EVANS & SUTHERLAND COMPUTER CORP.

LINE DRAWING SYSTEM MODEL 1

SYSTEM REFERENCE MANUAL

January 1, 1970

Evans & Sutherland Computer Corporation
Three Research Road
Salt Lake City, Utah 84112

EVANS & SUTHERLAND

LINE DRAWING SYSTEM MODEL 1

SYSTEM REFERENCE MANUAL

Table of Contents

FOREWORD

The design of the Evans & Sutherland Line Drawing System's display processor was developed in the autumn of 1968 by Charles L. Seitz, who was then at the Massachusetts Institute of Technology. The design is based on philosophies and concepts developed from programmer and user experience on large interactive display systems over the last several years. This manual describes the total system, but careful reading makes it fully applicable to reduced systems.

The associates of Evans & Sutherland express their gratitude to Dr. Seitz for his elegant design and careful preparation on the original manual on which the following description is based.

Significant contributions to the design of the LDS-1 display processor were made by the computer research group of Bolt, Beranek and Newman, Inc., Cambridge, Massachusetts, on whose computer the first LDS-1 system was installed in August 1969. The Bolt, Beranek and Newman group's foresight and tenacious inquiry into the subtle aspects of the system were a substantial help during critical early phases of the design. We owe a particular debt of gratitude to Servero Ornstein of BB&N for his critical reading of the original manual.

I. THE DISPLAY SYSTEM

CAPABILITIES

The DISPLAY PROCESSOR described here is part of a new, high-performance display system applicable to a variety of tasks of current importance in advanced research and engineering. The complete system not only provides for high-speed display of lines so that it can present more complex pictures than usual, but also provides high-speed graphic arithmetic processing. The available graphic arithmetic units are separately timed computational elements that are driven from the display processor. One arithmetic unit, the CLIPPING DIVIDER, enables the system to choose dynamically the portions of the material for presentation, choose the scale of presentation, and choose the position of presentation. The second arithmetic unit, the MATRIX MULTIPLIER, enables the system to translate, rotate, and scale two and three-dimensional objects dynamically as well as to plot three-dimensional curved surfaces automatically.

The pictorial information processed by the CLIPPING DIVIDER and the MATRIX MULTIPLIER may be plotted automatically on a LINE DRAWING SCOPE, or may be written back into main memory for output onto plotters, film writers, or remote displays. The display processor also includes provision for a hardware CHARACTER GENERATOR.

There are several classes of applications for which this display system provides unique capability. The display system can generate perspective (including isometric and orthographic) views of three-dimensional wire frame objects. The system can be used for displaying perspective views of three-dimensional material such as a landscape sketch for an airplane simulation, or a perspective view of a three-dimensional mathematical curve. The display not only computes the perspective view, but also automatically omits any lines or parts of lines behind the observer or outside the field of view defined by the scope. The system can be used for displaying complex two-dimensional pictures such as circuit diagrams, representations of list structures, printed circuit layouts, symbolic mathematical expressions, and so on. The display's capability to expand such pictures, and the inherently high digital resolution provided, make the system uniquely capable of presenting complex pictures. The user can define and display pictures in which the effective drawing area is over an acre, and still 'zoom' in to examine a ten-inch square area without loss of resolution.

DATA FORMATS

The display is refreshed dynamically from information stored in the memory of the associated general purpose computer. The display processor interfaces to the memory of the parent computer and operates as an autonomous processor which can be started and stopped by main processor I/O control. The coordinate information in memory is purely numerical, and thus may be manipulated easily by the central processor. The information in memory describes the picture in terms of:

(1) end point coordinates for line segments, or coordinates for 'dots'. Coordinate information may be either absolute, or relative to a remembered point called the CURRENT POINT. For drawing lines the current point is used to define one end of the line segment. Several formats for displaying tables of coordinate data are provided.

(2) pairs of address pointers in the right and left halves of 'full' words, pointing to coordinate information as in (1). In this format the two pointers may be used to point to the two end points of a line, and an end point may be used for as many line segments as desired. If the coordinate value is changed by the central processor, all the lines 'attached' to the coordinate will move. Thus it is possible to represent the 'topology' of the picture separate from its 'geometry'.

(3) multiple references to separate definitions of substructures. An object appearing in several places in a picture need be defined only once and can be 'called as a subroutine', even if it appears in different sizes.

