)

where SNAME, if supplied, is a pointer to the first of two consecutive display registers. DPN will be a pointer to the first of the two registers. Subsequent additions made in sequence to the display constitute the definition body of the subpicture. While the subpicture is being defined, it is displayed on the screen. When the definition is complete, the procedure call

ENDSUB( )

causes the subpicture to be "wrapped-up" and it disappears from the screen, although its definition remains in the display file. The process of defining a subpicture is illustrated in Fig. 3.5.

To cause the subpicture to appear on the screen, write

DPN = CALL(SNAME, NAME*, PNAME*)

where SNAME is the name of the subpicture (given with procedure DEFSUB) and NAME and PNAME have the same meaning as for PLOT. The display sequence may be seen in Fig. 3.5(d). Note:

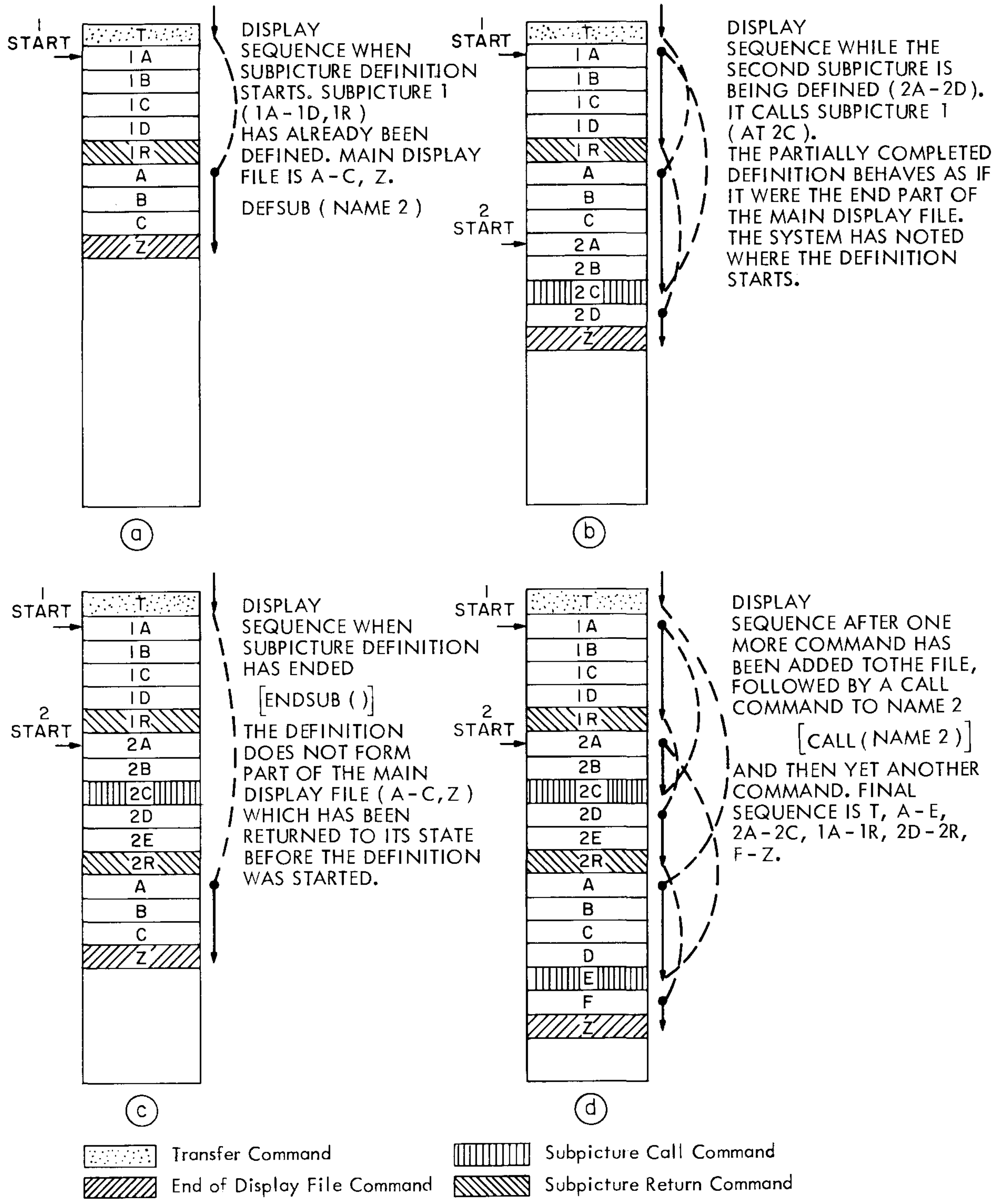

1 START
T
1A
1B
1C
1D
1R
A
B
C
Z
DISPLAY SEQUENCE WHEN SUBPICTURE DEFINITION STARTS. SUBPICTURE 1 (1A-1D, 1R) HAS ALREADY BEEN DEFINED. MAIN DISPLAY FILE IS A-C, Z.
DEFSUB (NAME 2)
(a)
1 START
2 START
2A
2B
2C
2D
DISPLAY SEQUENCE WHILE THE SECOND SUBPICTURE IS BEING DEFINED (2A-2D). IT CALLS SUBPICTURE 1 (AT 2C). THE PARTIALLY COMPLETED DEFINITION BEHAVES AS IF IT WERE THE END PART OF THE MAIN DISPLAY FILE. THE SYSTEM HAS NOTED WHERE THE DEFINITION STARTS.
(b)
2E
2R
DISPLAY SEQUENCE WHEN SUBPICTURE DEFINITION HAS ENDED
[ENDSUB ( )]
THE DEFINITION DOES NOT FORM PART OF THE MAIN DISPLAY FILE (A-C,Z) WHICH HAS BEEN RETURNED TO ITS STATE BEFORE THE DEFINITION WAS STARTED.
(c)
D
E
F
DISPLAY SEQUENCE AFTER ONE MORE COMMAND HAS BEEN ADDED TOTHE FILE, FOLLOWED BY A CALL COMMAND TO NAME 2
[CALL (NAME 2)]
AND THEN YET ANOTHER COMMAND. FINAL SEQUENCE IS T, A-E, 2A-2C, 1A-1R, 2D-2R, F-Z.
(d)
Transfer Command
End of Display File Command
Subpicture Call Command
Subpicture Return Command

Fig. 3.5 Defining a Subpicture - Diagrammatic View

1) Subpictures may be defined within subpictures, to an arbitrary depth (in principle--up to 10, in fact!) They may still be called from outside those subpictures, however. They behave exactly as if they had all been defined in parallel at the highest level.

2) Subpicture definitions may include calls to other subpictures and may be edited by using the optional third argument in PLOT, etc., for controlled insertion. A subpicture may not, however, call itself; i.e., subpictures are not recursive.

3) After a call to ENDSUB the program returns to the state it was in before the previous call to DEFSUB. (Fig. 3.5c). In other words the current end of the display file is effectively the same as it was before that last subpicture was defined. The subpicture definition is "transparent" to the display sequence.

4) Set points should not be included in the definitions of subpictures, so that they may be called for display at any part of the screen. (Including a setpoint command in a subpicture and calling it many times just makes a very bright picture!)

4. The Copy Function

If an identical copy of the console commands which represent some object is required in the display file, then procedure CPY should be used.

DPN = CPY (THIS, NAME*, PNAME*)

will cause an identical set of console commands which describe the object whose name is "THIS" to be placed in the display file. "THIS" must previously have been added to the display file. NAME and PNAME have the same meanings as for PLOT. Note that "THIS" may either be a part of the main display file or part of a subpicture definition.

5. The Replace Function

DPN = RPL(NEW, INAME, NAME*)

replaces one object in the display file with another. The objects can be in any position and need not be represented by the same number of console commands.

The item INAME (previously given with RPL, PLOT, CALL, CPY) is replaced by NEW. NEW may have the same meaning as OBJECT in procedure PLOT, or it may be the name of a subpicture (same as SNAME in procedure CALL) in which case a call will be generated. If NAME is omitted then the NEW item will be given the name INAME, otherwise it will be named NAME.

6. The Remove Function

RMV(HERE*, THERE*)

causes all objects and calls to subpictures from HERE to THERE inclusive to be removed permanently from the display file. If THERE is omitted, then only the single atomic object or call corresponding to HERE is removed.

If HERE is a subpicture name, then that whole subpicture definition will be removed. If THERE is given it is ignored in this case. It is an error to attempt to remove a subpicture to which calls still exist. If no arguments are given, the whole display file and all subpicture definitions are removed.

E. STANDARD AND NONSTANDARD OBJECTS

1. General Description

In Section B an OBJECT was described as any number of console commands added to the display at the same time, which are to be thought of as a single atomic entity. Objects are added to the display file with PLOT or RPL which require the console commands to be arranged in the format shown in Fig. 3.6

A single name is assigned to an object, to be used for communications about that object between the user's program and GRAPHSYS. There is no way to refer to individual commands within the object, since they have no individual name. If they need to have a name, they must be added to the display file separately with their own name.

If the return code bit (see Fig.3.6) is nonzero, then after the commands have been added to the display file, PLOT (or RPL) returns

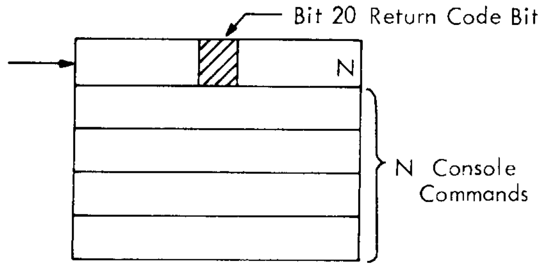


Fig. 3.6 Format required for adding console commands to the display file with PLOT or RPL

the element that contained them to free storage in B-core.

2. Standard Objects

Several pointer procedures are provided for building the most common or "standard" objects. These may be used as the first argument of PLOT or RPL, since the value of each of them is a pointer to an element with the format shown in Fig. 3.6. For example, to add a line to the display file, use LIN and write:

PLOT (LIN(DELX#, DELY#, DELZ#*), NAME*, PNAME*)

The names of these procedures is given below. (See Section K for the detailed descriptions.)

On calling the procedure SGNON to sign on to the display unit, a condition is set whereby all the procedures building standard objects insert the return code bit. This situation may be reversed or restored again by calling RTNCOD(ONOFF≠).

SETPT	A set point.
LIN	A line in any plane.
PNT	A point.
CIRCLE	An arc in the XY plane made up from short straight-line segments.
CIRC3D	An arc in any plane.
ROT	A pair of rotation matrix commands. (Must precede objects which are to be rotated.)
SETC	A Set C command. (To indicate control information to the display.)

SETF	A Set F command. (To indicate control information to the display.)
SINGLE	A single character of the "unpacked" type.
SPECIAL	A "special" character designed by the user to be plotted incrementally by the character generator.
PACKED	"Packed" characters from a .BCQ. string.
TEXT	"Packed" characters from a .C. string.

Examples of Use:

PLOT (PNT( ), NAMEX) adds a point to the file.

RPL (NEVIS(LIN(ΔX,ΔY)), NAMEX) replaces that point with a permanently invisible line. (See Section E.4 for NEVIS.)

Note: Before any of the character plotting procedures are used, a call should first be made to procedure LAYOUT. Four sizes of characters are available, and the hardware plots a blank line after it plots each character for spacing. The procedure

LAYOUT(ΔH#, ΔV#, SIZE#)

adds nothing to the display file, but merely defines the parameters of the spacing line, and the size of the characters. ΔH, ΔV, and SIZE remain constant for all calls to PACKED, SINGLE, and SPECIAL until LAYOUT is called again. Characters and their following blank lines are not affected by the rotation matrix multiplication, and therefore, may not be rotated. If LAYOUT is not called, the assumption ΔH = 6, ΔV = 0, SIZE = 0 is made for packed and single characters, and ΔH = 1, ΔV = 7, SIZE = 0 for special characters.

The procedure LINEQ is provided for reading in a string of characters from the teletype and putting it into the .BCQ. format so that it can be used with the procedure PACKED. LINEQ prints "TYPE." when it is expecting input.

PLOT(PACKED(LINEQ( ))) will read a line of text from the teletype and plot it at the current beam position.

3. Nonstandard Objects

Any object may be added to the display file with PLOT or RPL as long as the console commands are arranged in the format of Fig. 3.6. Occasionally the user may wish to build up such an element himself in order to create an entity that will have a single "name". The following integer procedures are provided for building console commands to help with this task. (See Section K for the detailed descriptions.)

MASEPOI	Make a set point command
MALIGEC	Make a line generate command
MAZWD	Make a Z word
MAROH	Make a ROH command
MAROV	Make a ROV command
MASGLE	Make a single (unpacked) character command
MASPEC	Make a special character command
MAPACK	Make a packed character command

Suppose we wish to add a square of side 100 to the display file as an object, which will always be treated as a single entity. If we make the Return Code bit (Fig. 3.6) zero, then the register containing the commands will not be returned to free storage by PLOT or RPL, and so we may add the same object to the file several times. E.g.,

```
INTEGER ARRAY A(4) $,(a forward stored array)
SQUARE = LOC A $,        (makes SQUARE a pointer to the 0th
                          register of the array A)
A = 4 $,                 (the system return code bit is set to 0,
                          the object is 4  words long)
A(1) = MALIGEC(100, 0) $,
A(2) = MALIGEC(0, -100) $,
A(3) = MALIGEC(-100, 0) $,
A(4) = MALIGEC(0, 100) $,
PLOT(SQUARE, NAME) $,(causes a square to be plotted at the
                       current beam position)
```

For users who plan to include rotation in their programs, a further procedure ROTMUL is included. This provides a facility for computing any rotation matrix which may then be used as input to the procedures MAROH, MAROV and ROT which build rotation matrix commands. ROTMUL may be used to produce an identity matrix, to multiply two matrices, or to produce a matrix corresponding to a rotation of some angle about one of the three axes.

4. Modifying Objects

Two procedures are provided for making objects invisible before they are added to the display file. When INVIS is used, the object may later be made visible using VIS; when NEVIS is used, the object is permanently invisible. These procedures only affect the line and set-point commands in the object. They are "transparent" to their object argument, so they may be nested in calls.

OB = INVIS(OB)	Allows invisible picture parts to be added to the display file. Used in conjunction with CIRCLE, LIN, SETPT. E.g., PLOT(INVIS(LIN(ΔX, ΔY))). After addition to the file they may be converted back to the visible type with procedure VIS (Section F).
OB = NEVIS(OB)	Same as INVIS except that the picture parts are converted permanently to the invisible type, i.e., VIS has no effect.

The procedure INSEN is provided for making objects insensitive to the light pen, i.e., they cannot be "seen" by the pen. It affects only lines, setpoint, and character commands.

OB = INSEN(OB)	Allows pen insensitive picture parts to be added to the display file. Used in conjunction with CIRCLE, LIN, PNT, SETP, SINGLE, SPECIAL, PACKED, TEXT. E.g., PLOT(INSEN(LIN(ΔX, ΔY))).

F. CONVERTING COMMANDS IN THE DISPLAY FILE FROM VISIBLE TO INVISIBLE AND VICE VERSA

Procedures INV and VIS convert line generate and set-point commands already added to the display file from visible to invisible, and from invisible to visible respectively.

These are both "here to there" type functions and work in the same way.

For example, INV(HERE, THERE*)
causes all line generate and set-point commands in the display file between HERE and THERE inclusive to be converted to the invisible type. If THERE is omitted then only HERE is affected. If HERE is a subpicture name, then all such commands within that subpicture definition are converted and THERE is ignored

VIS(HERE, THERE*) reverses the above call.

Only lines and set-points which are not permanently invisible will be affected.

G. INITIALIZATION

The preceding sections have described procedures for controlling the display file. We have considered how to create and edit the display file, the method for defining and calling subpictures, the naming scheme for items in the display, and the building of standard and non-standard objects that are to be added to the display file.

This chapter concerns initialization functions to be performed at the beginning of a display program. The procedure SGNON(NCON #) must be called at the start of the user's program before any communication can take place with the display console. NCON is the number of consoles on which output should appear (1 or 2).

SGNON sets a condition whereby it is put into "input wait" status if an attention, such as a light pen "see", is asked for and there are none. When an attention arrives, the former status is restored. SGNON also sets the condition whereby the Return Code bit is inserted in standard objects (see Section E).

The program may be switched out of the "input wait" condition (so that it remains in "working" status) with procedure IW(ONOFF≠), and the Return Code condition may be controlled with procedure RTNCOD(ONOFF≠).

If attentions are to be recorded, then procedure SATBUF(N#) must be called to set up an attention buffer of size N. (See Section I. 1). The SATBUF call also causes a <u>Set C</u> word which enables the light pen to be placed at the beginning of the picture.

A call to procedure SGNOFF( ) or a CTSS "logout" signs the user off as a console user. The picture disappears, and the console is then available for another user. A second user already using the other console is unaffected.

H. DISPLAY FILE INFORMATION

1. <u>Dumping the Display File into B-Core</u>

DMP(HERE*, THERE*)

Procedure DMP may be used to dump either the complete buffer or selected sections of the display file into a block of free storage in B-core. All objects and subpicture calls from HERE to THERE inclusive will be dumped. If THERE is omitted, only the single object or call corresponding to HERE is dumped. If HERE is a subpicture name, then the whole subpicture definition will be dumped. If no arguments are given, the whole display file and all subpicture definitions are dumped.

2. Display File Parameters

Procedure ABUF gives the overall size of the display buffer and procedure VAC gives the number of vacant words in it at the time the call is executed. These may be used at any time. These procedures have no arguments, e.g., N=ABUF( ).

I. INPUTS

For any type of interaction with the user, the Display Console must be capable of inputting information to the user's program, as well as acting as an output device displaying a picture. In this section we consider the two distinct types of input available -- real-time inputs and passive inputs. The real-time inputs include "pen sees" and button pushes; the passive inputs include toggle switch settings, and the position of the "globe" or "crystal ball".

1. Real-Time Inputs

ATTN(ATAR##)

Pushing any of the 36 push buttons or the light pen seeing a picture part on the screen are real-time events that are known as attentions. The way these attentions are serviced is a function of operating the display in time-sharing. They cannot have a real-time effect on the user's program, since it may not be running in the time-sharing system at the time of the interrupt. Rather, they must be stored by the Display Controller and made available to the user the next time his program is in core. Of course, the faster the response of the time-sharing system, the nearer they appear to have a real-time effect. Such a system does enable the user to continue operating the display even though his program is not in core.

The attentions are stored in a section of the display buffer maintained by the display controller in a "first-in first-out" list called the attention queue. When an attention is added to the queue, a character is displayed on the screen at the current position of the light pen. The addition of the character to the display reassures the user that his attention has been recorded, even though the associated action must await the next time his program is run by the time-sharing system.

When the user's program cycles into active status, each time a call to procedure ATTN is made, the top item on the attention queue is read and the corresponding character is removed from the screen. ATTN interprets the data from the queue, and stores it into the attention array ATAR supplied by the user. No attention information is available to the user unless he calls ATTN, even though some exists in the queue. A dimension of 10 for the array ATAR will be adequate unless the user has subpicture calls nested more than three deep, in which case an additional 2 registers will be required for each extra level of call.

It is up to the user to set the size of the attention buffer. This should be done immediately after signing on by making a call to procedure SATBUF (Section G). The larger this buffer, the less room there is for the display file, since they are both located in the same display buffer. On the other hand, if the attention buffer is too small it is easily filled while the user's program is out of core, so causing further attentions to be lost. This forces the user to wait until his program comes back into core, to process the waiting attentions before he may continue. A fair balance with present time-sharing characteristics seems to be 100 registers for the attention buffer.

2. Passive Input Registers

In addition to the push buttons and "light pen sees" which cause interrupts, the console hardware provides an assortment of passive inputs. These are highly important to the concept of "man/machine interaction", as they provide natural and convenient means for interacting with, and so controlling, the display. They are described in detail in Chapter I, and include 9 decimal or "digi"-switches, two rows of 36 toggle switches, three 7-bit shaft encoders, and a joystick device known as the "globe" or the "crystal ball", which has

7 discrete positions about each of its three axes. Finally, the light pen may also be a passive input since its position may be read at any time by the computer. They are referred to as "passive inputs", since their values are read only by specific request from the computer, contrasting with the "active inputs", such as button pushes or "light pen sees" which cause interrupts.

It must be emphasized that none of these passive devices affects the display directly. Rather, they are a comprehensive set of different kinds of input, whose functions are assigned entirely by program.

The "crystal ball" is designed principally for use as a rate control. Its main use is for rotation of three-dimensional pictures, although other functions could equally well be assigned to it by program. Similarly, the three shaft encoders are convenient for controlling horizontal and vertical translation, and magnification of pictures, but may be used for other purposes as well.