All coordinates are described as 'half word' two's complement numbers, stored in either the right or left half of a full word. This description will concentrate on 18 bit half words, and 36 bit full words in order to explicitly describe the various word formats. Two dimensional information is stored with the X coordinate in the left half and the Y coordinate in the right half:

0 17	18 35
X	Y

Three dimensional information is stored in two consecutive full words with X and Y in the first word and two Z values in the second word.

0 17	18 35
X	Y
Z_x	Z_y

Usually Z_x and Z_y will be the same; making them different provides for a different perspective divisor in X and Y, a capability which is useful in the display of curves. Using a full two words for three dimensional point information simplifies both the hardware and programming of the display. Since many control processors have a half word instruction repertoire, the use of the half word makes it fairly easy to manipulate data in this format.

Coordinates for this display system are stored as half word numbers, rather than the customary 10 or 12 bits, for the following reason. Because the display itself can magnify two dimensional pictures, use of the full half word provides good resolution even at high magnifications, e.g. up to 256 for 18 bit half words. In three dimensional perspective views the high resolution is necessary for viewing objects that are close to the observer.

THE CLIPPING DIVIDER AND SCOPE

Clipping in two and three dimensions

We call the space in which a DRAWING can be defined the PAGE, and coordinates in it PAGE COORDINATES. The page coordinate system is centered about zero so that for two dimensions (0,0) is the middle of the page, for an 18 bit 2's complement space expressed in octal (400000, 400000) is the lower left corner, and (377777, 377777) is the upper right corner. In two dimensions, the portion of the drawing which is to appear is selected by specifying a WINDOW. The window is a rectangular portion of the page aligned with the X and Y axes. The window is defined by the X coordinates of its left and right edges, and the Y coordinates of its bottom and top, all in page coordinates. These values are held in the WINDOW registers of the clipping divider. The clipping divider has the ability to filter material for display through this window, transmitting only that part of the material which lies within the window.

Whatever appears through the window is mapped onto a rectangular region of the scope, called the VIEWPORT. The VIEWPORT is defined for both two and three dimensional pictures by its left, right, bottom, and top edges in SCOPE COORDINATES. These values are stored in the VIEWPORT registers of the clipping divider. Coordinates for the scope are a full half word length, identical to page coordinates, but because only the 12 least significant bits are used to drive the scope, coordinates for the scope must lie between 774000 and 003777. (Larger viewports may be used to create wraparound, if desired, and smaller viewports to paint the picture on less than the full area of the scope.) Thus if the WINDOW is specified to be as large as possible