Several procedures are provided to read the values of these inputs. ANALOG and DIGI both provide the user with a "raw" word, read straight from the hardware which requires decoding to extract the derived information.

The remaining procedures provide the value of the particular input in a convenient form.

Inputs are read only at the time when the procedure is actually being run. There is no relationship whatsoever between these inputs and the attention array (Section I.1). Care should be taken to ensure that an input has been read before it is changed.

As an example, suppose we want to control whether lines plotted are invisible or visible by setting digi-switch number 9. We will assume that the line is to have parameters, DELX, DELY and DELZ, and that it is to be invisible whenever the switch is set to 1. REDIGI(LEFT, RIGHT*) decodes any consecutive set of digi-switches into a right-justified integer value. The AED-0 statement:

```
    IF REDIGI(9) EQL 1
    THEN PLOT(INVIS(LIN(DELX, DELY, DELZ)))
    ELSE PLOT(LIN(DELX, DELY, DELZ)) $,
```

performs the required function.

J. REAL-TIME PROGRAMS

1. General Description

Since the display console is operating in time-sharing, any real-time programs must be included in the Display Controller. A limited number of real-time programs have been provided in the present Controller. These perform rotation, scaling, translation, and dragging of pictures under the control of the light pen, and may be initiated by calls to GRAPHSYS procedures. They are initiated or terminated by a call to procedure RLT. Several pictures may be simultaneously rotated, scaled, or translated at varying rates under control of an input device (axis of the crystal ball or a shaft encoder) of the user's choosing. The only limit to the number of pictures is the size of the display buffer.

As we have already discussed, the display buffer is used to store the display file, and the attention buffer. We come now to its final use, namely storage of the Real-Time Buffer and the Rotation Matrix Buffers.

When a real-time program is initiated, GRAPHSYS enters a single word instruction into the real-time buffer. Each of these real-time instructions tells the display controller which program is required, which input device controls the program (shaft encoder, crystal ball), the speed and direction of change required, and finally which word(s) in the display file (Rotation, Set Point, or Set C commands) are to be adjusted. Before outputting each frame of the display to the console, the display controller inspects all the real-time instructions in the buffer and makes an incremental change to the appropriate commands.

These incremental changes are sufficiently small for the rotation, translation, etc., to appear quite smooth and continuous on the screen.

The size of the real-time buffer may be set by the user after SGNON with procedure SRLBUF( ). If SRLBUF ( ) is not called, its length is automatically set to 25. The length of the buffer may not be changed dynamically.

2. The Real-Time Functions

Let us now consider the real-time functions available:

RLT(FCN#, NAME, SPEED#, DIRN#, CONTROL# , CNAME*)
Suffice it to say here that FCN indicates which real-time function is required, and NAME is the name of the object in the display file whose display commands are to be affected. Each of the available functions is discussed below.

Rotation. Rotation may be requested about the x, y and z axes. The user must specify the name of a pair of rotation matrix commands which are to be changed in real-time. For each new pair of commands affected by rotation (or scaling, see below), the system sets up real-time rotation in the display buffer a 10 word Rotation Matrix Buffer required by the program. It contains at any time the 9 current values of the rotation matrix plus a scale factor (see Fig. 3.7) which

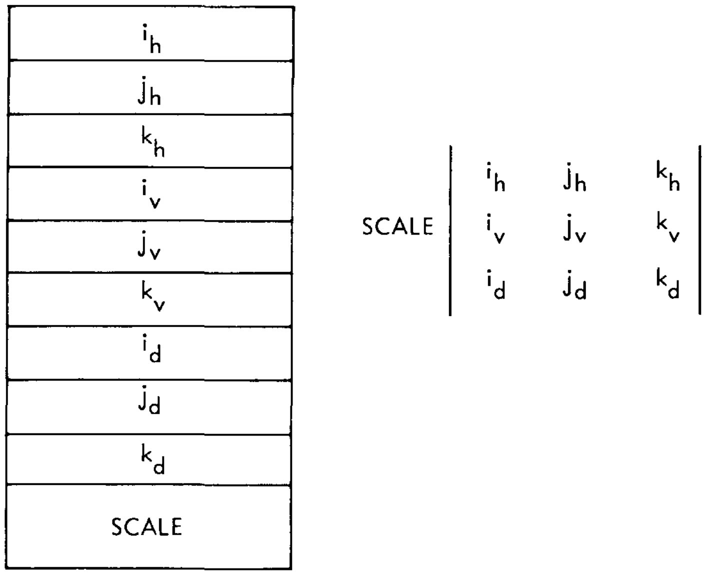


Fig. 3.7 Rotation Matrix Buffer format

are used to modify the line generator output to create the illusion of rotation. They are stored as 35 bit sign magnitude fractions, i.e., the largest number, and nearest to 1 is 377777777777(8).

This buffer is automatically built when required, and deleted when no longer needed. There is a separate buffer for each pair of rotation commands upon which a real-time program is operating. The user may inspect or change the contents of this buffer by using RWROT(NAME, MATRIX, RW). NAME refers to the rotation commands in question, MATRIX is a 10-word forward-stored array in B-core (format of Fig. 3.7), and RW tells whether the buffer is to be read into MATRIX, or whether MATRIX is to be written into the buffer.

Scaling. Reduction in size is performed by scaling a rotation matrix whose terms have a maximum value of 1. Hence when scaling is initiated, the name of a pair of rotation matrix commands must be given. Magnification is effected by altering the "M bits" (they affect the "bit spacing" of the display) in a Set C word which immediately precedes the rotation commands in display sequence. (N.B. It is up to the user to ensure the Set C word is in the right place.) If scaling greater than the original size is required the optional argument CNAME must be used. The real-time program automatically adjusts the "M bits" when the input device is changed to give a scaling greater than original.

Translation. Translation may be requested up and down or right and left. The name of a set point command must be given.

Dragging with the Light Pen. Pictures may be made to follow the light pen. The name of a set point command must be given, and the effect of this function is for that set point to remain coincident with the center of the tracking cross.

3. Disconnecting Real-Time Functions

The procedure

RLTRMV(NAME*, FCN #*)

removes from real-time everything if there are no arguments, all functions operating on NAME for one argument, or the specified FCN operating on NAME if both arguments are given.

4. Synchronized Movie Camera

The display console includes hardware to trigger a movie camera to take one frame of film for each frame of picture displayed. The camera may be controlled totally from the user's program which would specify the number of frames to be taken, or it may be attached to push button 4 and controlled by the operator. The push button may be used either as a signal to take a specified number of frames, or, by setting the number of frames very large, as an on/off switch for the camera. The call

CAMERA (N#, NOTNOW #*)

attaches the camera to button 4 and takes N frames unless NOTNOW is present and nonzero.

K. PROCEDURE DESCRIPTIONS

This section details the various GRAPHSYS procedures as a ready reference.

1. Manipulating the Display File

The value of several of the procedures is a display pointer name (DPN). Its value is identical to the optional argument NAME. When NAME is omitted, its value is the display pointer name assigned by the system. RPL is an exception; when NAME is omitted, its value is the display pointer name corresponding to INAME.

DPN = PLOT(OBJECT, NAME*, PNAME*)

Use:	To add console commands to the display file.
OBJECT	Any of the integer procedures described in the following section, CIRCLE, INVIS, LIN, NEVIS, PACKED, PNT, ROT, SETC, SETPT, SINGLE, SETF, SPECIAL, or a pointer to an array of the form shown in Fig. 3.6. If the return code bit (bit 20) is a 1, PLOT returns the N+1 words to free storage. (See Fig. 3.6)
NAME*	The name assigned by the user to this item. It is a pointer to a single register which the user makes available to GRAPHSYS for storage of display information concerning OBJECT.
PNAME*	The name of the object or subpicture call after which OBJECT is to be inserted. If PNAME is omitted, then OBJECT will be added to the end of the display file.
Note:	Insertions cannot be made at the beginning of the display file, or at the beginning of a subpicture definition. The same effect can be achieved, however, by starting the display file or definition with a blank command.

DPN = DEFSUB(NAME*)

Use:	To start defining a subpicture.
NAME*	As for PLOT except that it must point to the first of two consecutive registers which the user makes available to GRAPHSYS.

ENDSUB( )

Use:	To end the definition of a subpicture.

DPN = CALL(SNAME, NAME*, PNAME*)

Use:	To call a previously defined subpicture.
SNAME	The name of a previously defined subpicture.
NAME*	As for PLOT except that it must point to the first of two consecutive registers which the user makes available to GRAPHSYS.
PNAME*	As for PLOT.

DPN = CPY(THIS, NAME*, PNAME*)

Use:	To make a duplicate copy in the display file of objects which are already in the file. Subpictures or subpicture calls may not be copied.
THIS	The name of some object previously added to the display file with PLOT, RPL, or CPY itself.
NAME* PNAME*	As for PLOT.

RMV(HERE*, THERE*)

Use:	To remove console commands permanently from the display file.
HERE*	(a) The name of any object of subpicture call added to the display file with CALL, CPY, PLOT, or RPL. (b) A subpicture name. In this case the function is applied to the complete subpicture definition and THERE is ignored.
THERE*	The name of any item added to the display file with CALL, CPY, PLOT, or RPL. When this optional argument is given the function is applied to all commands in the file, in display sequence from HERE to THERE inclusive. HERE and THERE must both be either within the main display or both within a single subpicture definition
Note:	When both arguments are omitted the whole of the display file and all subpicture definitions are removed permanently.

DPN = RPL(NEW, INAME, NAME*)

Use:	To replace an item in the display file with another. This is not an overlay function, so the old and new items need not be represented by the same number of console commands.
NEW	Either (a) as OBJECT in PLOT or (b) as SNAME in CALL. (In case (b) a subpicture call will be generated.)
INAME	The name of the item which is to be replaced. It may be either an object previously added to the display file with PLOT, CPY, or RPL, or the name of a subpicture call generated with CALL or RPL.
NAME*	The name of the new object, or call. If omitted the new itemwill automatically be named INAME.
Note:	If NEW is an SNAME and NAME is not given, INAME must point to the first of two consecutive display registers, even if the INAME being replaced is an object rather than a call.

2. Creating Objects

a. Standard Objects

OB = SETPT(H#, V#)

Use:	Used in conjunction with PLOT or RPL to add a set point to the display file.
H#	Horizontal coordinate of set point.
V#	Vertical coordinate of set point.
Note:	Arguments are taken modulo 2^{12}.

OB = LIN(DELX#, DELY#, DELZ#*)

Use:	Used in conjunction with PLOT or RPL to add a line to the display file, e.g., PLOT(LIN(100, 200, 300)).
DELX#	ΔX component of line.
DELY#	ΔY component of line.
DELZ#*	ΔZ component of line. This component allows three-dimensional objects to be easily specified in a form that can be rotated.
Note:	Arguments are taken modulo 2^{10}.

OB = LLIN(DELX#, DELY #, DELZ#*)

Use:	Used in conjunction with PLOT or RPL to add a long line (longer than can be drawn by the hardware with a single line command) to the display file. Arguments are the same as for LIN except that they are taken modulo 2^{13}.

OB = PNT( )

Use:	Used in conjunction with PLOT or RPL to add a point to the display file.
Note:	This is not a set point. It consists of two short vectors of unit length, one positive and the other negative such that the beam position is unchanged.

OB = CIRCLE(XC #, YC#, X1#, Y1#, X2#, Y2#, CLOCK≠, XE#**, YE#**)

Use:	Used in conjunction with PLOT or RPL to add an arc in the XY plane to the display file.
XC#,YC#	Coordinates of circle center in scope units.
X1#,Y1#	Coordinates of start point of arc in scope units.
X2#,Y2#	Coordinates of any point on the radius vector which intersects the end of the arc.
CLOCK≠	If TRUE, a clockwise arc is produced from (X1, Y1), to the radius vector. If FALSE an anticlockwise arc is produced from (X1, Y1) to the radius vector.
XE#**, YE#**	Optional output arguments. The calculated end points of the arc. (See Fig. 3.8)

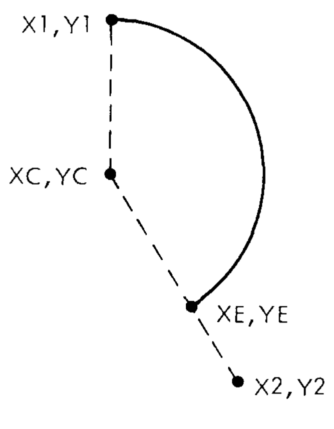


Fig. 3.8 Arc computed by procedure CIRCLE

Note:	The arc is made up of short straight-line segments. The number of segments is proportional to the radius, which has no limits on size. The distance of (X2, Y2) from the center may be less than, equal to, or greater than the radius. If (X1, Y1) coincides with (X2, Y2) a complete circle will be displayed, and (XE, YE) is the same point. Note also that the parameters define only SIZE, orientation, and extent of arc. Its absolute location when plotted will depend on "current beam position".

OB=CIRC3D(XC#, YC#, ZC#, X1#, Y1#, Z1#, X2#, Y2#, Z2#, CLOCK≠, <u>XE#**</u>, <u>YE#**</u>, <u>ZE#**</u>)

Use:	Used in conjunction with PLOT or RPL to add an arc in any arbitrary plane to the display file. Arguments are the same as for CIRCLE, except that they are three-dimensional.
Note:	If the three points given are colinear, CIRC3D will not be able to determine a plane in which to draw the circle. It will print a comment and allow the user to type in another point which determines the plane.

OB = SINGLE(CHARACTER#)

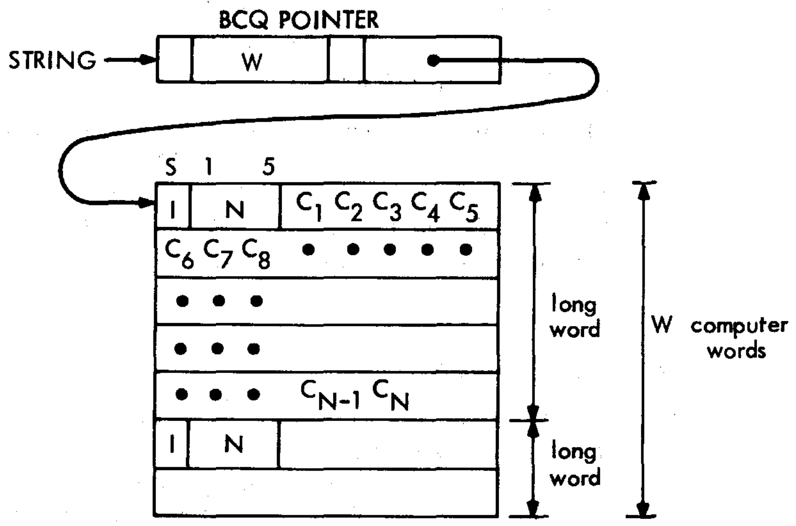
Use:	Used in conjunction with PLOT or RPL to add an "unpacked" character to the display file. E.g., to display a "D" write PLOT(SINGLE(24C))$.
SINGLE	Code number of desired character (See Table 1.6).
Note:	LAYOUT should be called before using SINGLE.

OB = SPECIAL(POINTER)

Use:	Used in conjunction with PLOT or RPL to add special characters to the display file.
POINTER	A pointer to an array containing N in the 0th position, and N special character words in positions 1 through N. This permits the user to design any special characters (such as Greek letters) to be displayed by the character generator.
Note:	LAYOUT should be called before using SPECIAL.

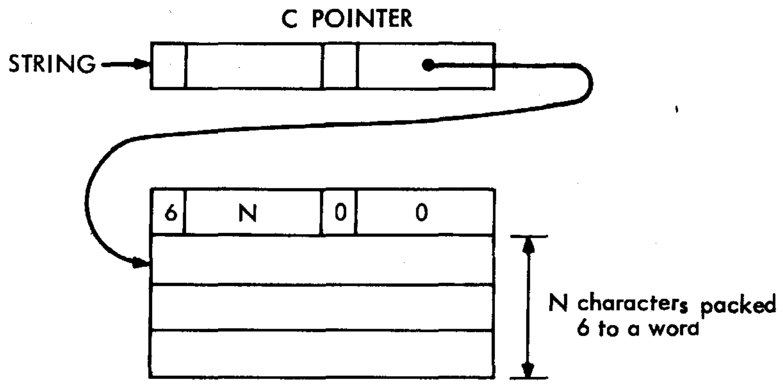
OB = PACKED(STRING)

Use:	Used in conjunction with PLOT or RPL to add packed characters to the display file.
STRING	A pointer to an array containing the characters in "long word" form as shown in Fig. 3.9a. The decrement of STRING <u>must</u> contain the number W of words in the array. The format required by STRING is automatically built by the procedure LINEQ and by the AED-0 operator .BCQ. for the



(a) .BCQ. String

Format of element pointed to by 'STRING' in procedure PACKED. W is the total number of computer words. In each "long word", N is the total number of characters and I is 0 if this is last "long word" or 1 if more "long words" follow.



(b) .C. String

Format of element pointed to by 'STRING' in procedure TEXT. N is the total number of characters. Each word contains six characters.

Fig. 3.9 Formats of Character Pointers and Strings

characters in the standard BCD set. E.g.,

PLOT(PACKED(LINEQ( ))) $, ...A line from the teletype will be plotted on the screen //

PLOT(PACKED(.BCQ. /THESE WORDS WILL APPEAR ON THE SCREEN/))$,

Notes:

1. LAYOUT should be called before using PACKED.
2. The code expected in the .BCQ. string is not BCD, but the console symbol generator code in Table 1.6 (different for special characters). The AED operator .BCQN. may be used to produce these special characters.

OB = TEXT(STRING)

Use:	Same as packed.
STRING	A pointer to an array containing characters in the .C. format as shown in Fig. 3.9b. The AED-0 operator .C. provides the proper format for characters in the standard BCD set. E.g.,

PLOT(TEXT(.C. /THESE WILL TOO/))

Notes:

1. LAYOUT should be called before using TEXT.
2. The console hardware uses the .BCQ. format required by PACKED, so when using AED-0, it is the more efficient form. TEXT is provided for compatibility and ease of use from other languages.
3. See Note 2 for PACKED. Use .CN. for special characters.

LAYOUT(DELH#, DELV#, SIZE#)

Use:	To specify the spacing between characters and their size.
DELH#	ΔH spacing after characters. DELH is taken modulo 2^6 for SINGLE characters and modulo 2^{10} for PACKED AND SPECIAL characters.
DELV#	ΔV spacing after character. DELV is taken modulo 2^{10}.
SIZE#	Size of character (0 through 3). Only sizes 1 and 3 are available for special characters.

Notes:

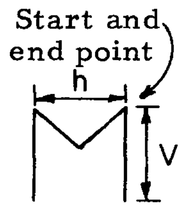
1. LAYOUT should be called <u>before</u> PACKED, SINGLE or SPECIAL are used. If it is not called, the assumption will be $\Delta H = 6$, $\Delta V = 0$, SIZE = 0 for packed and single, $\Delta H = 1$, $\Delta V = 7$, SIZE = 1 for special characters.

2. The parameters specified with LAYOUT remain constant for all subsequent characters that are plotted until LAYOUT is called again.

3. The start and end point of a character plotted with integer procedures SINGLE or PACKED is at its top right corner.

4. The sizes of characters displayed on the scope, are approximately as given below. These measurements were based on the letter M.

Size	H inches	V inches
0	.07	.10
1	.10	.14
2	.14	.20
3	.20	.28



5. Character commands built with PACKED, SINGLE or SPECIAL and not yet added to the display file are <u>not</u> affected by a new call to LAYOUT with new values of DELH, DELV, and SIZE.

STRING = LINEQ( )

Use: To take a line of text typed onto the teletype and pack it into the correct format (.BCQ.) for addition to the display file as "packed" characters.

Notes:

1. The procedure prints "TYPE." when it is expecting input.

2. A line up to 84 characters long will be accepted.

3. LINEQ should be used in conjunction with PACKED and PLOT. The call

   PLOT(PACKED(LINEQ( ))) $,

   gives the facility to display the typed text. Its position on the screen will start at the current beam location.

OB = ROT(MATRIX##*)

Use: Used in conjunction with PLOT or RPL to add a pair of rotation matrix commands to the display file.

MATRIX##* The name of a ten-word integer array containing members of the rotation matrix, plus a scale factor. The format of the words is the same as for the Rotation Matrix Buffers. (See Section J.2, and Fig. 3.7)

Notes:

1. If MATRIX is omitted the rotation matrix commands correspond to the identity matrix.

2. See also Procedure ROTMUL, Section K.2.b.

OB=SETC(MASTER#**, SLAVE#**, MAG#**, INTEN#**, LP1#**, LP2#**)

Use:	Used in conjunction with PLOT or RPL to add a set C command to the display file. (See Chapter I.D.7)
MASTER#**	1 for Master scope, otherwise 0.
SLAVE#**	1 for Slave scope, otherwise 0.
MAG#**	$0 \leq MAG \leq 3$ Magnifies bit spacing by a factor of 2^{MAG}.
INTEN#**	Intensity level $0 \leq INTEN \leq 15$.
LP1#**	1 for light pen 1 sensitive, otherwise 0.
LP2#**	1 for light pen 2 sensitive, otherwise 0.
Note:	If SETC is called with no arguments then a standard Set C word is built, equivalent to SETC (1,1,0,15,1,1) If the user is only logged into one console, then the MASTER/SLAVE and LP1/LP2 bits will be automatically adjusted by the Display Controller to refer only to his console.

OB = SETF(MOVH#, SCALH#, MOVV#, SCALV#, MOVI#, NOPLOT#)

Use:	Used in conjunction with PLOT or RPL to add a Set F command to the display file. (See Chapter I.D.8)
MOVH#	0 = No movement.
	1 = Move left.
	2 = Move right.
SCALH#	$0 \leq SCALH \leq 3$
MOVV#	As for MOVH but in vertical direction (1=down, 2=up)
SCALV#	$0 \leq SCALV \leq 3$.
	0 = No movement.
	1 = Decrease intensity 1 unit.
	2 = Increase intensity 1 unit.
NOPLOT#	0 = Plot.
	1 = Blank.

b. Nonstandard Objects

All the procedures below except ROTMUL build a single console command CC.

CC = MASEPOI(H#, V#, TYPE#*)

Use:	To make a set point command. (See Chapter I.D.1)

H#	Horizontal coordinate of set point (taken modulo 2^{12})
V#	Vertical coordinate of set point (taken modulo 2^{12})
TYPE#*	The type of point. The following types are available: 0 Visible, Pen Sensitive (may be seen by light pen). 1 Visible, Pen Insensitive (cannot be seen by light pen). 2 Invisible, Pen Sensitive (See Procedures VIS, INV, Section K.3). 3 Invisible, Pen Insensitive.
Note:	If TYPE is not given, type 0 is assumed.

CC = MALIGEC(DELX#, DELY#, TYPE#*)

Use:	To make a line generate command. (See Chapter I.D.2)
DELX#	ΔX component of line (taken modulo 2^{10}).
DELY#	ΔY component of line (taken modulo 2^{10}).
TYPE#*	As for MASEPOI.

CC = MAZWD(DELZ#)

Use:	To make a Z-word for plotting lines in three-dimensional space. In making an object, the console command produced by MAZWD must immediately precede the command produced by MALIGEC, which specifies the X and Y components. (See Chapter I.D.2)
DELZ#	ΔZ component of line (taken modulo 2^{10}).

CC = MAPACK(TYPE#*)

Use:	To make a packed character command. (See Chapter I.D.9.b)
TYPE#*	As for MALIGEC, except that only types 0 and 2 are available.
Note:	The value of ΔH, ΔV and SIZE in this command are equal to those specified by the last call to LAYOUT.

CC = MASGLE(CHARACTER#, TYPE#*)

Use:	To make a single (unpacked) character command. (See Chapter I.D.9.a)
CHARACTER#	The code of the desired character (see Table 1.6)
TYPE#*	As for MAPACK.
Note:	As for MAPACK.

CC = MASPEC (TYPE#*)

Use:	To make a special character command. (See Chapter I. D. 9. c)
TYPE#*	As for MAPACK.
Note:	As for MAPACK.

CC = MAROH(MATRIX##*)

Use:	To build an "ROH" rotation matrix command (first line of matrix). (See Chapter I. D.3)
MATRIX##*	The name of 10 word integer array containing a rotation matrix and scale factor in standard format. (See Fig.3.7) MAROH uses only the first three words and scale factor. When MATRIX is omitted an identity matrix is assumed.

CC = MAROV(MATRIX##*)

Use:	To build an "ROV" rotation matrix command (second line of matrix) (See Chapter I. D. 3)
MATRIX ##*	As for 'MAROH' except that MAROV uses the second three words plus the scale factor.

CC=MASETC(MASTER#**, SLAVE#**, MAG#**, INTEN#**,
LP1#**, LP2 #**)

Use:	To make a Set C Control command. (See Chapter I. D. 7)
Arguments:	As for Set C, Section K.2.a.

CC = MASETF(MOVH#, SCALH#, MOVV#, SCALV#, MOV1#, NOPLOT#)

Use:	To make a set F command (See Chapter I. D. 8)
Arguments:	As for SETF, Section K.2.a.

ROTMUL(OUTPUT##, INPUT##*, AXIS#*, ANGLE+*)

Use:	To compute rotation matrices in standard format (see Fig.3.7) for use as input to procedures MAROH, MAROV, and ROT.
OUTPUT ##	Name of 10-word integer array where output is to be stored.
INPUT##*	Name of 10-word integer array containing an input rotation matrix. When INPUT is omitted, OUTPUT will be an identity matrix with unit scale factor.
AXIS#*	(a) When ANGLE is omitted AXIS is the name of a 10-word integer array containing a rotation matrix. It is multiplied by that in INPUT, and the result stored in OUTPUT.

	(b) When ANGLE is included, AXIS is a single integer indicating an axis.
	1 = Horizontal.
	2 = Vertical.
	3 = Depth.
	ANGLE and AXIS together specify the matrix to be multiplied by INPUT to form the OUTPUT matrix.
ANGLE+*	Real value in radians of angle by which rotation is desired about the specified axis. Positive values give anticlockwise rotation.

c. <u>Modifying Objects</u>

OB = INVIS(OB)

Use:	To convert all lines and set points of OB to be of the invisible type.
OB	Any of the integer procedures, CIRCLE, LIN, PNT, SETPT or a pointer to an array of the form specified by OB (See Fig. 3.6). Console commands in such an array which are not line or set point commands are affected.
Notes:	1. INVIS may be used as the first argument of PLOT or RPL. E.g., PLOT(INVIS(LIN(200,200))).
	2. Lines and set points made invisible with INVIS and added to the display file, may be converted back to visible type with procedure VIS.
	3. The value of INVIS is identical to its argument.

OB = NEVIS(OB)

Use:	To convert all lines and set points of OBJECT to be <u>permanently</u> of the invisible type.
OB	As for INVIS.
Notes:	1. As for INVIS.
	2. Lines and set points made invisible with NEVIS and added to the display file will be unaffected by procedure VIS.
	3. The value of NEVIS is identical to its argument.

OB = INSEN(OB)

Use:	To convert all lines, setpoints, and characters of OB to be of the pen insensitive type.
OB	Similar to INVIS. Include also procedures SINGLE, SPECIAL, PACKED, and TEXT. Only setpoint, line, and character commands are affected.

Notes: 1. As for INVIS.

2. The value of INSEN is identical to its argument.

3. Converting Display File Commands from Visible to Invisible

INV(HERE*, THERE*)

Use:	To convert visible type lines and setpoints within the display file to invisible type.
HERE* THERE* }	As for RMV, Section K.1.

VIS(HERE*, THERE*)

Use:	To convert invisible type lines and setpoints within the display file to visible type.
HERE* THERE* }	As for RMV, Section K.1.
Note:	Lines and setpoints made permanently invisible with NEVIS (Section K.2.c) are not affected.

4. Initialization

SGNON(NCON#)

Use: To sign on as a console user.

NCON# Number of consoles required, 1 or 2.

Notes:

1. SGNON must be called before any communication can take place with the console.
2. It sets a condition whereby the user is put into "input wait" if he asks for an attention and there are none.
3. If the user is already signed on, the display file is cleared, the attention buffer set to zero, the attention display (rubber band lines, etc.) is cleared and tracking is stopped. A second call to SGNON does not, however, accomplish a RMV( ); i.e., GRAPHSYS data structure beads are not returned to free storage.
4. The condition is set whereby the Return Code bit is inserted when standard objects are built (see Section G).
5. If two consoles are requested and only one is available, the user is signed on with that one, and is informed via the state word and the teletype.

SGNOFF( )

Use: To sign off.

IW(ONOFF≠)

Use:	To control whether the user's program is put into "input wait" or remains in "working" status when an attention is asked for and there are none.
ONOFF≠	If TRUE "input wait", while if FALSE "working" condition is set up.

RTNCOD(ONOFF≠)

Use:	To control whether blocks containing display file commands are returned to free storage by PLOT or RPL.
ONOFF≠	If FALSE system procedures building standard objects (Section E.2) will <u>not</u> insert the Return Code bit and the element containing the console commands will not be returned to free storage. If TRUE the Return Code bit is inserted.

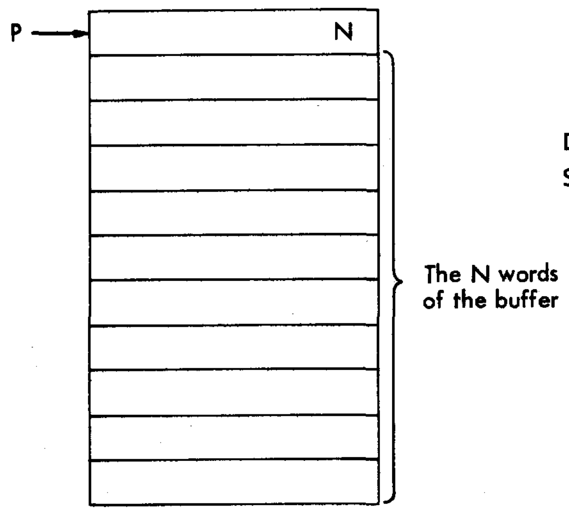
N = SATBUF(N#*)

Use:	To set up the size of the attention buffer in the display buffer and to enable the light pen (via. a <u>Set C</u> word).
N#*	Number of registers to be used for the buffer.
Notes:	1. This procedure may only be called once with an argument, and that must be before any attentions are generated. The recommended time is immediately after SGNON. All attentions that occur before it is called will be lost. A reasonable value for N is 100. (See also procedure ATTN, Section K.6.a)
	2. N is identical to N#*. If called without the argument, N equals the size to which the attention buffer has been previously set.

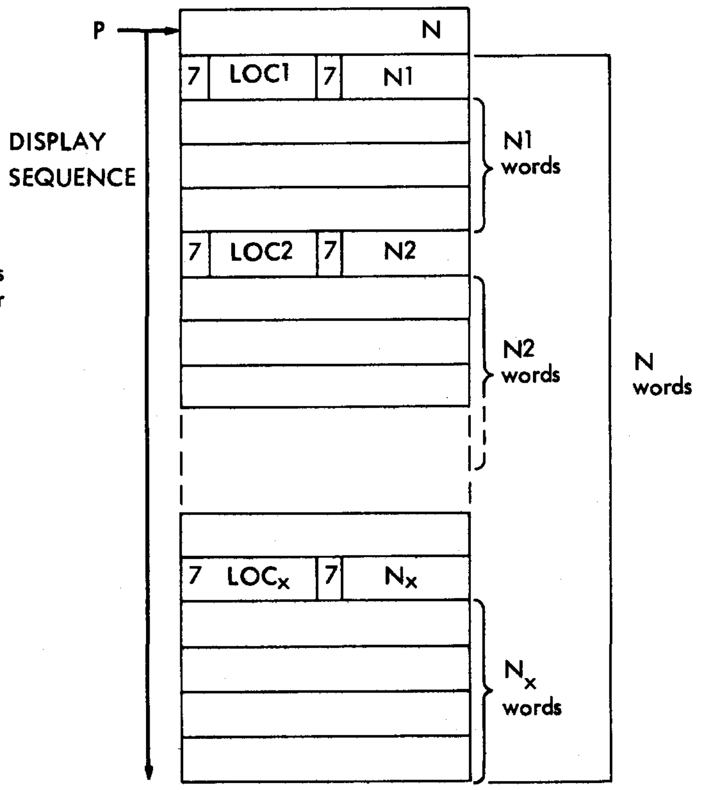
5. <u>Reading Display File Commands and Parameters</u>

P = DMP(HERE*, THERE*)

Use:	To dump the whole display buffer, or selected parts of the display file into B-core.
HERE* } THERE* }	As for RMV, Section K.1.
Notes:	1. When neither argument is given the whole display buffer is dumped into contiguous registers as shown in Fig. 3.10a.
	2. When argument(s) are given the required display file commands are dumped into contiguous registers as shown in Fig. 3.10b. LOCi is a pointer to the position which the following commands occupied in the display buffer. The first position in the display buffer has location 0.



(a) Dump of whole display buffer in B-core



(b) Dump of selected portion of Display file in B-core

Fig. 3.10 Formats Used for Dumping the Display Buffer

3. The buffer pointed to by P is obtained from free storage[2] and it is the user's responsibility to return it when it is no longer needed. Its size may be determined by adding 1 to the contents of the first word of the buffer.

N = ABUF( )

Use:	To obtain the size of the display buffer.

N = VAC( )

Use:	To obtain the number of vacant words remaining in the display buffer.

6. Reading Inputs

a. Real-Time Inputs

ATTN(ATAR##)

Use:	Provides details of real-time events that have occurred.
ATAR##	The name of the array (not a pointer) to be used as the attention array. The number of words in the array which will be filled depends upon the type of attention, and is given in the zeroth word of the array. The first word contains a code number specifying the type of attention. The three types of attention, (plus "no attentions") and their codes are listed below. The contents of the attention array are summarized in Table 3.1. This array is stored in the form appropriate to AED-0 and FAP programs, i.e., forward in core. (Programs written in MAD must recognize this fact.) A dimension of 10 for ATAR will be adequate unless the user expects pen sees on items nested more than three levels deep in subpicture calls.

1. Button Pushed. Codes 0 through 35 - (Character B on screen.) Any of the buttons numbered 4 through 35 give an attention, hence their function may be assigned by the user. Buttons S and 1 through 3 have preassigned functions and do not give a "button pushed" attention. Button S turns the tracking cross on and off. Button 3 is used to turn the picture on and off. See 4. for buttons 1 and 2. Button 4 does not give an attention if the camera is attached to it.

2. No Attention. Code 36 - If a request is made for an attention and there are none, either (a) the user is informed of the fact in the attention array, or (b) his program is put into "input wait" status, where it remains until another

Table 3.1

ATAR - Attention Array

FUNCTION	ATAR(0)	ATAR(1)	ATAR(2)	ATAR(3)	ATAR(4)	ATAR(5)	ATAR(6)	....
Button Pushed	N	0 - 35	H (present if tracking)	V (present if tracking)				
No Attention	1	36						
Picture Part Seen by Pen	N	38	HB	VB	<u>Part Seen</u> Display Pointer	<u>Outermost Subpicture</u> Display Pointer of Subpicture Called.	<u>Outermost Subpicture</u> Display Pointer of Subpicture Call.	etc. Subpictures and Calls in pairs to innermost.
Line Completed	5	41	H of start of line.	V of start of line.	ΔX of line.	ΔY of line.		

H	- Horizontal coordinate of pen position.
HB	- Horizontal coordinate of beam position at time of interrupt.
N	- Number of words required to specify this attention.
V	- Vertical coordinate of pen position.
VB	- Vertical coordinate of beam position at time of interrupt.

attention occurs. When signing on, (b) is automatically set. However, the user may change to (a), or swap between them both by calling procedure IW(ONOFF≠). (TRUE for ON.)

3. Object Seen by Light Pen. Code 38- (Character P on screen). When the light pen has seen an object, the display pointer name of that object is put into the attention array. If the object seen is a part of a subpicture, then the display pointer name of that subpicture and of that particular call of the subpicture are made available, see Table 3.1. The subpicture names and call names continue for as many levels as the particular call of the object seen was nested. They are ordered outermost (first called) to innermost (last called). The array ATAR must be dimensioned at least 4+2*n registers long, where n is the maximum subpicture depth.

4. Rubber Band Line Completed. Code 41 - (Line and character L on screen.) The first push of button 1 causes a rubber band line to start at the current pen position. Subsequent pushes fix the rubber band line, and start a new one. Pushing button 2 also fixes the line, but releases the cross without starting a new line. For each line completed its ΔX and ΔY are available in the attention array. This allows figures consisting of straight-line segments to be drawn easily.

Notes:

1. Each time ATTN is called, the topmost attention on the queue is read, interpreted, and the information is stored in the attention array.

2. The size of the attention buffer in the display buffer must be specified by the user after SGNON with procedure SATBUF. Each call to ATTN specifies ATAR which may be the same or different for each call.

b. Passive Inputs

Notes: Every procedure in this section is an integer procedure.

All values are right justified.

PI = ANALOG( ) Value is the "raw" analog word:

Bits S - 2 No meaning
Bits 3 - 6 Crystal Ball Horizontal Axis
Bits 7 - 10 Crystal Ball Vertical Axis
Bits 11 - 14 Crystal Ball Depth Axis

Bits 15 - 21 Left Shaft Encoder
Bits 22 - 28 Center Shaft Encoder
Bits 29 - 35 Right Shaft Encoder

PI = CRYSTD( ) Value is the crystal ball depth axis (4 bits)

PI = CRYSTH( ) Value is the crystal ball horizontal axis (4 bits)

PI = CRYSTV( ) Value is the crystal ball vertical axis (4 bits)

PI = DIGI( ) Value is the raw "digi-switch" word

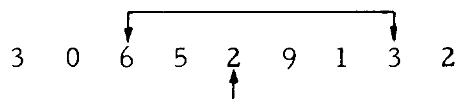
Bits S - 3 1st (leftmost) digi-switch
Bits 4 - 7 2nd digi-switch
Bits 8 - 11 3rd digi-switch
Bits 12 - 15 4th digi-switch
Bits 16 - 19 5th digi-switch
Bits 20 - 23 6th digi-switch
Bits 24 - 27 7th digi-switch
Bits 28 - 31 8th digi-switch
Bits 32 - 35 9th (rightmost) digi-switch

See also REDIGI below.

PI = PENH( ) Value is the horizontal coordinate of the pen position, in sign/magnitude form. If the pen is not tracking, the value is meaningless.

PI = PENV( ) As for PENH, but the vertical coordinate.

PI = REDIGI (LEFT#, RIGHT#*) Value is a right justified integer equal to the setting of any consecutive set of digi-switches. Its use may be more easily understood by an example: Suppose the digi-switches had the setting



Then REDIGI (3, 8) = 652913

REDIGI (5) = 2

PI = SHENC( ) Value is the center shaft encoder (7 bits)

PI = SHENL( ) Value is the left shaft encoder (7 bits)

PI = SHENR( ) Value is the right shaft encoder (7 bits)

PI = TOGA( ) Value is the upper row of toggle switches (36 bits)

PI = TOGB( ) Value is the lower row of toggle switches (36 bits)

7. Real Time Functions

RLT(FCN#, NAME, SPEED#, DIRN#, CONTROL#, CNAME*)

Use: To initiate or terminate operation of a real-time program.

FCN# The function to be performed. Values of FCN have the following meanings:

1. Rotation about z-axis
2. Rotation about x-axis
3. Rotation about y-axis
4. Translate up and down
5. Translate right and left
6. Scale--see description of CNAME below
9. Drag with the light pen
16. Standardized rotation about x, y and z-axes under control of the crystal ball.

When FCN is 16, SPEED, DIRN and CONTROL are ignored, and may be omitted. In this case, SPEED is set to 10, DIRN to 0.

NAME	The name of a <u>pair</u> of rotation matrix commands if FCN refers to rotation or scaling (FCN=1, 2, 3, 6, 16) or the name of a set point command if FCN refers to translation or dragging with the light pen. (FCN=4, 5, 9). RLT checks to ensure that NAME refers to the correct type of command for FCN. If not, the user is informed via the teletype, and the real-time program is not initiated.
SPEED#	A speed multiplier where $0 \leq \text{SPEED} \leq 15$. This determines the rate at which rotation, translation, or scaling occurs. (Ignored for function 9.)
DIRN#	Determines the direction in which the function changes when CONTROL is changed -- 0 or 1. Use 0 for clockwise rotation, positive x, y translation, or increasing scaling (magnification) when the control is turned clockwise. (Ignored for function 9.)
CONTROL#	An integer indicating the input control for the function

1. Left shaft encoder
2. Center shaft encoder
3. Right shaft encoder
4. x-axis of crystal ball
5. y-axis of crystal ball
6. z-axis of crystal ball
7. light pen

CNAME* An optional argument that may be used when scaling is requested (FCN = 6). Normally scaling produces only a <u>reduction</u> in size. If magnification is required, CNAME should be the name of a <u>Set C Command</u> which immediately precedes, in display sequence, the rotation matrix commands to which the scaling is being requested. Magnification is effected by altering the "M bits" in this Set C word. If it does not immediately precede the rotation commands, then that part of the display represented by commands in the display file between them will be affected also.

RLTRMV (NAME*, FCN#*)

Use:	To terminate operation of real-time programs.
NAME	The name of the object to be removed from real-time. If NAME is not given, all real-time operation is terminated.
FCN#*	The particular function associated with NAME which is to be terminated. If FCN is not given, all real-time operation associated with NAME is terminated.