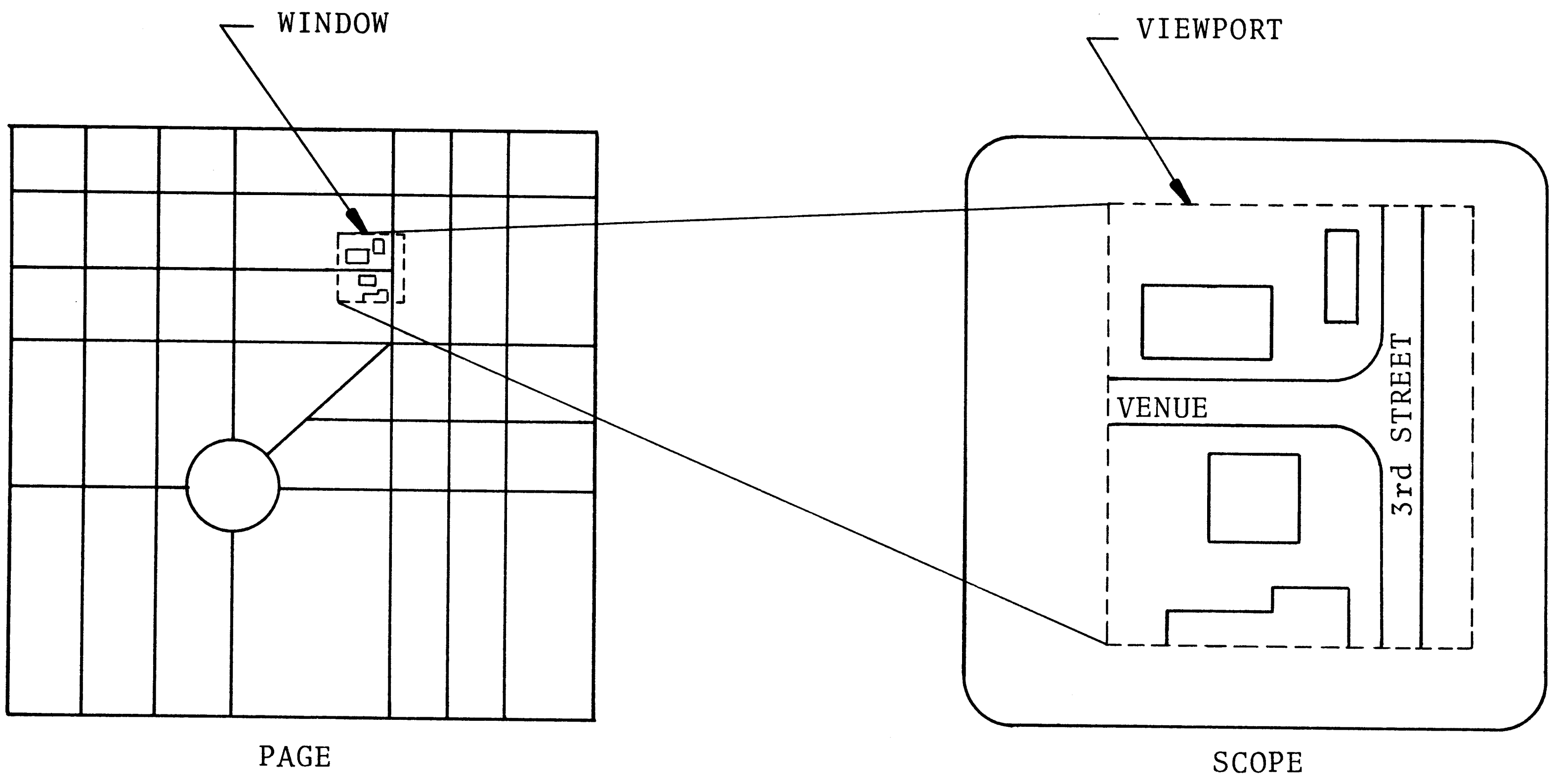


Figure 1-1 Two-dimensional operation of the Clipping Divider

and the VIEWPORT is specified to run from 774000 and 003777, the entire PAGE will be plotted exactly to fit the SCOPE. The clipping divider performs a linear mapping of everything within the window onto the viewport. If the window is geometrically similar to the viewport, the picture produced on the scope is geometrically similar to the drawing on the page (i.e. angles are preserved). If the window and viewport are not geometrically similar, the drawing is magnified by a different amount in X and Y. If the 'right' edge of the viewport is placed to the left of the 'left' edge, the picture is reflected about a vertical axis. Similar reflection about a horizontal axis or about both axes is possible. If the right and left or top and bottom edges of the window are 'backward', nothing appears through the window. The two dimensional operation of the clipping divider is illustrated in figure I.1.

Operation of the clipping divider in three dimensions is similar to its operation in two dimensions. The page coordinate system uses four half-word numbers $[X, Z_x, Y, Z_y]$ to specify a point. The contents of the window registers is not used in three dimensions. The 'window' is a pyramid whose vertex is at [0,0,0,0] and which extends along the positive Z axis including all the volume where $|X| \leq |Z_x|$ and $|Y| \leq |Z_y|$. All dots and lines within or passing through this volume are projected through the origin onto a plane normal to the Z axis by means of the perspective divisions: $X_{scope} = X/Z_x$, $Y_{scope} = Y/Z_y$, and are scaled so as to map onto the viewport. The resulting picture is a perspective view as seen from the origin looking in the posibive Z direction. Because the field of view is 90^o, the resulting perspective looks strange unless viewed from very close to the scope (5 inches for a 10 inch square viewport). In order to produce more reasonable looking perspectives, the Z values given to the clipping divider should be multiplied by the tangent of half the desired viewing angle.* This scaling can be accomplished easily by scaling the appropriate terms in the rotation and translation matrix which defines the point of view. This rotation and translation may be accomplished either by central processor multiplications or by using the matrix multiplier.

Frequently a drawing will include repetitions of a single object in several places on the drawing. When this occurs it is desireable to define this object or symbol only once, and in effect to call it as a subroutine. One mechanism for doing this kind of subroutining is built into the processor. So that the symbol may be positioned anywhere on the page, it must be described by relative drawing specifications, and it may appear in only one size.