SRLBUF(N#*)

Use:	To set the size of the real-time buffer.
N#*	The size of buffer required.
Note:	This procedure may be called only <u>once</u>, and must be called <u>before</u> any calls to RLT. If it is not called at all, and RLT is called, N is set automatically to 25.

RWROT(NAME, <u>MATRIX</u>##, RW#)

Use:	To read and write a rotation matrix buffer.
NAME	The name of a pair of rotation commands on which rotation or magnification is currently being applied.
<u>MATRIX</u>##	The name of a 10-word array in B-core (see Fig. 3.7).
RW#	If RW = 1 the 10-word rotation matrix buffer in the display buffer is read into MATRIX. If RW = 0 the contents of MATRIX is written into the buffer, and the rotation matrix commands are altered to have the same values as contained in the new matrix.

<u>Example</u> Suppose we have a set point with name SP, a Set C command word with name SC, and immediately following it in display sequence a pair of rotation matrix commands R.

(a) To initiate pen drag

RLT(9, SP, SPEED, DIRN, 7) $,

Note: SPEED and DIRN are ignored in this case.

(b) To initiate scaling with magnification greater than actual size, under control of the right shaft encoder:

```
RLT(6, R, SPEED, DIRN, 3, SC) $,
```

(c) To initiate the standardized rotation, under control of the crystal ball:

```
RLT (16, R)
```

(d) To terminate all the above:

```
RLTRMV (SP, 9) $,
RLTRMV (R)     $,
```

CAMERA (N#, NOTNOW#*)

Use:	To control the synchronized movie camera. The camera will be started immediately (unless NOTNOW is specified) and will be started again each time button 4 is pushed.
N#	The number of frames to be taken each time the camera is started. If N is 0, the camera is disconnected from button 4. Maximum value of N is 32767.
NOTNOW#*	If this argument is present and nonzero the camera will not be turned on until button 4 is pushed. Otherwise it will be turned on immediately for N frames and again for N frames on each occurrence of button 4.
Note:	If button 4 is pushed while the camera is running, it is stopped immediately. The next push of button 4 resets the count to N and starts the camera again. If CAMERA is called while the camera is running, the count is reset and the camera continues running (unless NOTNOW is set).

L. HOW TO USE GRAPHSYS

The GRAPHSYS procedures are in the CTSS as a library file called KLULIB BSS and are packaged either singly or in small interrelated groups. Hence, as far as possible, only those required by a user's program will be loaded.

1. Calling the Procedures

An example is given below of the method of calling a procedure from each of the main languages. Suppose the GRAPHSYS procedure is EXAMPLE (ARG1, ARG2, ARG3)

(i) AED-0 EXAMPLE(ARG1, ARG2, ARG3) $,

(ii) MAD EXAMPL. (ARG1, ARG2, ARG3)

Integer or pointer valued procedures should be declared to be type INTEGER, and boolean valued procedures to be type BOOLEAN, e.g.,

```
INTEGER PLOT., LIN., DPN, DELX, DELY, DELZ
DPN = PLOT.(LIN.(DELX, DELY, DELZ))
```

When a pointer is required, the GRAPHSYS procedure P=PTR.(X) may be used. P is a 15-bit pointer to X. Remember that all arrays are forward stored in GRAPHSYS. MAD calls should therefore give the Nth element of an array as an argument wherever an array name is required.

```
INTEGER A
DIMENSION A(10)
 . . .
ATTN.(A(10))
```

Now A(10-I) in MAD corresponds to A(I) in AED.

(iii) <u>FAP</u>

```
TSX    $EXAMPL,4
TXH    ARG1
TXH    ARG2
TXH    ARG3
```

Note that except for AED-0, only the first six letters of the procedure name may be used.

2. <u>Loading</u>

To load with user program PROG write:

```
LOAD PROG (SYS)   KLULIB
```

or, since KLULIB is properly ordered, the more effcient (SRCH) function of the LAED loader may be used:

```
LINK KLULIB BSS M1416 CMFL04
LAED PROG (SRCH) KLULIB
```

3. <u>Free Storage</u>

GRAPHSYS uses the AED-0 free storage[2] system for dynamically allocating memory space for its data structure. AED programmers requiring free storage for their own programs should note which of the free-storage procedures are automatically loaded with GRAPHSYS. They are FRED, FREE, FREC, FRET, and FREZ, as well as the associated procedures that these use internally. These procedures should <u>not</u> be loaded again with the user's program. The GRAPHSYS always takes memory locations from, and returns them to, the "open free storage zone".

4. Coons' Surfaces and Graphs

KLULIB contains routines for plotting Coons' surfaces and for plotting graphs. These facilities are described in MAC memoranda MAC-M-252 and MAC-M-224 respectively.

M. CONSOLE SIMULATOR

1. General Description

A simulator allows checkout of programs without using the console itself. No changes at all are needed in the user's program. The simulator replaces all communications between GRAPHSYS and the Display Controller, and requests the user to type in information that under normal operation would come from the Controller. This includes the attention array, the rotation matrix buffers, and all of the passive inputs described in Section I.2. The display buffer size is arbitrarily set to 2000(8), which is approximately the size available in practice when one user occupies both console stations.

2. Loading the Simulator

The simulator, KLUSIM BSS, may be linked to in M1416 CMFL04. It contains procedures that simulate those with identical calls in the console library KLULIB. Hence KLUSIM must be loaded before the KLULIB routines:

LAED PROG (SRCH) KLUSIM (SRCH) KLULIB

3. Operating the Simulator

The simulator expects only numbers as input. A number followed by the letter "C" is taken as octal, otherwise decimal. Numbers may be separated by spaces, commas, or carriage returns.

(i) ATTN (ATAR##)

The user must type in the complete contents of the attention array ATAR. For example, suppose we want to simulate the pushing of button 6, when the pen is tracking at position (100(8), 200(8)). On calling ATTN the simulator types:

```
Sim:  TYPE ATTN ARRAY
User: 3   6   100C   200C
```

The first number N is the zeroth register of the attention array. Hence the total number of items typed is always N+1. (See ATTN, Section K.6.a). The special number 77777C resets the ATTN simulator and allows the user to retype an uncompleted attention.

(ii) RWROT (NAME, MATRIX##, RW#, ERRLBL*)

When RWROT is called for reading, the simulator types:

```
TYPE 0 FOR IDENTITY MATRIX, 1 FOR 10 LINES
```

If the user types:

```
0
```

then the following matrix is set automatically:

```
377777777777C
000000000000C
000000000000C
000000000000C
377777777777C
000000000000C
000000000000C
000000000000C
377777777777C
377777777777C
```

If the user types:

```
1
```

Then he should type the ten numbers for the matrix setting. (Remember that numbers must be followed by the letter "C".) Note that input is required only when RW=1, i.e., when reading from the rotation matrix buffer is being simulated. When RW=0 (writing into the buffer) there is no action. (See RWROT, Section K.7)

(iii) Passive Inputs

All the procedures of Section K.6.b may be simulated, and all work similarly. For example on calling TOGA( ) the simulator types:

```
TYPE TOGA:
```

The user replies with the desired setting of the upper bank of toggle switches, an input string of digits in decimal. The number must be followed by a "C" if it is octal.

N. PROCEDURE INDEX

The first number gives the page where the procedure is discussed, the second that of its detailed description.

CC	=	MASEPOI (H#, V#, TYPE#*)	3.16	3.33
CC	=	MASGLE(CHARACTER#, TYPE#*)	3.16	3.34
CC	=	MASPEC(TYPE#*)	3.16	3.35
CC	=	MAZWD(DELZ#)	3.16	3.34
OB	=	NEVIS(OB)	3.17	3.36
OB	=	PACKED(STRING)	3.15	3.29
PI	=	PENH( )	3.21	3.43
PI	=	PENV( )	3.21	3.43
DPN	=	PLOT(OBJECT, NAME*, PNAME*)	3.9	3.25
OB	=	PNT( )	3.14	3.28
PI	=	REDIGI(LEFT#, RIGHT#*)	3.21	3.43
OB	=	ROT(MATRIX##*)	3.14	3.32
		ROTMUL(OUTPUT##, INPUT##*, AXIS#*, ANGLE+*)	3.16	3.35
		RMV(HERE*, THERE*)	3.13	3.26
DPN	=	RPL(NEW, INAME, NAME*)	3.13	3.27
		RLT(FCN#, NAME, SPEED#, DIRN#, CONTROL#, CNAME*)	3.22	3.43
		RLTRMV(NAME*, FCN#*)	3.24	3.45
B	=	RTNCOD(ONOFF≠)	3.18	3.38
		RWROT(NAME, MATRIX##, RW#)	3.24	3.45
N	=	SATBUF(N#*)	3.18	3.38
OB	=	SETC(MASTER#**, SLAVE#**, MAG#**, INTEN#**, LP1#**, LP2#**)	3.14	3.33
OB	=	SETF(MOVH#, SCALH#, MOVV#, SCALV#, MOVI#, NOPLOT#)	3.15	3.33
OB	=	SETPT(H#, V#)	3.14	3.27
		SGNOFF( )	3.18	3.37
		SGNON(NCON#)	3.18	3.37
PI	=	SHENC( )	3.21	3.43
PI	=	SHENL( )	3.21	3.43
PI	=	SHENR( )	3.21	3.43
OB	=	SINGLE(CHARACTER#)	3.15	3.29
OB	=	SPECIAL(POINTER)	3.15	3.29
		SRLBUF(N#*)	3.22	3.45
OB	=	TEXT(STRING)	3.15	3.31

PI = TOGA( )	3.21	3.43
PI = TOGB( )	3.21	3.43
N = VAC( )	3.19	3.40
VIS(HERE*, THERE*)	3.17	3.37

REFERENCES

1. Ross, D. T., AED-0 Programming Manual - Preliminary Releases Nos. 1-4, October - December, 1964.

2. Ross, D. T., New Free Storage Package, Electronic Systems Laboratory Memorandum 9442-M-173/MAC-M-318, July 1966. Also The AED Free Storage Package. Communications of the ACM, Vol. 10, No. 8, August, 1967, pp. 481-492.

APPENDIX A

THE DESIGN AND PROGRAMMING OF A DISPLAY INTERFACE SYSTEM INTEGRATING MULTIACCESS AND SATELLITE COMPUTERS

by

D. T. Ross, R. H. Stotz, D. E. Thornhill

and

C. A. Lang
University Mathematical Laboratory
Cambridge, England

This paper was presented at the ACM/SHARE Fourth Annual Design Automation Workshop held in Los Angeles, California, June 26-28, 1967.

ABSTRACT

Application of highly interactive computer graphics to large, complex problems imposes stringent requirements for both real-time response and massive computation. The needed computation can be supplied most economically by a large multiaccess computer, but the necessary real-time service must be provided by a satellite display computer, which is directly connected to one or more display terminals. This paper describes a Display Interface System which allows the user to employ these facilities in a manner which is independent not only of his particular problem, but also of the computer, communication link, display hardware, and the multiaccess operating system.

The major design criteria are that maximum real-time capability (both computation and storage) be made available to the user, with minimum overhead on the multiaccess system. These criteria are met by employing <u>minimal executive</u> programs in both the multiaccess and display computers. These minimal executives may be augmented from libraries of standard executive routines and utility routines to form a particular user's operating system.

The work reported in this document has been made possible through support and sponsorship extended to the M.I.T. Electronic Systems Laboratory by the Manufacturing Technology Laboratory, RTD, Wright-Patterson Air Force Base under Contract F33615-67-C-1530. Project DSR 70429.

The work reported herein was supported (in part) by Project MAC, an M.I.T. Research Program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract NOnr-4102(01).

I. INTRODUCTION

This paper describes a system for providing mixed graphical and verbal communications with a multiaccess computer via a satellite computer with a display. We assume that such an arrangement is necessary to satisfy an on-line user's large-scale computation and storage requirements economically, and to provide graphical facilities at remote terminals. The general-purpose Display Interface System, parts of which reside in each machine, acts as an interface between the user's programs and the display terminal. It provides means for communication as well as system functions required in each machine. Since it is general purpose, and no user will require all the features it provides, the software exists in the form of system libraries from which a user may select an appropriate subset for his particular application. The user's interface with this system in each machine is intended to be hardware independent.

II. SELECTION OF SYSTEM COMPONENTS

Application of highly interactive computer graphics to large, complex design problems imposes stringent requirements for both real-time response and extensive computation and storage facilities. If a sophisticated graphics console is to provide acceptable interaction with a user, certain real-time actions are necessary:

1. The picture must be maintained on the screen at all times by <u>executing</u> a <u>display file</u> of commands to the display unit sufficiently often to minimize flicker,

2. There must be very fast response to <u>interrupts</u>, (e.g., light pen "see", button push), and

3. Execution of real-time programs must be provided at regular intervals in order that certain transformations on the screen (e.g., rotation of a three-dimensional picture) appear to be taking place continuously.

In addition to these real-time requirements, the user also needs occasional extensive computation for manipulating his data structure, for analysis, or for simulation. Finally, he needs mass storage to retain his data and programs.

It is uneconomic for a single on-line user to operate a computer with facilities which he needs only occasionally. Multiaccess and multiprogramming techniques reduce the costs by sharing the computer facilities among several users. The high data-rates and real-time interaction requirements of displays, however, are incompatible with the standard time-sharing and core-swapping techniques of such systems. If a display is attached directly to a multiaccess computer, then core must be permanently assigned to store the real-time programs and the display file mentioned above. Sufficiently fast response can not be given if these have to be retrieved from backing store when they are required. Further, many display-oriented actions would have to be given highest priority at all times over other users' programs. Such a system might be workable for a single display user within the multiaccess system, but if several displays were required, serious conflicts of storage and time would occur. These conflicts have been experienced at M.I.T. where the Electronic Systems Laboratory display console has been connected directly to the IBM 7094 time-sharing system since 1964.[1]

A possible solution might be to provide the display with its own buffer store for the display file. Not only would this remove the necessity to store the display file in the multiaccess computer, but it would allow the display to be maintained remotely, connected to the multiaccess computer by means of a communications link. Such a simple buffering scheme is inadequate, however, for the real-time program requirements can only be satisfied if the bandwidth of the link is sufficiently high, and still the display priorities would disrupt normal time-sharing scheduling. The only complete solution is to provide remotely at the display some computational power which may be dedicated to the requirements of the display without affecting other programs in the multiaccess computer.

Figure A.1 shows the system design which has resulted. A large, general-purpose, multiaccess computer (MC) is coupled with a dedicated general-purpose display computer (DC) to drive a number of teletypewriter and display terminal stations. A large number of users have access from remotely-located keyboard consoles such as teletypewriters to the MC. The MC contains mass storage, a file retrieval mechanism, editing programs, compilers for numerous languages, and other similar programming aids. User programs are run inter-

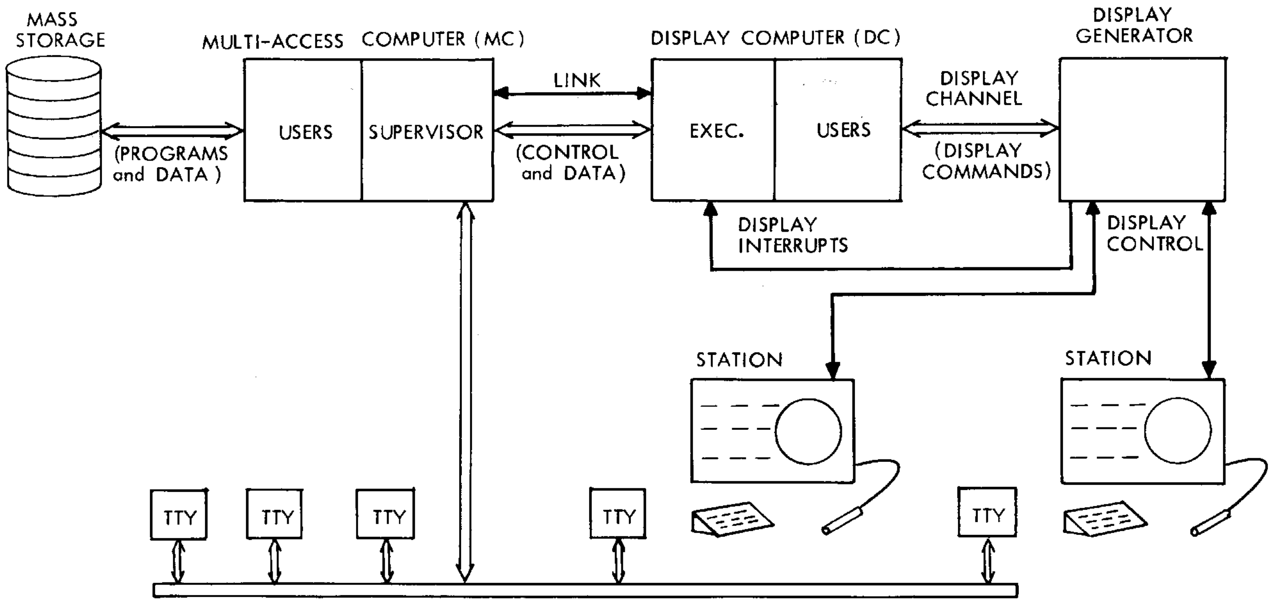


Fig. A.1 General Hardware Scheme

mittently in accordance with the time-sharing scheduling algorithm and any individual user program is operated only aperiodically for short bursts of time. For users at keyboard consoles, the effects of this swapping are essentially unnoticeable if the system is well designed and is not overloaded.

Connected to the MC by a communications link is the satellite display computer (DC) which stores the display file and meets the real-time requirements of the display station users. The DC may be large, medium, or small, depending upon the requirements of the users, the characteristics of the communications link, and other factors. Directly coupled to the DC are the graphic consoles which provide graphical displays, light pens, knobs, push-buttons, etc. (Teletypewriters associated with the graphic consoles may be connected to the DC instead of the MC as shown in Fig. A.1.)

The generalized system design of Fig. A.1 can be made to satisfy a wide range of requirements coupling dynamic displays with a time-shared environment. The principal problem to be faced is how to achieve a complementary general software system design, so that a single programming philosophy can be applied to the full range of system configurations. This paper presents such a display interface system design which not only is applicable to many choices of hardware, but which also allows maximum flexibility to the user for any particular hardware selection.

III. THE DISPLAY INTERFACE SYSTEM[2,3,4]

Figure A.2 shows the user's view of the system. His problem is represented by programs and data structure which are coupled to a

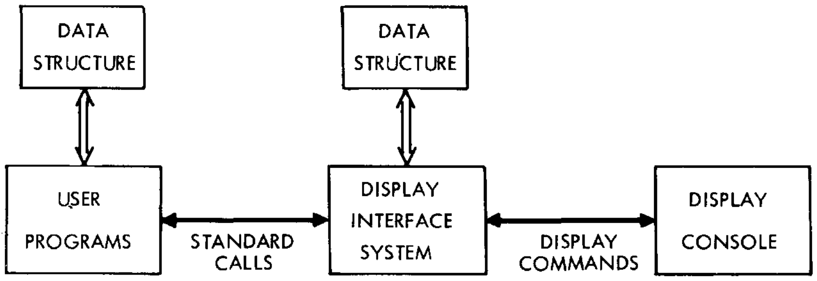


Fig. A.2 User's View of the System

display console station through a <u>Display Interface System</u> (DIS). It is very important to realize that the user program does not affect the display directly, but only through the standardized calls for actions to be performed by the intermediary interface system. Only by such a scheme can the user programs remain constant as display and hardware configurations change. All of the device- and system-dependent characteristics are absorbed into the inner workings of the DIS.

As far as the user is concerned, the standard DIS is a set of system-provided procedures which allow him to employ the facilities of the display system without a detailed knowledge of how it works. The DIS allows the user to create pictures, perform real-time computation, interpret display console actions, allocate DC resources, etc. The DIS does not retain any information relevant to the user's problem. For example, the user may create a picture, giving to each picture part (e.g., point, line, or subroutine picture) a <u>name</u> which will then be used for all communications between the DIS and the user's program. The user might say, "Plot a point at position (x, y) andname it

N1," or "Delete the object named N2," or the system might inform the user, "Picture part N3 has been seen by the light pen." The interface system builds the necessary display commands and inserts them into the display file for the picture being displayed. The DIS maintains information about the correspondence between those commands and the user-given names, but any geometric or structural information about the picture is contained in the user's own program and data structure.

With the general hardware configuration and user's view established, we now may turn to the design of the Display Interface System itself. For this discussion, it is useful to make a distinction between the <u>user</u> of the system (as described above) and a <u>system programmer</u> (who may be the same individual) who adapts the various features of the DIS to a particular hardware configuration and set of usage characteristics. In other words, the generalized DIS design can be adjusted to account for frequency-of-use and real-time requirements, etc., needed to satisfy a given style of use, as well as hardware characteristics. These adjustments are performed by the programmer so that the user need not be concerned with the workings of the DIS. The alternatives available to the programmer are the subject of the remainder of this paper.

IV. SOFTWARE DESIGN CRITERIA

The DC, however large, has limited capacity. Therefore the primary goal is to achieve a software system design in which the maximum amount of DC capacity (storage and execution time) is made available for assignment as best meets the needs of the individual user. To put the matter another way, a minimum of fixed system overhead should be compulsory, but there should be a maximum of optional system assistance which the programmer can make available to the user. The basic system design must reflect only those aspects which have their basis in the physical nature of the hardware configuration and the fundamental nature of communication, and are therefore required by all users. Then each type of usage can be handled by suitably augmenting the hard core of the system.

The minimum DIS structure which is dictated by hardware configuration is concerned with controlling the communication link between the MC and the DC, and providing a skeleton executive structure to which augmenting features may be attached. We refer to this basic core as the minimal executive and it resides partially in the MC and partially in the DC. In both the MC and the DC there is a communications package which is able to transmit and receive "messages" consisting of programs, display file commands, data structure, or any other form of data, and also of control information requesting or demanding that the recipient take some action by executing a program.

In the MC, the minimal executive is imbedded in the multi-access system supervisor and coupled with the standardized calling mechanism of the DIS. In the DC, however, the minimal executive contains two more almost trivial parts, an interrupt receiver and a program cycle. Their purpose is solely to satisfy hardware needs and to provide tie points for augmenting the minimal executive for particular styles of usage. The minimal interrupt receiver dispatches interrupts from the link to the communications package for processing, but ignores all other interrupts. The program cycle merely keeps the DC running. Thus the DC minimal executive does nothing but await augmenting instructions.

The overall DIS design provides many features for the user, and in principle almost all of these features can be carried out by programs and data structures residing in either the MC or the DC or partially in both. The design is predicated on the assumption that there will be a library of system routines (residing on the backing store of the MC or DC or both) which may be loaded and linked into the appropriate places in the minimal executives. We will assume that the DIS includes the capability for this system assembly, and will not concern ourselves here with the methods whereby the system programmer assembles a particular arrangement for a particular user. It is worthwhile, however, to consider in more detail the various major forms of augmentation and how they can satisfy the criteria for flexibility and efficiency stated earlier.

V. AUGMENTING THE DC EXECUTIVE

Certain augmentations must take place exclusively in the DC. The simplest of these is the Interrupt Dispatcher whereby the user may attach a program of his choice to any individual interrupt of the DC (excepting those concerned with the communication link which must be handled rigidly by the system itself). In addition, there are the attention and program queue mechanisms, as well as provision for multi-station time-sharing of the DC itself, which merit elaboration. Fig. A.3

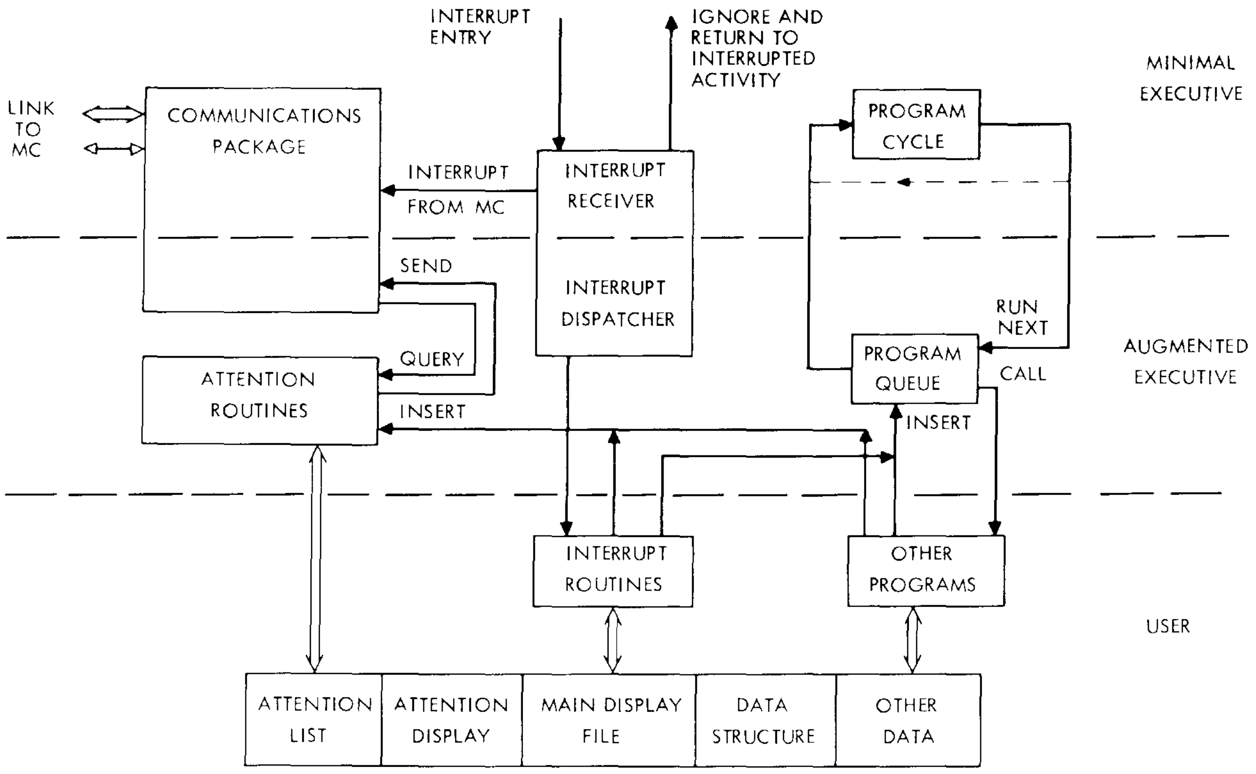


Fig. A.3 Programs in the Display Computer

shows the complete contents of the satellite display computer. It should be kept in mind that only the minimal executive is essential, so that any of the other items may not be present for a particular user's application. Figure A.3 shows how these parts are interconnected when all are present.

1. The Attention Routines

The attention routines allow the DC to initiate an information transfer to the MC. These routines queue blocks of data (called attentions) which are to be sent to the MC in a first in-first out attention list. As soon as the link is free after the creation of such a block, the user's MC program is interrupted and informed that an attention is waiting. The user may then read this top attention from the DC, process it, and return to the state in which he is again prepared to be interrupted. Only when the top attention has been read is the information about the next attention (if there is a next one) available to the MC. The attention may contain arbitrary information including requests to the user's MC program to read or write further data to the DC.

The attention mechanism has two important properties. First, by having the ability to interrupt the MC program it allows the user's DC and MC programs to work asynchronously if desired. Needless to say this in no way prevents them from being operated synchronously. Secondly, it enables the user to work continuously on the DC even though his MC program is only running intermittently. For example, whenever a light pen or push button action requires interpretation by an MC program, an appropriate attention is added to the queue in real time, and the user may continue immediately. Only when a desired action depends upon some displayed result of an earlier action interpreted by an MC program will the user be dependent upon MC time-sharing response.

A useful option is the attention display, whereby a mnemonic display is automatically maintained on the screen for each attention as long as it is in the queue. As soon as an attention is read by the MC its display is deleted. This serves to leave a history on the screen of actions taken by the user, and to provide feedback about actions taken but not yet processed by the MC. In many cases the attention display can be treated as a token for some future result from an MC program, so that time-sharing delays are further alleviated.

2. The Program Queue

The program queue is actually a queue of queues, headed by a stack, as shown in Fig. A.4. All manipulations of this mechanism are handled on a priority interrupt basis, where the manipulations are as minimal as possible. Each mark below the line in Fig. A.4 represents a complete call (function name and argument values) for some action to be performed. Insertions may be made at various places, depending upon the nature of the action and the status of the action currently being performed. If in Fig. A.4 the inserted program is to be left to the current interrupted program and there is a bar above the line between

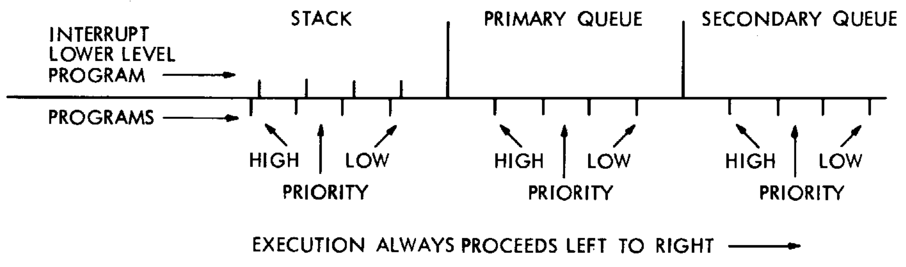


Fig. A.4 The Program Queue

them, then the inserted program is begun and is run to completion before the interrupted program is resumed. This may happen repeatedly, and whenever a program is completed, the leftmost program is resumed or begun.

This rather elaborate scheme is made necessary by the real-time requirements of dynamic displays and the desire to maintain flicker-free pictures in spite of speed limitations of the DC, as much as possible. Calls to place actions on the queue may be imbedded throughout the display file, coupled to interrupts, tied to real-time clocks, etc., so that the user (or his system programmer) may have full control of the way in which conflicting time requirements are met.

Another feature of the program queue conventions is that in many cases the results of actions also are queued and are made active only at specific times by other actions. Thus, for example, if a series of actions compute new values for variables which affect the appearance of the display, the display continues to cycle based upon the old values until all of the new values can be made available at the same time. This technique prevents the breaking up of the picture which otherwise would result.

3. Multiple Users in the DC

When the DC is to be time-shared between several graphic terminal users, the augmented executive must include routines for scheduling processor time and display channel time among the users and for protecting them from each other. This requires more elaborate attention queue, program queue, and interrupt processor routines. Separate attention queues must be maintained for each user, and the interrupt dispatcher must be able to decide to which user an interrupt belongs and to call his routine. The display cycling routines and the processing of interrupts directly involved in cycling the display must now become a part of the augmented executive. Memory protection hardware in the DC is probably necessary to ensure the users do not damage each other's program or the executive.

VI. AUGMENTING THE MC EXECUTIVE

As was mentioned above, the remaining features of the Display Interface System may be mechanized in the MC or DC or both. Since it is most likely that the DC will be a small computer so that the criterion that the maximum capability be available to the user becomes paramount, we will describe the remaining features primarily from the viewpoint of implementation in the MC, with occasional indications of how portions can be carried out in the DC. In the following discussion, as is indicated in Fig. 5, only the communications package is imbedded

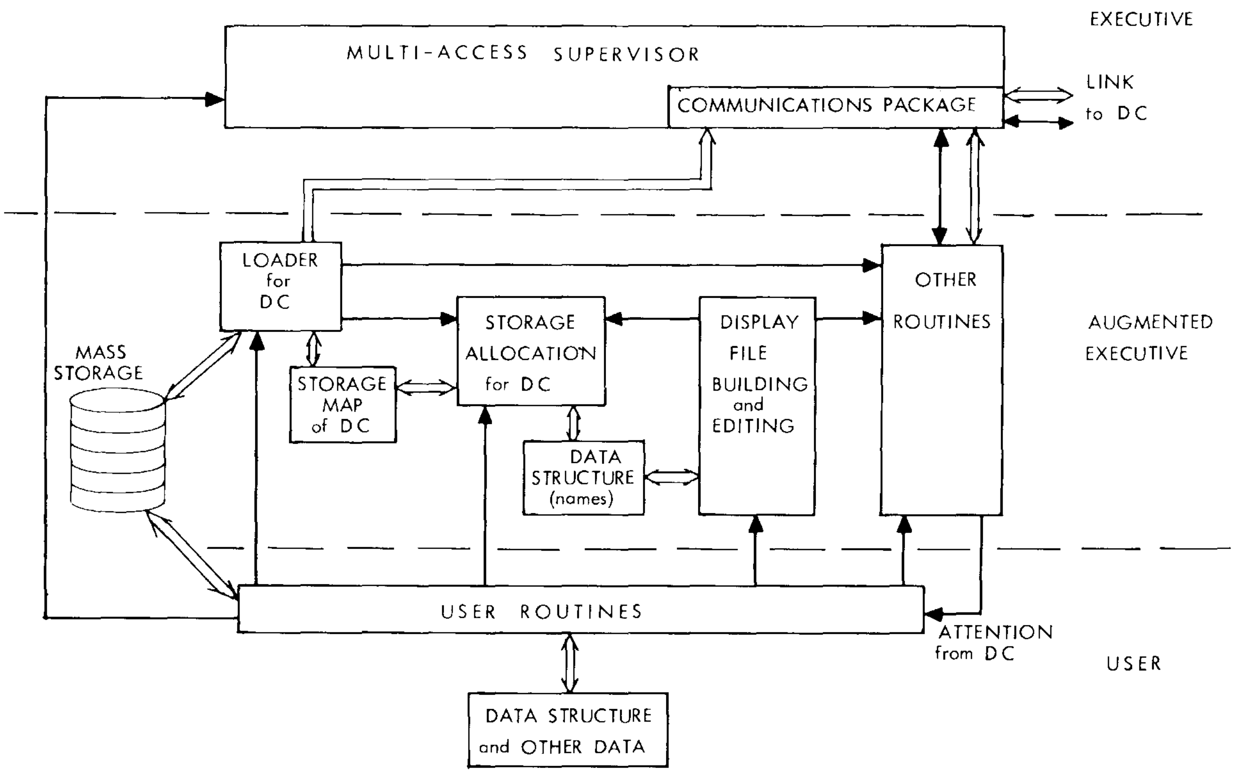


Fig. A.5 Programs in the Multiaccess Computer

in the Multiaccess Supervisor of the MC. All other augmenting of the MC executive takes place in space allocated to the user by the supervisor.

1. DC Storage Allocation[5]

If the DC is small, the task of allocating storage in DC memory will almost always be split between the MC and the DC. Every feature of the DIS requires some form of storage in the DC for programs, data structure, or display file. If the DC memory is not large enough to contain the type of elaborate free storage allocation mechanism required for all purposes, then the elaborate allocations may be performed in the MC in non-real-time. Almost always, however, the DC will need at least a basic free storage allocation system for the attention queue, attention displays, etc., since these requirements must be met in real time. As Fig. 5 indicates, when some of this function is performed in the MC, the MC maintains a storage map of DC memory usage, including those segments turned over to the management of the DC free-storage system.

2. Display File Building and Editing[3,4]

Routines to build and edit display files in response to calls on the standard DIS interface usually reside in the MC so that the maximum DC memory is available for display files. Basic routines to build standard picture parts, such as points, lines, characters, circles, etc., and to control display parameters such as intensity, and sensitivity to the light pen are provided. When one of these routines is called, it builds the appropriate display file commands, obtains space for them from the storage allocation routines, transmits them to the DC, and inserts them into the display file. More complicated picture parts may be built consisting of any combination of the standard picture parts above. These are built up in the MC and then inserted into the display file as a single part. They behave rather like a macro, for once they have been built in the MC, any number of copies may be transmitted. A further facility allows a subroutine picture to be added once to the display, and any number of calls may be made to it from the display file. This is useful for repetitive pictures.

A more important provision of these routines is a feature whereby the user may assign a <u>name</u> to any picture part, or subroutine picture definition into the display file. This name may then be used for communication between the user and the routines. The routines provide editing facilities for the display file, and the names are used, for example, to specify which item to delete, or where in the display file to insert a new item. When a light pen interrupt occurs the routines tell the user the name of the item seen. If this name is a pointer to that part of his data structure describing the item seen, it gives very quick access to all the relevant data about the object seen.

In order to carry out the above functions, these routines maintain in the MC a map of the display file, and the relation between the names and the corresponding display commands in the DC. No geometric or structural information about the picture is maintained, however, for that is a function of the user's problem.[6, 7]

3. Loading of DC Programs

Just as display file generation will usually take place in the MC, the complexities of DC program loading and linking also require MC action and large libraries of DC programs require the MC mass storage facilities. The DC space for loading programs is obtained from the storage allocation mechanism, and the structure of the DC executive, the program queue mechanism, and the display file itself provide the tie points to which programs may be attached. If the programs are stored in relocatable form, any necessary relocation is performed in the MC and the resulting relocated programs are transmitted to the DC. This takes place both for the initial loading of the DC with the augmented executive and user routines and for dynamic loading to make maximum use of DC storage. The exact mechanism depends on the paging and relocation hardware of the DC (if any).

Most DC programs will be selected from fail-safe library routines stored in the MC mass memory, but special user-written programs may also be compiled or assembled on the MC for transmission to the DC. The compiler or assembler must contain special provisions to guarantee that new programs cannot damage the system if they malfunction. Memory protection hardware for the DC would be helpful in this regard. Finally, it is worth noting at this point that it is certainly possible for the DC to have its own loader and a backing store on which the library of system routines are kept. This alternative may be particularly attractive if the link between the two computers is of low bandwidth or if dynamic swapping of DC storage areas is desired. The DC could then manage all its own storage allocation and loading.

VII. DISCUSSION OF SYSTEM USAGE

Now that the program-building, loading, and sending routines have been introduced, the needs of the user who requires the full flexibility of the system can be considered. The programmer has at his disposal not only such of the standard system routines for each computer as he desires to employ, but also facility for writing his own additional (or substitute) routines for each machine. Factors which must be considered include the storage and computation facilities of the two machines, the relative cost of storage and computation in the two machines, the bandwidth of the link, and the type of response that is required at the display for the problem which the program is designed to solve. One can visualize the two extremes under which the interconnected system could operate. At one extreme the MC might be used only to transfer data between its backing store and the DC, while at the other extreme extensive computation might be done in the MC, with the DC being used only to output results on the display for direct viewing or photographing. In between there are many levels where the programs in each machine will want to communicate. For example, one might design a casting on-line at the DC using the display, light pen, and other input/output devices and then press a button which causes a description of the geometry of the casting to be transmitted to the MC where an analysis is carried out to compute the weight of metal in the casting. The results are then transmitted back to the satellite and displayed.

Put more generally the spectrum of uses covers those who require pictures, perhaps very elaborate, with little or no dynamic action to those who require pictures, perhaps quite simple, with a great deal of real-time computation. Our aim therefore has been to provide general-purpose communication software between the machines independent of the bandwidth of the link and of the hardware as far as possible, along with a library of routines for performing system functions in each machine in a compatible way.

VIII. IMPLEMENTATION

Systems based on the above philosophy are being implemented at the Mathematical Laboratory, Cambridge University, where a PDP-7/340 is connected to an Atlas II computer,[8] and at M.I.T. where a PDP-7 driving the Electronic Systems Laboratory Display is connected to an IBM 7094.[2] When the M.I.T. PDP-7 and display are connected to the GE 645 which is due to replace the 7094, only the communications portion of the minimal executive must be reprogrammed. The other 7094 routines, which are written in the machine-independent AED language, and the PDP-7 routines can still be used.

REFERENCES

1. Ross, D. T., et al., "A-Core/B-Core Display Interface for Time-Sharing," in Interim Engineering Progress Report, ESL-IR-221 for 1 December 1963 through 30 May 1964, M.I.T. Electronic Systems Laboratory, pp. 35-37.

2. Ross, D. T., "The Display Interface System," ESL Memorandum 9442-M-170/MAC-M-312, June 1966, pp. 20-28. Also appears in Interim Engineering Progress Report, ESL-IR-278 for 1 June 1965 through 31 May 1966, M.I.T. Electronic Systems Laboratory, pp. 43-54.

3. Lang, C. A., "New B-Core System for Programming the ESL Display Console," ESL Memorandum 9442-M-122/MAC-M-216, M.I.T. Electronic Systems Laboratory, April 1965.

4. Mozley, A. N., "A Display Interface System," University Mathematical Laboratory, Cambridge. To be published.

5. McCallum, K., "Titan Routines for Controlling PDP Free Storage," University Mathematical Laboratory, Cambridge, February 1967.

6. Gray, J. C., Lang, C. A., "ASP - An Associative Data Structuring Package," University Mathematical Laboratory, Cambridge. To be published.

7. Newman, W., "The ASP-7 Ring Structure Processor," Imperial College, London, January 1967.

8. Lang, C. A., "The PDP-Titan Link," University Mathematical Laboratory, Cambridge, November 1966.

APPENDIX B

A SAMPLE INTERACTIVE GRAPHICS PROGRAM

by

Daniel E. Thornhill,
John W. Brackett, and
Jorge E. Rodriguez

This program was prepared as a classroom example for a course on Computer Graphics presented by the Special Interest Committee on Time-Sharing of the Greater Boston Chapter of the Association for Computing Machinery on February 10, 1968 at M.I.T.

I. INTRODUCTION

Any interactive graphics program is more complex than the usual one page "sample program" unless it is only 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.