A more general mechanism for multiple references to a symbol is provided in the clipping divider. This mechanism permits everything within a MASTER rectangle on one page to be mapped onto an INSTANCE rectangle defined on another page, in a way similar to the window-to-viewport mapping. Because this second page may be mapped similarly onto another page, or onto the scope by the window-to-viewport mapping, it is necessary to concatenate

*The perspective transformation is $X = (x/z)\cot(\alpha/2)$, where α is the viewing angle. Thus the x and y values could be scaled by the cotangent, or the z values by the tangent.

two of these transformations. This concatenation operation, called BOXing, is a computational function of the clipping divider, and does not itself result in the generation of any output to the scope or to memory. Figure I.2 illustrates the BOX operation, and the figure is helpful for following the text which follows.

The symbol or object to be mapped onto the PAGE is delineated by a MASTER rectangle on DEFINITION space. The left, right, bottom, and top of the MASTER are specified as the data for the BOX operation exactly as if one were drawing a line across a diagonal of the MASTER. There is nothing particularly special about the DEFINITION space; it is simply a page and even may be the same page onto which the object is to be mapped. Everything appearing within the MASTER is to be mapped onto a rectangular area called the INSTANCE. The INSTANCE is specified by loading the page coordinates of its left, right, bottom and top into the 'instance' registers of the clipping divider. The master-to-instance mapping has the same nature as the window-to-viewport mapping in that it is linear. The INSTANCE may be specified 'backward' to create mirror images, and a 'backward' MASTER results in none of the lines in the symbol being visible. Once the symbol is mapped onto the page, it must be mapped by the existing window-to-viewport mapping onto the scope. The BOX operation forms the composite transformation of these two linear transformations and leaves the results in the clipping divider window and viewport registers. The new window is a window on the DEFINITION space, and the new viewport is a viewport on the scope. The composite transformation behaves just as if the symbol were first mapped according to the master-to-instance mapping onto the page, and then mapped according to the existing window-to-viewport mapping onto the scope.

The clipping divider computes the composite transformation by operating on the area in common between the instance and the window. This common area is projected onto the scope by the window-to-viewport mapping and onto the definition space by the inverse master-to-instance mapping. It is possible that the instance and window had no area in common. The clipping divider 'AIC' (Area In Common) bit may be tested by the display processor conditional instruction after loading the clipping divider instance register. BOXing may be used in subroutines to any level, and its use is illustrated in the programming examples.

Two other operations are planned for the clipping divider. On a 'best effort' basis we are attempting to include these capabilities. They do not, however, form a part of the acceptance testing of the system.

(1) Curve mode. In this mode the perspective view computed is that portion of the picture where $|X| \leq |Z_x|$ and $|Y| \leq |Z_y|$;i.e., material with negative Z values (in the negative Z pyramid) will be visible. This facility is believed to be useful for displaying certain kinds of curves.

(2) Minimum Effort. In this mode the clipper merely computes the X, Y, and Z coordinates for some point which is visible on the specified line. Since the clipper endeavors to do the least work possible while still accomplishing this end, we call this mode 'minimum effort' (MEF). The clipping divider may also accumulate angle information about lines which may assist in discovering whether or not a 'polygon' surrounds the