The problem chosen is drawing and editing a simple electrical circuit. The program allows the user to construct a circuit containing nodes, capacitors, resistors, and short circuits, to modify the circuit, and to assign names and values to resistors and capacitors. Obviously, such a 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 analysis routines, which are completely missing in this case. Therefore, the sample program should be regarded as being only a small fraction of the size and complexity of a useful on-line circuit design system.

II. PROGRAM FUNCTIONS

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

1. To begin drawing the circuit at a fixed point on the screen where the first node is always defined. A node is a point of intersection of circuit elements.
2. To draw, from the node last pointed to with the light pen, an element which may be:
   a. a horizontal or vertical resistor,
   b. a horizontal or vertical capacitor,
   c. a horizontal or vertical short circuit.
3. To delete a node or an element pointed to by the light pen.
4. To give a name, which will appear on the screen, to a resistor or capacitor.
5. To assign a value to a resistor or capacitor.

The user operates the program by using a light pen to indicate the node or element of interest and a box of push buttons to indicate the function to be performed on the object to which he has already pointed with the light pen. Values and names for the capacitors and resistors are input on the teletype.

III. 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 co-ordinates.
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 data structure.

One way to store all this information would be to use four Fortran-type arrays, one for nodes and one for each of the other three types of elements. For nodes, each node could be a row of a matrix, with the number of columns being the number of values needed to represent the information about the node.

There are several disadvantages to this scheme. If arrays must be dimensioned at compile time, the person who writes the program must decide upon the maximum number of nodes and of each element the user can define. Since the user is to be allowed to edit the circuit on-line, the program must also be able to keep track of what rows in the matrices correspond to deleted nodes and to reuse these rows if the total number of nodes defined at any time begins to approach the dimension of the array. For programming convenience, it would be useful to denote that the fourth element in each row of the node matrix is to have the name "XCOORD" to indicate the X coordinate of the node. This cannot be done directly in FORTRAN, since it would violate the assumption that all elements of an array are similar entities, but can be done indirectly by the clever use of "EQUIVALENCE" statements.

Many computer representations of these elements and their interconnections can be devised. We shall use the AED (Algol Extended for Design) System which has been designed by the M.I.T. Electronic Systems Laboratory Computer Application Group and which operates on the M.I.T. time-sharing system. The circuit will be modeled by using a bead or block of words to represent each node or element. Beads are obtained from a free storage system (a storage management system). The AED components feature will allow us to refer symbolically to each property of an item to be stored in the bead. Pointers will be used to interconnect beads. These techniques will be discussed in a later section.

IV. 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 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, and the first node is defined to be at $x=-400$, $y=400$.)

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

1. Obtain an <u>element</u> <u>bead</u> which will contain all the information about the element; the type of element is stored in the bead.
2. Find the node bead corresponding to the node the user pointed to with the light pen.
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.
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 nodes which it connects.

The data structure used to represent a simple circuit is shown in Fig. B.1. In addition to the information indicated previously to be stored for each element, there is a pointer chain connecting all nodes together. This chain, which permits the program to follow pointers from one node to another, is necessary since it is possible for a node to exist with no elements attached to it.

To delete an element pointed to by the light pen the program must:

1. Find the element bead corresponding to the element the user pointed to with the light pen.
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.
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 B.1 illustrates the data structure for a partially completed circuit. For node beads only the up, left, right and down

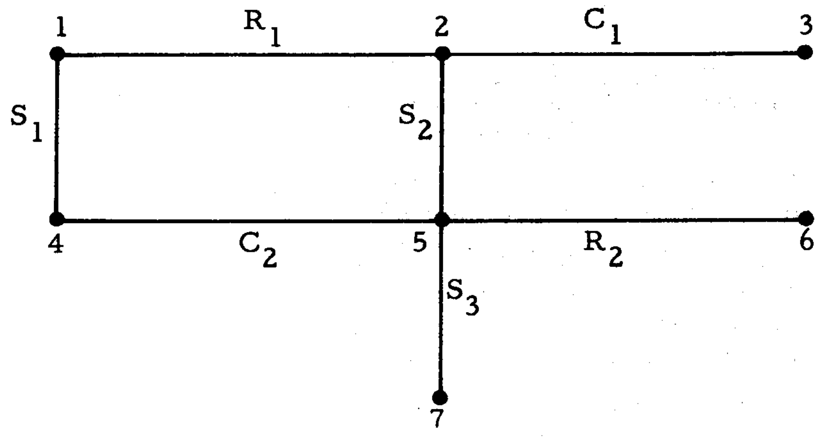


Fig. B.1a Schematic Diagram of a Partially Completed Circuit

TYPE = 0
LEADS
NEXTNODE
NAME
UP ELEMENT
LEFT ELEMENT
RIGHT ELEMENT
DOWN ELEMENT
X
Y

Node Bead

TYPE = T
FROMNODE
INTONODE
NAME
VALUE

T=1 Resistor Bead
T=2 Capacitor Bead

TYPE = 3
FROMNODE
INTONODE

Short Circuit Bead

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

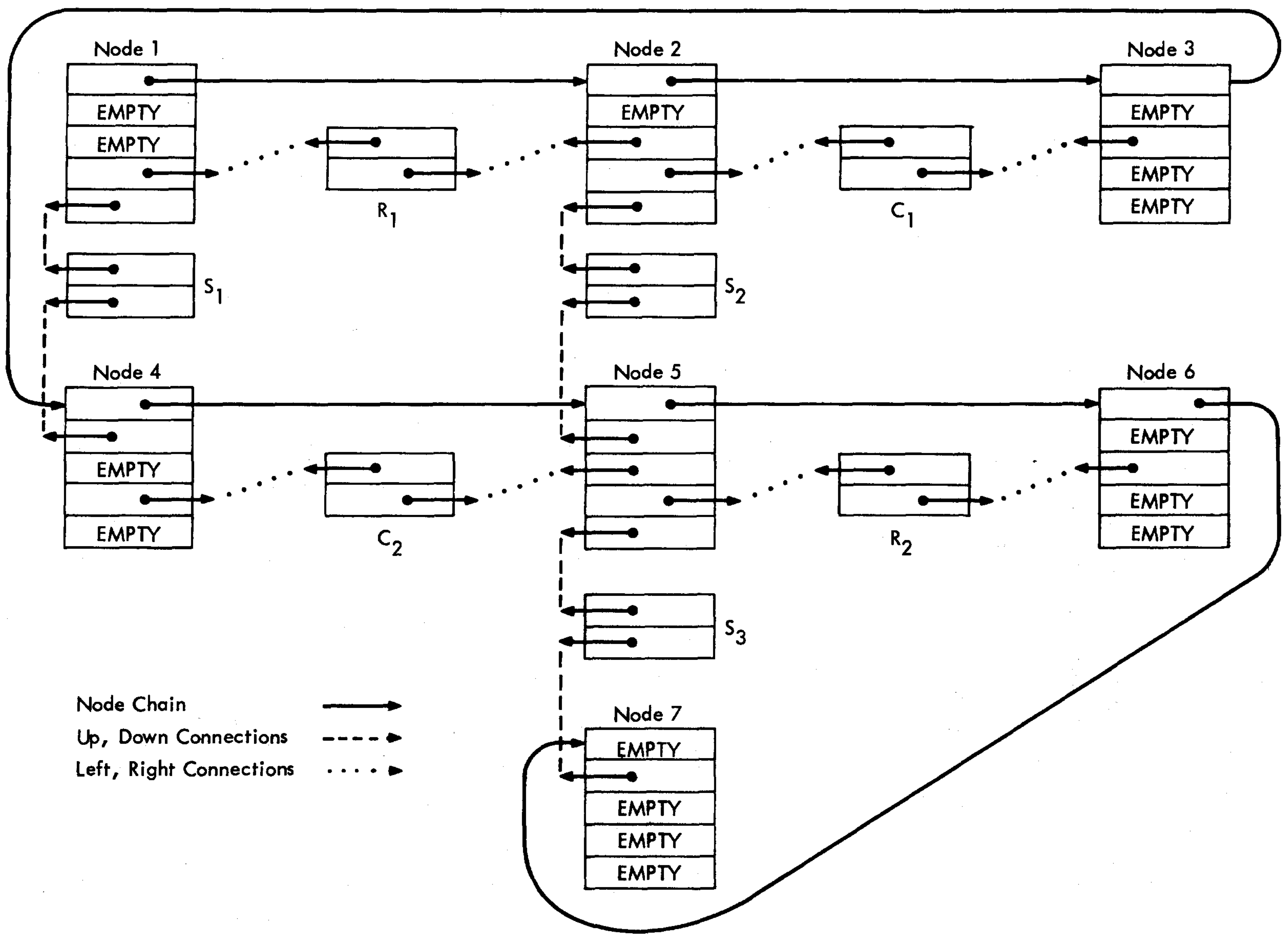


Fig. B.1c Data Structure for Partially Completed Circuit

element connections are shown; for resistor, capacitor and short circuit beads only the connections to the node beads are shown.

The most complicated program operation is deleting a node. If a node is deleted, all elements attached to it are to be deleted. Since these elements are also attached to other nodes, one must update 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 would also want to 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.

V. SPECIFICATION OF THE DISPLAY

The programming system that allows the display console to be used with the M.I.T. time-sharing system has been given the name GRAPHSYS. 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 the light pen. The way in which GRAPHSYS organizes its data is similar in some ways to the organization of the model of the circuit in the program; therefore, we will discuss the job of generating the graphical representation of the circuit before treating the program in more detail.

The display file is the ordered sequence of console commands which is sent to the display console to produce a picture. Although the 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).

When a object is placed in the display file, it must be given a unique name if it is to be identified again. The user may wish to specify some object on 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 light pen). The name is used to refer to an object in all communications between GRAPHSYS and the user.

The simplest type of unique name which can be assigned to an object is the starting location in the display buffer (the area of memory in which the display file resides) of its console commands. Every object is, of course, uniquely specified by this number. These locations are chosen by GRAPHSYS which automatically performs the tedious tasks of allocating space for the display file and organizing it. This is a dynamic process in which, without altering the display sequence, commands may be moved around in the display buffer, (e.g., when new commands are inserted into the middle of the display file). Display buffer locations, therefore, do not constitute a suitable naming scheme for the user, since the user would be forced to keep his own data structure continually updated as GRAPHSYS moved objects and thereby changed the names.

The simplest form of invariant name that can be assigned to an object is an integer number. If this type of name were chosen, GRAPHSYS would have to build an array of length equal to the total number of names. The name of an object would be the index for the position in the array containing the actual display buffer locations. In addition, the user would need a similar array giving the item in his data structure corresponding to the name. This is a lot of mechanism and requires the existence of large arrays. It would be much simpler if the user and GRAPHSYS had a mutually convenient single name for each object.

The naming scheme which has been employed by GRAPHSYS meets these criteria of uniqueness, invariance, and convenience. GRAPHSYS builds its own data structure which is a string of items arranged in display file sequence. Each item in the string represents an object and specifies its current display buffer location. The register in which the system stores its information is called the display register of the object. The name of the object is a pointer to (the core location of) its display register and is called the display pointer name. The user communicates with GRAPHSYS in terms of these names, and the system performs the transformation to and from display buffer positions for communications with the Display Controller.(See Fig. B.2)

This scheme so far guarantees a unique name for each object since the core locations of the display registers are unique numbers and

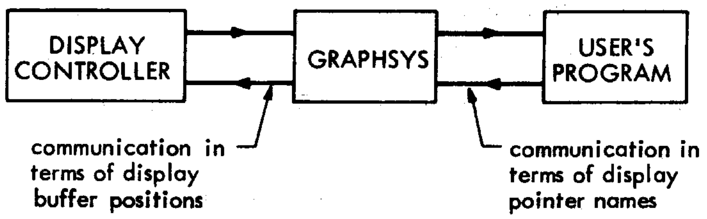
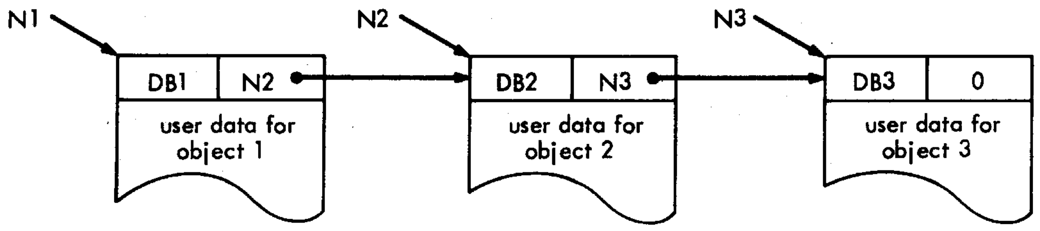

DISPLAY CONTROLLER
GRAPHSYS
USER'S PROGRAM
communication in terms of display buffer positions
communication in terms of display pointer names

Fig. B.2 Communications about Items in the Display File

eliminates the need for large arrays since the display registers can be obtained one at a time from a free storage list as needed. Finally, it provides a mutually convenient name for the user and GRAPHSYS by allowing the user to supply the display register as a part of his own data structure.

Figure B.3 illustrates the combined User-and-GRAPHSYS data structure for a display file containing three objects. In our example



N_i are the display pointer names of objects
DB_i are the display buffer locations of the display commands of the objects

Fig. B.3 The Combined User-and-GRAPHSYS Data Structure

the display register will be the first word in the bead, the block of registers representing a node or an element.

Each display register contains the display pointer name of the next object in sequence in the display file. A name of zero signifies the end of the list. The contents of the display registers are maintained solely by GRAPHSYS; the user does not have to be concerned with the display register string or the display buffer locations.

Only a small subset of the GRAPHSYS procedures are used in this example; in particular the facilities to define and call subpictures and to perform real-time operations, such as rotation, are not illustrated.

In order to build an object to be plotted, such as the representation of a resistor, a series of line generate commands must be stored in an array. In this example the display code for plotting an element such as a resistor consists of a point command to move the beam to the x and y coordinates where the resistor is to be drawn followed by the Δx, Δy increments for lines to draw the element. These commands are stored in an array, the zeroth element of which is the number of display commands to follow; this array is called a <u>display object.</u> The point command is generated in the program by the procedure MASEPOI. The procedure MALIGEC will generate the correct line generate commands from the arguments giving the increments in the x and y beam positions and whether the line should be visible or not. Individual characters can be plotted by using the procedure MASGLE; more elaborate procedures are also available in GRAPHSYS for dealing with strings of characters.

When the display object representing a resistor has been created, it must be added to the display file before the display can interpret the newly-created commands. The procedure PLOT will insert the designated object in the display file and will use the pointer provided as the GRAPHSYS display pointer name. An object may be removed from the display file by calling the procedure RMV and indicating the object to be removed by giving the display pointer name.

Pushing any of the 36 push buttons or the light pen seeing a picture part on the screen are real-time events that are known as <u>attentions</u>. The way these attentions are serviced is a function of operating the display in time-sharing. They cannot have a real-time effect on the user's program as it may not be running in the time-sharing system at the time of the interrupt. Rather, they must be stored by the time-sharing supervisor and made available to the user the next time he is in core. Of course, the faster the response of the time-sharing system, the nearer they appear to have a real-time effect. Such a system does enable the user to continue operating the display even though his program is not in core.

The attentions are stored in the attention buffer maintained by the time-sharing system in a "first-in-first-out" list called the attention list. When an attention is added to the list, a character is displayed on the screen at the position of the light pen. The addition of the character to the display reassures the user that his attention has been recorded even though the associated action must wait the next time his program is run by the time-sharing system.

It is up to the user to set the size of the attention buffer. This should be done immediately after signing on by making a call to procedure SATBUF. A fair balance with present time-sharing characteristics seems to be 100 registers for the attention buffer.

When the user's program cycles into active status, each time a call to procedure ATTN is made, the top item on the attention list is read and the corresponding character removed from the screen. ATTN interprets the data from the attention list, and stores it into the attention array specified as an argument to ATTN, where it is available to the user. No attention information is available to the user unless he calls ATTN, even though some exists in the buffer. If the user's program is in fact running when an attention arrives, it is not interrupted.

The contents of the attention array indicate the origin of the attention, e.g., a light pen see occurred or button 7 was pushed. For a light pen see, also available are the x and y position of the beam and the display pointer name of the object seen, i.e., the pointer to the display register. Since this display register is a part of the user's bead, he has direct access to his information about the mode or element seen.

VI. AED-0 LANGUAGE FEATURES

In order to study the sample program closely, one must be familiar with some of the features of the AED-0 language in which it is written. AED-0 is an extension of ALGOL and includes many features especially suited for building large systems and for manipulating complex data structures. A knowledge of ALGOL sufficient to understand this problem can be obtained from any ALGOL primer, such as the one written by McCracken, (D. McCracken, A Guide to ALGOL Programming, Wiley).

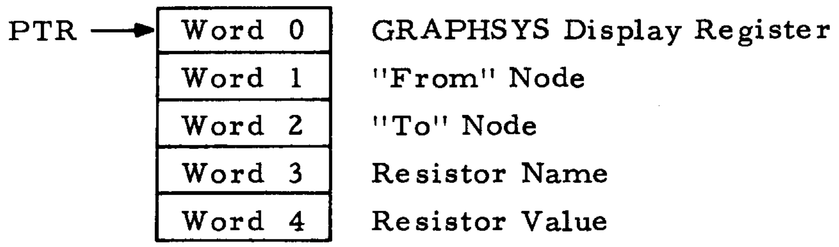
The most important AED-0 features used in this program which are not included in ALGOL are the free storage system, pointer variables, and components. In our previous discussion we have used the term "bead" to describe a block of storage used for some particular purpose, such as to represent all the information about a resistor; the term "bead" is used rather than "array" because a "bead" is not dimensioned at the time the program is compiled.

To create a new bead, the system procedure FREE is called as follows:

PTR = FREE (N) $,

The result of executing this statement will be that the pointer variable PTR will point to a fresh block of contiguous storage N-words long. The pointer in AED-0 on the 7094 has as its value the absolute core location of the first word of the block of storage allocated by FREE. 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 individually refer to the name and value of the resistor and also be each of the two connections to the nodes.

The ability to name items within a bead is provided by the AED-0 "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 FROM, INTO, NAME and VAL for words 1-4 of the bead. These names would then be put in the declaration statements

```
INTEGER COMPONENT NAME $,
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 beads representing resistors. The pointer returned by the call to FREE uniquely identifies the bead in which a component is located; the AED-0 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 on the 7094 as "take the location given by the pointer P, add to it 4 and store 7.2 in that location."

Output in AED is done by the "Assemble Package" which provides a general facility for converting numbers and building lines of output. The calls to the procedures ASMDEC and ASMBCD from the package are used in the program to print decimal numbers and character information; CARET adds a carriage return. The procedure READIN prints a message on the console and accepts format-free input of numerical or character information.

The following figures provide the flow diagrams for the sample program; the listing of the program has been heavily commented in the hope that anyone seriously interested in the workings of the program will be able to study it. (See Figs. B.4 and B.5.)

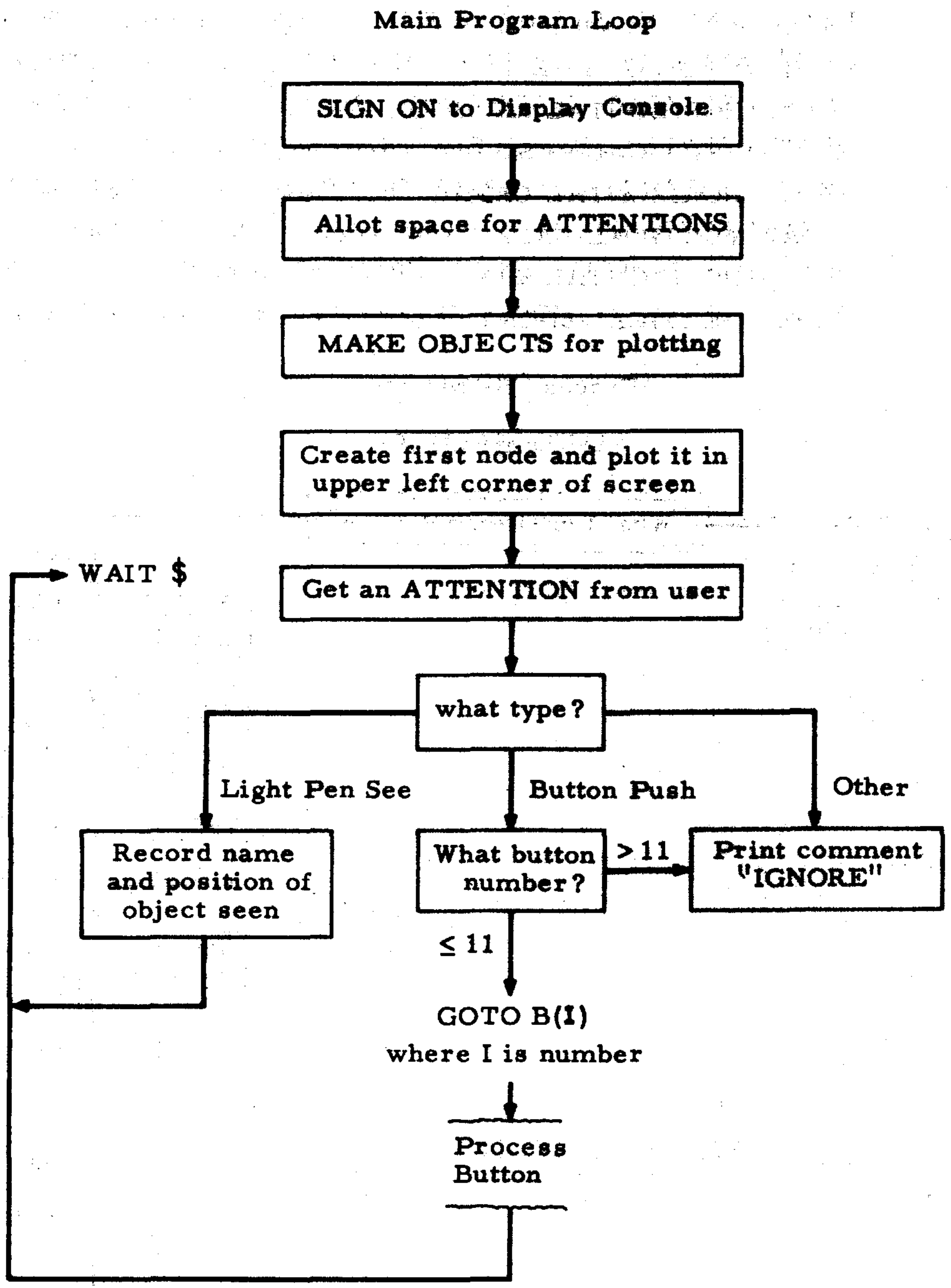

Main Program Loop
SIGN ON to Display Console
Allot space for ATTENTIONS
MAKE OBJECTS for plotting
Create first node and plot it in upper left corner of screen
Get an ATTENTION from user
WAIT $
what type?
Light Pen See
Button Push
Other
Record name and position of object seen
What button number?
> 11
Print comment "IGNORE"
≤ 11
GOTO B(I) where I is number
Process Button

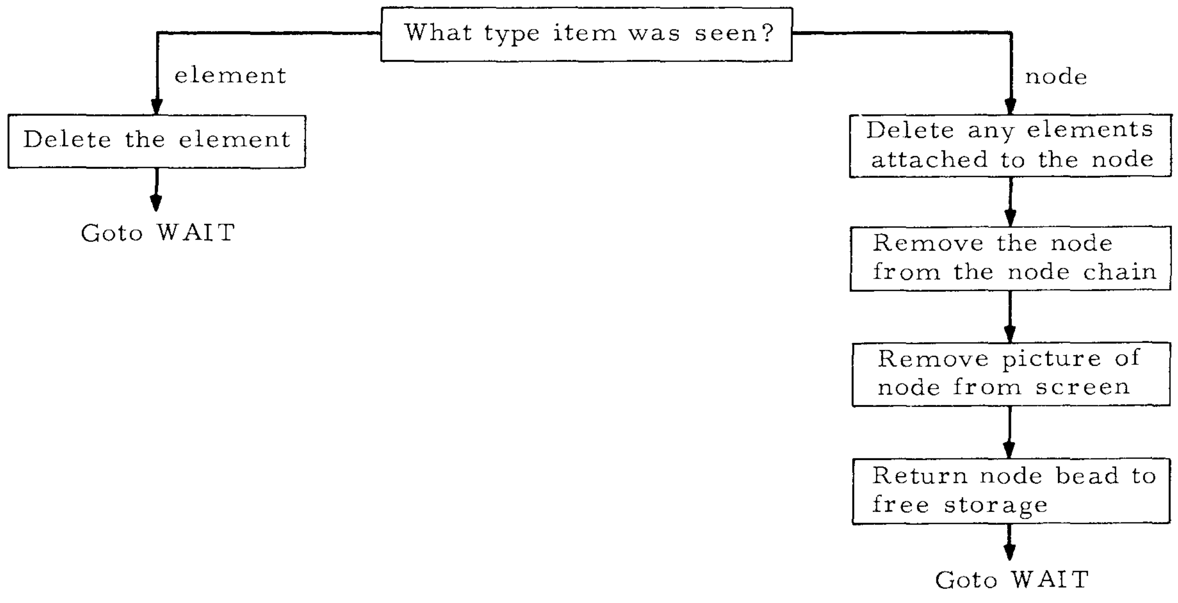
Fig. B.4 Flow Diagrams for the Sample Program

BUTTON	BUTTON FUNCTION
4	DELETE NODE OR ELEMENT LAST SEEN BY LIGHT PEN
5	GIVE VALUE OF ELEMENT LAST SEEN, ALLOW CHANGE
6	DRAW HORIZONTAL RESISTOR FROM NODE LAST SEEN
7	DRAW VERTICAL RESISTOR FROM NODE LAST SEEN
8	DRAW HORIZONTAL CAPACITOR FROM NODE LAST SEEN
9	DRAW VERTICAL CAPACITOR FROM NODE LAST SEEN
10	DRAW HORIZONTAL SHORT CIRCUIT FROM NODE LAST SEEN
11	DRAW VERTICAL SHORT CIRCUIT FROM NODE LAST SEEN

Process Buttons 4-5

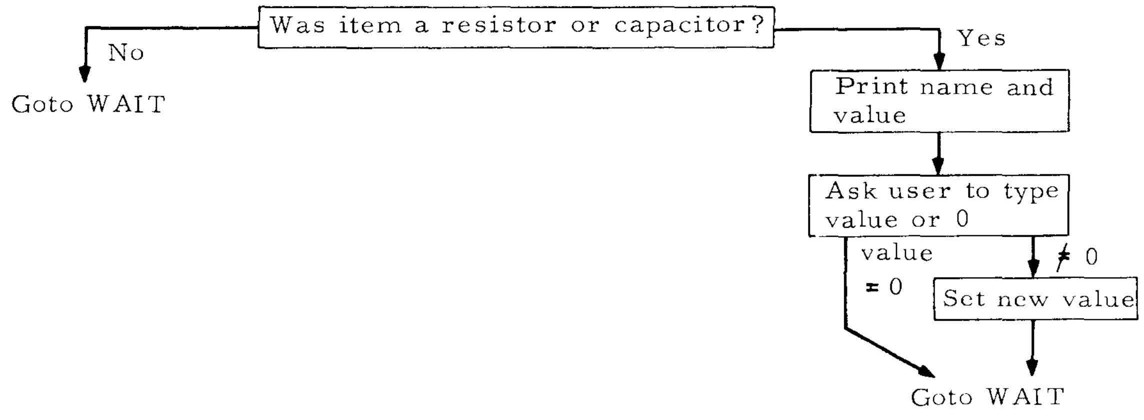
B4 $

Delete item seen

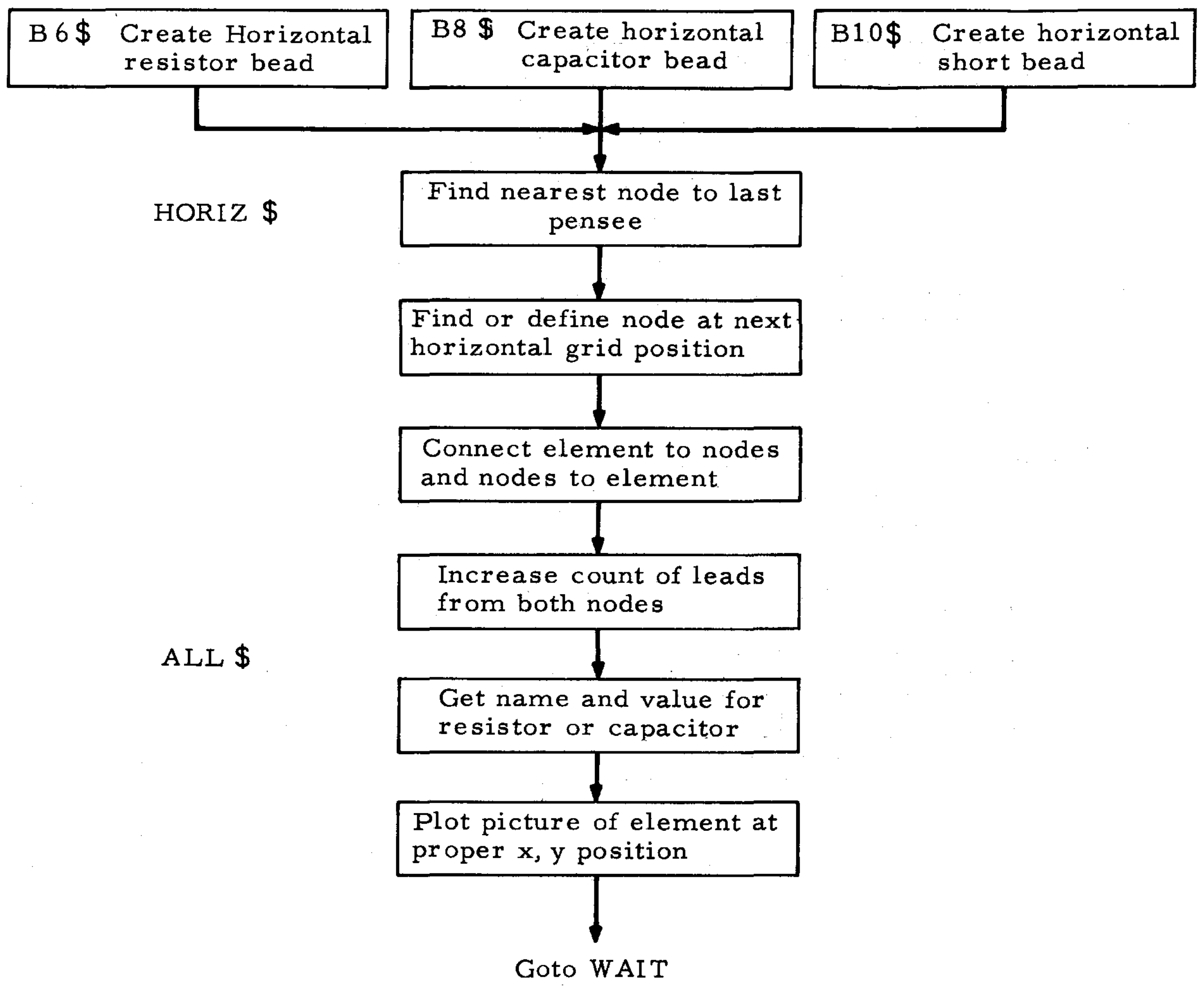

What type item was seen?
element
node
Delete the element
Goto WAIT
Delete any elements attached to the node
Remove the node from the node chain
Remove picture of node from screen
Return node bead to free storage
Goto WAIT

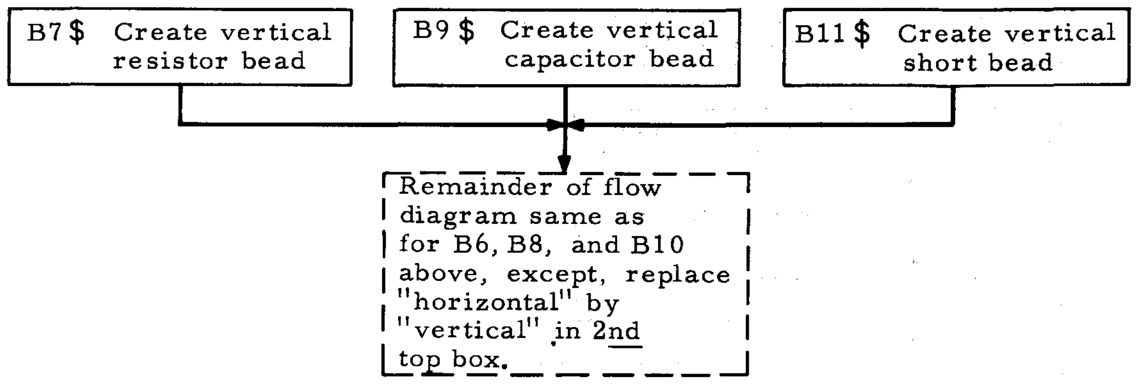
B5 $

Allow change of value for item seen



PROCESS BUTTONS 6-11

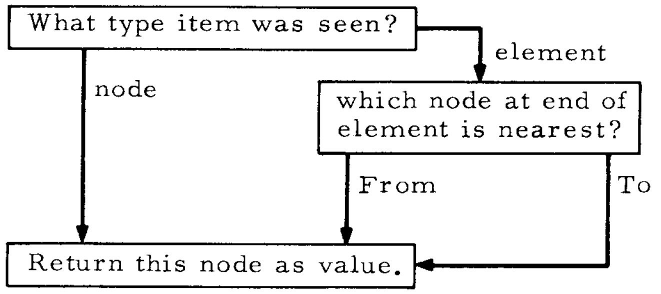

B6$ Create Horizontal resistor bead
B8 $ Create horizontal capacitor bead
B10$ Create horizontal short bead
HORIZ $
Find nearest node to last pensee
Find or define node at next horizontal grid position
Connect element to nodes and nodes to element
Increase count of leads from both nodes
ALL $
Get name and value for resistor or capacitor
Plot picture of element at proper x, y position
Goto WAIT


B7$ Create vertical resistor bead
B9 $ Create vertical capacitor bead
B11$ Create vertical short bead
Remainder of flow diagram same as for B6, B8, and B10 above, except, replace "horizontal" by "vertical" in 2nd top box.

Procedures

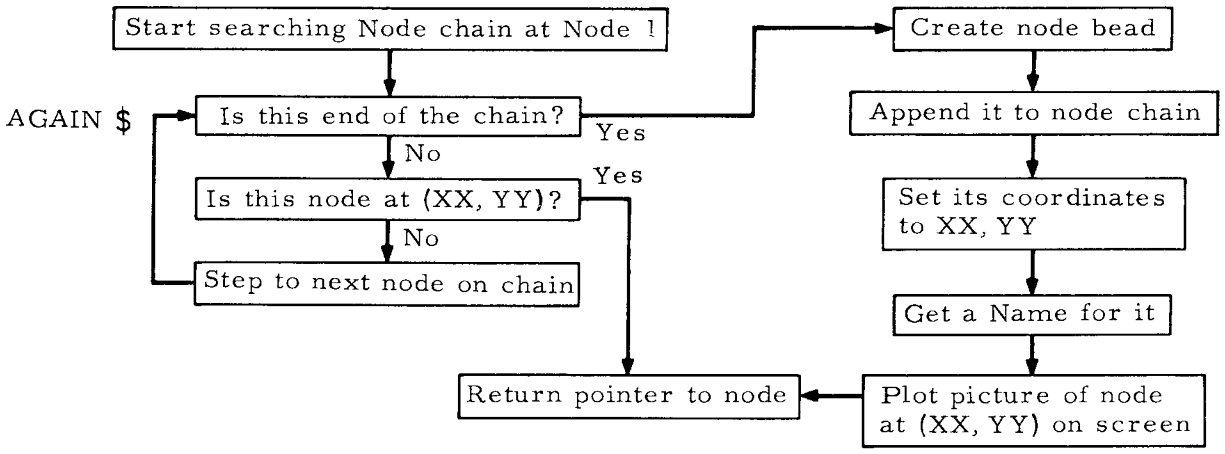
NEARNODE ( )

Find node nearest to item last seen.

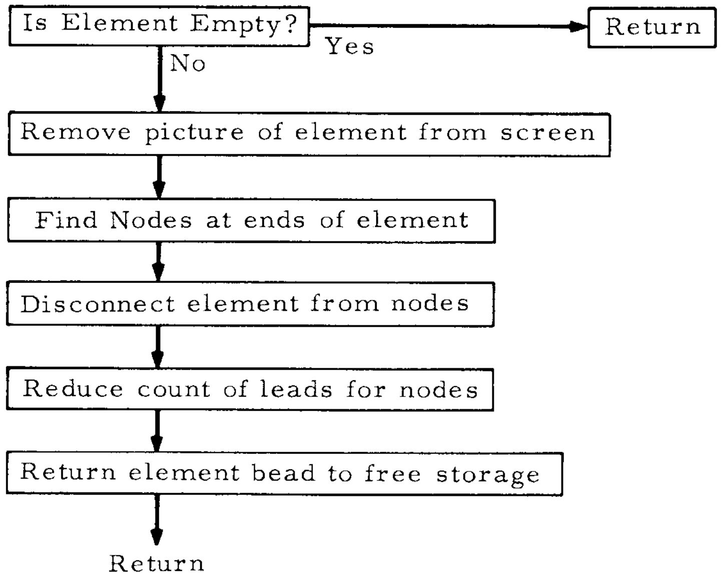

What type item was seen?
element
node
which node at end of element is nearest?
From
To
Return this node as value.

GETNODE(XX, YY)

Find or create Node bead at (XX, YY)


Start searching Node chain at Node 1
AGAIN $
Is this end of the chain?
Yes
No
Is this node at (XX, YY)?
Yes
No
Step to next node on chain
Create node bead
Append it to node chain
Set its coordinates to XX, YY
Get a Name for it
Plot picture of node at (XX, YY) on screen
Return pointer to node

DELETE (ELEMENT)


Is Element Empty?
Yes
Return
No
Remove picture of element from screen
Find Nodes at ends of element
Disconnect element from nodes
Reduce count of leads for nodes
Return element bead to free storage
Return

```
     BEGIN
COMMENT DECLARATIONS $,
     SYMONYMS INTEGER = POINTER $,

     INTEGER XSEEN,YSEEN,N,V,I $,
     INTEGER NODE,RESISTOR,CAPACITOR,SHORT $,
     INTEGER NODESZ,RESSZ,CAPSZ,SHTSZ,GRIDSP $,
     INTEGER ARRAY A(10) $,
     INTEGER ARRAY NOD(10),HR(12),VR(12),HC(12),VC(12),HS(4),VS(4) $,

     POINTER THIS,ITEM,OLD,FROMNODE,TONODE,PREV,NODE1,P,LASTSEE $,
     POINTER EMPTY,NODEPT,HRES,VRES,HCAP,VCAP,HSHT,VSHT $,

     POINTER PROCEDURE FREZ,PLOT,NEARNODE,GETNODE $,
     INTEGER PROCEDURE MALIGEC,MASEPOI,MASGLE $,
     PROCEDURE FRET $,
     PROCEDURE SGNON,SATBUF,ATTN,RMV $,
     PROCEDURE NAMENODE,NAMEELEM,DELETE,MAKEPICS $,
     PROCEDURE READIN,ASMBCD,ASMDEC,CARET $,

     INTEGER COMPONENT TYPE,NAME,X,Y,LEADS,VAL,WHOLE $,
     INTEGER COMPONENT SETPOS,CHAR1,CHAR2,CHAR3 $,
     POINTER COMPONENT U,L,R,D,NEXT,FROM,INTO $,
                                                   ... DEFINE COMPONENT POSITIONS IN BEADS //
     WHOLE $=$ 0 $,                                ...    DISPLAY REGISTER    DISPLAY REGISTER    DISPLAY REGISTER   //
     TYPE $=$ SETPOS $=$ 1 $,                      ...        TYPE = 0            TYPE = T            TYPE = 3       //
     LEADS $=$ FROM $=$ 2 $,                       ...         LEADS              FROMNODE            FROMNODE       //
     NEXT $=$ INTO $=$ 3 $,                        ...        NEXTNODE            INTONODE            INTONODE       //
     NAME $=$ 4 $,                                 ...          NAME                NAME                             //
     U $=$ VAL $=$ 5 $,                            ...       UP  ELEMENT            VALUE                            //
     L $=$ 6 $,                                    ...      LEFT ELEMENT                                             //
     R $=$ 7 $,                                    ...     RIGHT ELEMENT                                             //
     D $=$ 8 $,                                    ...      DOWN ELEMENT                                             //
     X $=$ 9 $,                                    ...            X                                                  //
     Y $=$ 10 $,                                   ...            Y                                                  //
                                                   ...       NODE  BEAD        T=1 RESISTOR BEAD  SHORT CIRCUIT BEAD //
                                                   ...                         T=2 CAPACITOR BEAD                    //
     CHAR1 $=$ 10 $,                               ... DEFINE POSITIONS FOR PACKING CHARACTERS INTO OBJECTS TO BE PLOTTED //
     CHAR2 $=$ 11 $,
     CHAR3 $=$ 12 $,