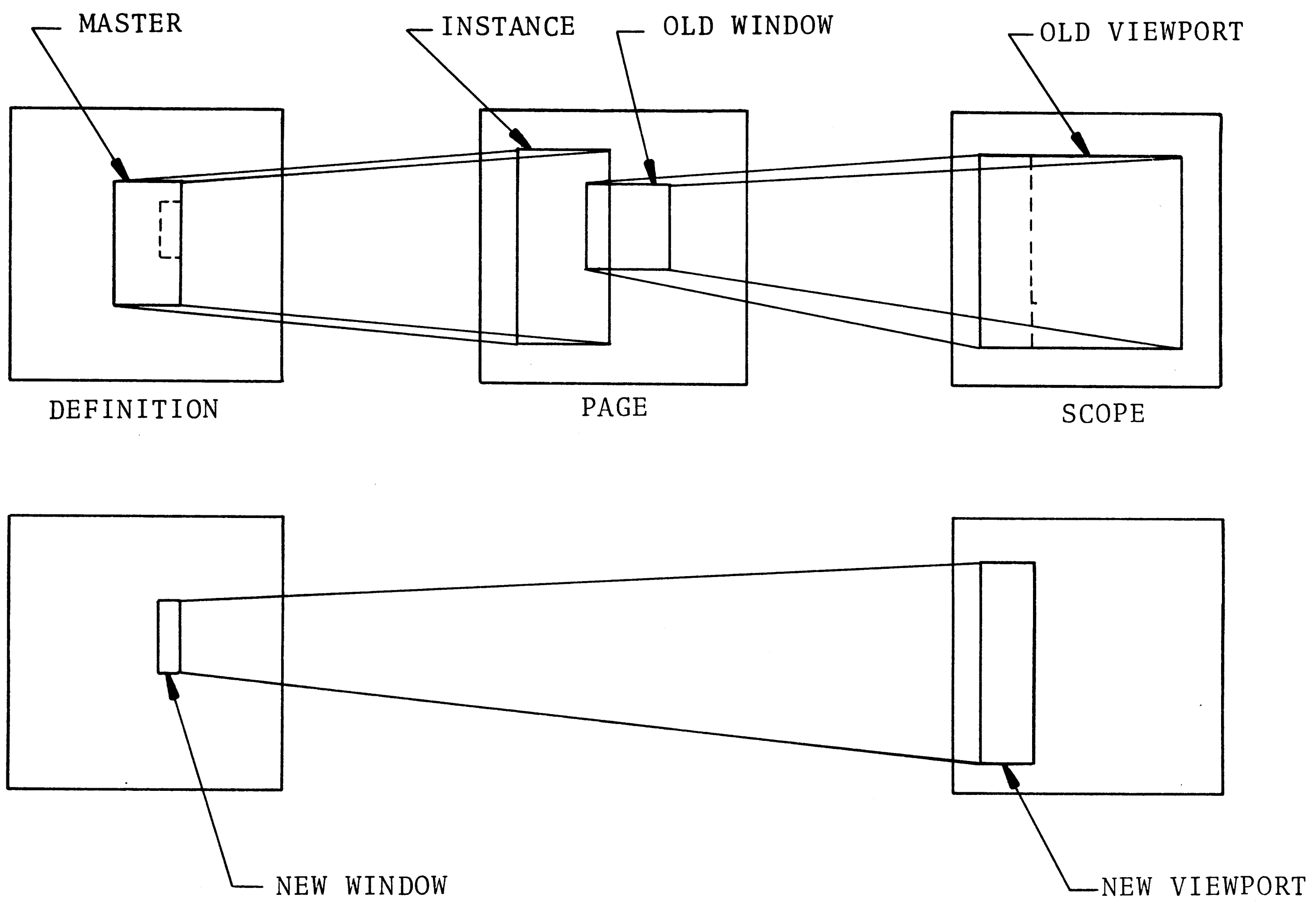


Figure 1-2 The BOX operation

current field of view. The clipping divider has 'hit count' and 'angle count' registers for this purpose (see Appendix). This capability is believed to be useful in the Warnock hidden line algorithm. Evans and Sutherland assumes no responsibility for the correct operation of the angle detection circuits nor for the central processor software required for hidden line computation.

The speed of the clipping divider is compatible with that of a fast line drawing scope, and is sufficient for providing considerable detail in the pictures presented. The specific display rate depends on the nature of the material being displayed, but the rate approaches two thousand lines flicker-free (i.e. a two thousand line picture can be processed by the display system--and the processed results displayed--thirty times a second). Because lines outside the window are rejected very quickly, the display rate does not suffer as badly as might be expected when the window covers only a small part of a drawing. The time required for processing data is explained in Appendix II.

Lines and Dots

The display system permits a drawing to be composed of dots (more conventionally called points) and lines. Such drawings are always defined in page coordinates. Any portion of the drawing may be plotted on any part of the scope by specifying a window and viewport (page to scope mapping). Alternatively any portion of the drawing may be superimposed on another drawing by specifying a definition and instance (page to page mapping), and then performing the BOX operation. This flexibility allows the user to define all drawings in page coordinates.

A line segment is represented by its end points, and a dot simply by its position on the page. The clipping divider remembers one point on the page, the CURRENT POINT, in the clipper 'save' registers. The current point exists in either two or three dimensions. The line segments processed by the clipping divider generally consist of the current point as one end and a NEW POINT as the other end. The clipping divider retains one end of each line that is drawn as the current point to be used in drawing the next line. For some lines which are drawn the clipping divider retains the new point as the current point for later use. Such lines can be thought of as being DRAWn TO the new point. For other lines the value of the current point is not changed. Such lines can be thought of as being DRAWn FROM the new point to the current point. The first alternative (DRAW TO) permits efficient specification of chain connected line segments while the second alternative (DRAW FROM) is useful for specifying star-like figures. The current point may be changed without displaying by a SET POINT operation. Displaying a dot causes the current point to change to the position of the dot. An illustration of these operations is shown in Figure I.3.