     PRESET                                        ... SET CONSTANT VALUES AT COMPILE TIME //
          BEGIN
          NODESZ = 11 $,                           ... PRESET BEAD SIZES //
          RESSZ = CAPSZ = 6 $,
          SHTSZ = 4 $,
          NODE = 0 $,                              ... PRESET TYPE NUMBERS //
          RESISTOR = 1 $,
          CAPACITOR = 2 $,
          SHORT = 3 $,
          GRIDSP = 200 $,                          ... GRID SPACING FOR DISPLAY //
          NODE1 = 0 $,                             ... FIRST NODE NOT YET DEFINED //
          EMPTY = 0 $,                             ... END OF STRING OR NULL POINTER //
          END $,

     SWITCH BUTTON = B,B,B,B4,B5,B6,B7,B8,B9,B10,B11 $, ... ARRAY OF STATEMENT LABELS //
     COMMENT   MAIN PART OF PROGRAM $,

START $   SGNON(1) $,                                   ... CONNECT TO DISPLAY UNIT //
          SATBUF(100) $,                                ... SET UP ATTENTION BUFFER //
          MAKEPICS() $,                                 ... CREATE OBJECTS FOR PLOTTING //
          NODE1 = GETNODE(-400,400) $,                  ... SET INITIAL NODE AT X=-400,Y=400 //

WAIT $    ATTN(A) $,                                    ... WAIT FOR AN ATTENTION TO BE READ INTO ARRAY A //
          IF A(1) EQL 38 THEN GOTO PENSEE $,            ... ATTENTION CODE 38 IS A PENSEE //
          IF A(1) LEQ 11 THEN GOTO BUTTON(A(1)) $,      ... ATTENTION CODE IS BUTTON 1-11 //
B $       ASMBCD(0,.BCD. /IGNORE/) $,                   ... IGNORE ALL OTHER ATTENTIONS //
          CARET() $,                                    ... PRINT COMMENT AND CARRIAGE RETURN //
          GOTO WAIT $,

PENSEE $  LASTSEE = A(4) $,                             ... 'NAME' OF OBJECT SEEN //
          XSEEN = A(2) $,                               ... PEN POSITION AT PENSEE //
          YSEEN = A(3) $,
          GOTO WAIT $,

B4 $                                                    ... DELETE NODE OR ELEMENT //
          IF TYPE(LASTSEE) NEQ NODE
          THEN BEGIN
               DELETE(LASTSEE) $,                       ... DELETE ELEMENT POINTED TO BY PEN //
               GOTO WAIT $,
               END $,
          IF LEADS(LASTSEE) NEQ 0                       ... IF ANY ELEMENTS ATTACHED TO NODE, DELETE THEM //
          THEN BEGIN
               DELETE(U(LASTSEE)) $,                    ... DELETE CHECKS TO SEE IF POINTER IS EMPTY BEFORE DELETING //
               DELETE(D(LASTSEE)) $,
               DELETE(L(LASTSEE)) $,
               DELETE(R(LASTSEE)) $,
               END $,
          IF LASTSEE EQL NODE1 THEN GOTO WAIT $,        ... DON'T DELETE NODE1 //
          P = NODE1 $,
HERE $    PREV = P $,                                   ... FIND PREVIOUS NODE BY SEARCHING NODE STRING //
          P = NEXT(PREV) $,                             ... STEP ONE ELEMENT ALONG STRING //
          IF P NEQ LASTSEE THEN GOTO HERE $,            ... IS THIS THE NODE SEEN //
          NEXT(PREV) = NEXT(P) $,                       ... YES, REMOVE THIS NODE FROM NODE STRING //
          RMV(P) $,                                     ... REMOVE PICTURE OF THIS NODE FROM DISPLAY //
          FRET(NODESZ,P) $,                             ... RETURN NODE BEAD TO FREE STORAGE //
          GOTO WAIT $,

B5 $                                                    ... GIVE VALUE, ALLOW CHANGE //
          IF TYPE(LASTSEE) EQL RESISTOR OR TYPE(LASTSEE) EQL CAPACITOR
          THEN BEGIN                                    ... IF ITEM SEEN WAS RESISTOR OR CAPACITOR //
               ASMBCD(3,NAME(LASTSEE),.BCD. / = /) $,   ... PRINT ITS NAME //
               ASMDEC(0,VAL(LASTSEE)) $,                ... AND VALUE //
               CARET() $,
               READIN(.BCI. /TYPE NEW VALUE OR 0/,V) $, ... PRINT MESSAGE, ACCEPT NEW VALUE //
               IF V NEQ 0 THEN VAL(LASTSEE) = V $,      ... STORE NEW VALUE IN ELEMENT BEAD //
               END $,
          GOTO WAIT $,
B6 $                                                         ... HORIZONTAL RESISTOR //
          THIS = FREZ(RESSZ) $,                              ... GET RESISTOR BEAD FROM FREE STORAGE //
          TYPE(THIS) = RESISTOR $,                           ... AND SET TYPE COMPONENT //
          ITEM = HRES $,                                     ... ITEM TO BE PLOTTED IS PICTURE OF HORIZONTAL RESISTOR //
HORIZ $                                                      ... PROCESS HORIZONTAL ELEMENTS //
          FROM(THIS) = FROMNODE = NEARNODE() $,              ... GET POINTER TO NEAREST NODE TO PENSEE //
          INTO(THIS) = TONODE = GETNODE(X(FROMNODE)+GRIDSP,Y(FROMNODE)) $, ... FIND OR CREATE NODE AT END OF ELEMENT //
          IF (OLD = R(FROMNODE)) NEQ EMPTY                   ... GET POINTER TO ELEMENT TO RIGHT OF FROMNODE //
          THEN DELETE(OLD) $,                                ... DELETE OLD ELEMENT IF ONE IS THERE //
          R(FROMNODE) = THIS $,                              ... MAKE NODE BEADS POINT TO NEW ELEMENT //
          L(TONODE) = THIS $,
ALL $                                                        ... COME HERE FOR ALL ELEMENTS //
          LEADS(FROMNODE) = LEADS(FROMNODE)+1 $,             ... INCREASE ELEMENT COUNT IN NODES //
          LEADS(TONODE) = LEADS(TONODE)+1 $,
          IF TYPE(THIS) NEQ SHORT                            ... IF THIS IS RESISTOR OR CAPACITOR //
          THEN NAMEELEM(ITEM,THIS) $,                        ... GET NAME AND VALUE //
          SETPOS(ITEM) = MASEPOI(X(FROMNODE),Y(FROMNODE),1) $, ... STORE SET POINT IN DISPLAY COMMANDS //
          PLOT(ITEM, THIS) $,                                ... PLOT PICTURE, 'NAME' IS THIS. NOTE THAT DISPLAY 'NAME' IS NOT
                                                               USER'S NAME,BUT A POINTER TO THE BEAD DESCRIBING THE ELEMENT //
          GOTO WAIT $,

B7 $                                                         ... VERTICAL RESISTOR //
          TYPE(THIS = FREZ(RESSZ)) = RESISTOR $,
          ITEM = VRES $,
VERT $                                                       ... PROCESS VERTICAL ELEMENTS //
          FROM(THIS) = FROMNODE = NEARNODE() $,
          INTO(THIS) = TONODE = GETNODE(X(FROMNODE),Y(FROMNODE)-GRIDSP) $,
          IF (OLD = D(FROMNODE)) NEQ EMPTY THEN DELETE(OLD) $,
          D(FROMNODE) = THIS $,
          U(TONODE) = THIS $,
          GOTO ALL $,

B8 $                                                         ... HORIZONTAL CAPACITOR //
          TYPE(THIS = FREZ(CAPSZ)) = CAPACITOR $,
          ITEM = HCAP $,
          GOTO HORIZ $,

B9 $                                                         ... VERTICAL CAPACITOR //
          TYPE(THIS = FREZ(CAPSZ)) = CAPACITOR $,
          ITEM = VCAP $,
          GOTO VERT $,

B10 $                                                        ... HORIZONTAL SHORT //
          TYPE(THIS = FREZ(SHTSZ)) = SHORT $,
          ITEM = HSHT $,
          GOTO HORIZ $,

B11 $                                                        ... VERTICAL SHORT //
          TYPE(THIS = FREZ(SHTSZ)) = SHORT $,
          ITEM = VSHT $,
          GOTO VERT $,
      COMMENT PROCEDURE DEFINITIONS $,


           DEFINE POINTER PROCEDURE NEARNODE TOBE
                BEGIN
      COMMENT NEARNODE GIVES A POINTER TO THE NODE NEAREST TO (XSEEN,YSEEN) $,
                IF TYPE(LASTSEE) EQL NODE
                THEN NEARNODE = LASTSEE                  ... ITEM SEEN WAS A NODE //
                ELSE BEGIN                               ... ITEM SEEN WAS ELEMENT //
                     FROMNODE = FROM(LASTSEE) $,
                     TONODE = INTO(LASTSEE) $,
                     IF (XSEEN-X(FROMNODE)+Y(FROMNODE)-YSEEN) LES (X(TONODE)-XSEEN+YSEEN-Y(TONODE))
                     THEN NEARNODE = FROMNODE
                     ELSE NEARNODE = TONODE $,
                     END $,
                END $,


           DEFINE POINTER PROCEDURE GETNODE(XX,YY) WHERE INTEGER XX,YY TOBE
                BEGIN
      COMMENT GETNODE GIVES A POINTER TO NODE AT (XX,YY), CREATING NODE IF NECESSARY $,
                P = PREV = NODE1 $,                      ... START SEARCH AT NODE1 //
 AGAIN $        IF P EQL EMPTY THEN GOTO GET $,          ... IF END OF STRING IS REACHED, NODE IS NOT YET DEFINED //
                IF X(P) EQL XX AND Y(P) EQL YY
                THEN BEGIN                               ... NODE ALREADY EXISTS AT (XX,YY) //
                     GETNODE = P $,
                     GOTO RETURN $,                      ... RETURN FROM PROCEDURE //
                     END $,
                PREV = P $,                              ... REMEMBER THIS NODE //
                P = NEXT(PREV) $,                        ... MOVE ONE NODE ALONG STRING //
                GOTO AGAIN $,                            ... AND KEEP CHECKING //
 GET $          GETNODE = P = FREZ(NODESZ) $,            ... GET NODE BEAD //
                TYPE(P) = NODE $,                        ... SET TYPE COMPONENT //
                NEXT(PREV) = P $,                        ... ADD NODE INTO STRING OF ALL NODES //
                NEXT(P) = EMPTY $,                       ... THIS NODE ENDS THE STRING //
                X(P) = XX $,                             ... SET COORDINATES IN NODE BEAD //
                Y(P) = YY $,
                NAMENODE(P) $,                           ... GET A NAME FOR THE NODE //
                SETPOS(NODEPT) = MASEPOI(XX,YY,1) $,     ... FILL IN SETPOINT IN DISPLAY COMMANDS //
                PLOT(NODEPT,P) $,                        ... AND PLOT PICTURE //
                END $,


           DEFINE PROCEDURE NAMENODE(NPTR) WHERE POINTER NPTR TOBE
                BEGIN
      COMMENT NAMENODE GIVES A UNIQUE NAME TO EACH NODE AND ADDS THE NAME TO THE PICTURE OF THE NODE. NAME IS 'N'
              FOLLOWED BY TWO DIGIT INTEGER. $,
                INTEGER NUM,TENS,ONES $,
                PRESET NUM = 0 $,
                NUM = NUM+1 $,
                TENS = NUM/10 $,
                ONES = NUM-10*TENS $,
                NOD(9) = MASGLE(TENS) $,                 ... STORE NAME INTO PICTURE //
                NOD(10) = MASGLE(ONES) $,
                NAME(NPTR) = .BCD. /N00000/ .V. (TENS .LS. 24) .V. (ONES .LS. 18) $, ... PACK NAME INTO NODE BEAD //
                END $,
     DEFINE PROCEDURE NAMEELEM(IPTR,EPTR) WHERE POINTER IPTR,EPTR TOBE
          BEGIN
COMMENT NAMEELEM GETS A NAME AND A VALUE FOR AN ELEMENT, ADDS THESE TO ITS BEAD, AND ADDS THE NAME TO THE PICTURE. $,
          READIN(.BCI. /TYPE NAME,VALUE/,N,V) $,  ... PRINT COMMENT, READ NAME AND VALUE //
          NAME(EPTR) = N $,
          VAL(EPTR) = V $,
          CHAR1(IPTR) = MASGLE(N .RS. 30 .A. 77C) $, ... GENERATE DISPLAY COMMANDS TO PLOT USER NAME //
          CHAR2(IPTR) = MASGLE(N .RS. 24 .A. 77C) $,
          CHAR3(IPTR) = MASGLE(N .RS. 18 .A. 77C) $,
          END $,


     DEFINE PROCEDURE DELETE(ELEMENT) WHERE POINTER ELEMENT TOBE
          BEGIN
COMMENT DELETE DELETES THE ELEMENT GIVEN, UPDATING DATA STRUCTURE $,
          IF ELEMENT EQL EMPTY THEN GOTO RETURN $, ... IF POINTER IS EMPTY, DO NOTHING //
          RMV(ELEMENT) $,                         ... REMOVE PICTURE FROM DISPLAY FILE //
          FROMNODE = FROM(ELEMENT) $,
          TONODE = INTO(ELEMENT) $,
          I = IF TYPE(ELEMENT) EQL SHORT          ... SET PROPER BEAD SIZE //
              THEN SHTSZ ELSE RESSZ $,
          FRET(I,ELEMENT) $,                      ... RETURN BEAD TO FREE STORAGE //
          IF R(FROMNODE) EQL ELEMENT              ... UPDATE THE NODE BEADS //
          THEN BEGIN
               R(FROMNODE) = EMPTY $,             ... ELEMENT WENT RIGHT FROM NODE //
               L(TONODE) = EMPTY $,
               END
          ELSE BEGIN
               D(FROMNODE) = EMPTY $,             ... ELEMENT WENT DOWN FROM NODE //
               U(TONODE) = EMPTY $,
               END $,
          LEADS(FROMNODE) = LEADS(FROMNODE)-1 $,  ... REDUCE ELEMENT COUNT ON NODES //
          LEADS(TONODE) = LEADS(TONODE)-1 $,
          END $,


     DEFINE PROCEDURE MAKEPICS TOBE
          BEGIN
COMMENT MAKEPICS GENERATES THE DISPLAY CONSOLE COMMANDS FOR OBJECTS TO BE PLOTTED $,

          NODEPT = LOC NOD $,                     ... GET POINTER TO ARRAY NOD //
          NOD(0) = 10 $,                          ... 10 CONSOLE COMMANDS //
          NOD(1) = 0 $,                           ... SET POINT (X,Y) COMMAND WILL BE INSERTED HERE //
          NOD(2) = MALIGEC(5,5,1) $,              ... LINE GENERATE COMMANDS //
          NOD(3) = MALIGEC(0,-10) $,              ... NO THIRD ARGUMENT OF MALIGEC FOR VISIBLE LINES //
          NOD(4) = MALIGEC(-10,0) $,              ... THIRD ARGUMENT IS 1 FOR INVISIBLE LINE //
          NOD(5) = MALIGEC(0,10) $,
          NOD(6) = MALIGEC(10,0) $,
          NOD(7) = MALIGEC(-7,20,1) $,
          NOD(8) = MASGLE(.BCD. /00000N/) $,      ... DISPLAY CHARACTER 'N' //
          NOD(9) = NOD(10) = 0 $,                 ... TWO DIGIT NODE NUMBER WILL BE FILLED IN //
```

```
        HRES = LOC HR $,                              ... DEFINE HORIZONTAL RESISTOR //
        HR(0) = 12 $,                                 ... TWELVE CONSOLE COMMANDS //
        HR(1) = 0 $,                                  ... SET POINT WILL BE INSERTED //
        HR(2) = MALIGEC(60,0,2) $,                    ... THIRD ARGUMENT 2 FOR PEN INSENSITIVE //
        HR(3) = MALIGEC(10,10) $,
        HR(4) = MALIGEC(20,-20) $,
        HR(5) = MALIGEC(20,20) $,
        HR(6) = MALIGEC(20,-20) $,
        HR(7) = MALIGEC(10,10) $,
        HR(8) = MALIGEC(60,0,2) $,
        HR(9) = MALIGEC(-100,40,1) $,
        HR(10) = HR(11) = HR(12) = 0 $,               ... SPACE FOR 3 CHARACTER NAME //

        VRES = LOC VR $,                              ... DEFINE VERTICAL RESISTOR //
        VR(0) = 12 $,
        VR(2) = MALIGEC(0,-60,2) $,
        VR(3) = MALIGEC(10,-10) $,
        VR(4) = MALIGEC(-20,-20) $,
        VR(5) = MALIGEC(20,-20) $,
        VR(6) = MALIGEC(-20,-20) $,
        VR(7) = MALIGEC(10,-10) $,
        VR(8) = MALIGEC(0,-60,2) $,
        VR(9) = MALIGEC(40,100,1) $,

        HCAP = LOC HC $,                              ... HORIZONTAL CAPACITOR //
        HC(0) = 12 $,
        HC(2) = MALIGEC(90,0,2) $,
        HC(3) = MALIGEC(0,20,1) $,
        HC(4) = MALIGEC(0,-40) $,
        HC(5) = MALIGEC(20,0,1) $,
        HC(6) = MALIGEC(0,40) $,
        HC(7) = MALIGEC(0,-20,1) $,
        HC(8) = MALIGEC(90,0,2) $,
        HC(9) = MALIGEC(-100,40,1) $,

        VCAP = LOC VC $,                              ... VERTICAL CAPACITOR //
        VC(0) = 12 $,
        VC(2) = MALIGEC(0,-90,2) $,
        VC(3) = MALIGEC(20,0,1) $,
        VC(4) = MALIGEC(-40,0) $,
        VC(5) = MALIGEC(0,-20,1) $,
        VC(6) = MALIGEC(40,0) $,
        VC(7) = MALIGEC(-20,0,1) $,
        VC(8) = MALIGEC(0,-90,2) $,
        VC(9) = MALIGEC(40,100,1) $,

       HSHT = LOC HS $,                               ... HORIZONTAL SHORT //
       HS(0) = 4 $,
       HS(2) = MALIGEC(50,0,2) $,
       HS(3) = MALIGEC(100,0) $,
       HS(4) = MALIGEC(50,0,2) $,

       VSHT = LOC VS $,                               ... VERTICAL SHORT //
       VS(0) = 4 $,
       VS(2) = MALIGEC(0,-50,2) $,
       VS(3) = MALIGEC(0,-100) $,
       VS(4) = MALIGEC(0,-50,2) $,

       END $,
   END FINI
```

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Massachusetts Institute of Technology Electronic Systems Laboratory and Project MAC	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP None

3. REPORT TITLE

An Integrated Hardware-Software System for Computer Graphics in Time-Sharing

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Technical Report, September 1962 to March 1968

5. AUTHOR(S) *(Last name, first name, initial)*

Thornhill, Daniel E., R. H. Stotz, D. T. Ross, and J. E. Ward

6. REPORT DATE November 1968	7a. TOTAL NO. OF PAGES 168	7b. NO. OF REFS 10

8a. CONTRACT OR GRANT NO. Air Force Contract F33615-67-C-1530 and Office of Naval Research Contract Nonr-4102(01)	9a. ORIGINATOR'S REPORT NUMBER(S) ESL-R-356 MAC-TR-56 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. AVAILABILITY/LIMITATION NOTICES

Distribution of this document is unlimited

11. SUPPLEMENTARY NOTES None	12. SPONSORING MILITARY ACTIVITY Air Force Manufacturing Laboratory, RTD Wright Patterson AFB	12. SPONSORING MILITARY ACTIVITY Advanced Research Projects Agency 3D-200 Pentagon Washington, D.C. 20301

13. ABSTRACT This report describes the ESL Display Console and its associated user-oriented software systems developed by the M.I.T. Computer-Aided Design Project with Project MAC. Console facilities include hardware projection of three-dimensional line drawings, automatic light pen tracking, and a flexible set of knob, switch, and push-button inputs. The console is attached to the Project MAC IBM 7094 Compatible Time-Sharing System either directly or through a PDP-7 Computer. Programs of the Display Controller software provide the real-time actions essential to running the display, and communication with the time-sharing supervisor. A companion graphics software system (GRAPHSYS) provides a convenient, high-level, and nearly display-independent interface between the user and the Display Controller. GRAPHSYS procedures allow the user to work with element "picture parts" as well as "subpictures" to which "names" are assigned for identification between user and Controller programs. Software is written mostly in the machine-independent AED-0 Language of the Project and many of the techniques described are applicable in other contexts.

14. KEY WORDS

AED software	Computer-aided design	On-line computers
Computer graphics	Machine-aided cognition	Time-sharing
Display software systems	Multiple-access computers	Time-shared computers

DD FORM 1 NOV 65 1473 (M.I.T.)

UNCLASSIFIED
Security Classification