The current point also serves the function of permitting relative coordinate specifications. The new point may be specified absolutely in page coordinates, or as a page coordinate displacement RELATIVE to the

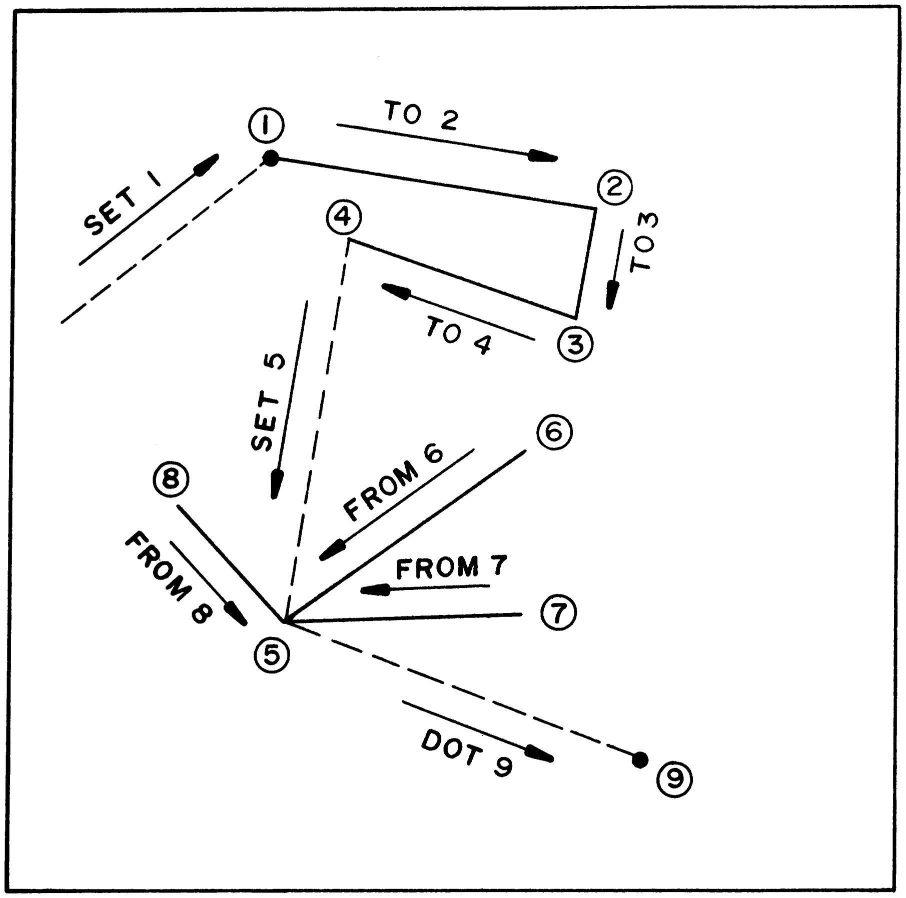


Figure 1.3: SET, DRAW TO, DRAW FROM, and DOT operations

current point. All of the basic drawing operations: SET POINT, DRAW TO, DRAW FROM, and DOT, may be specified relative to the current point. The current point may also be used to specify the midpoint of the line to be specified. In the SIZE RELATIVE specification, the clipping divider uses the new data to specify the half-length of a line segment in X and Y, and uses the current point as the midpoint of the line. The current point is not changed by a SIZE RELATIVE draw. SIZE ABSOLUTE operates in an analogous manner, however taking (0,0) to be the current point.

The clipping divider has a special operating mode which is useful for plotting simple graphs. For these graph modes the clipping divider ignores either the X or Y part of the drawing data sent to it, and uses instead a relative displacement held in the instance registers.

Loading the Clipper Registers.

There are four numerical registers in the clipping divider, called SAVE, VIEWPORT, WINDOW, and INSTANCE, each of which stores a four-component vector. Each component of the vector is a 20-bit signed two's complement number. The two half words map into the least significant of the 20 bits with leftward sign extension. There are in addition several registers for storing some miscellaneous items: HIT COUNT, ANGLE COUNT, SCOPE SELECT, INTENSITY, and NAME. These registers are in a configuration as shown in Figure I.4, and may be addressed as indicated.

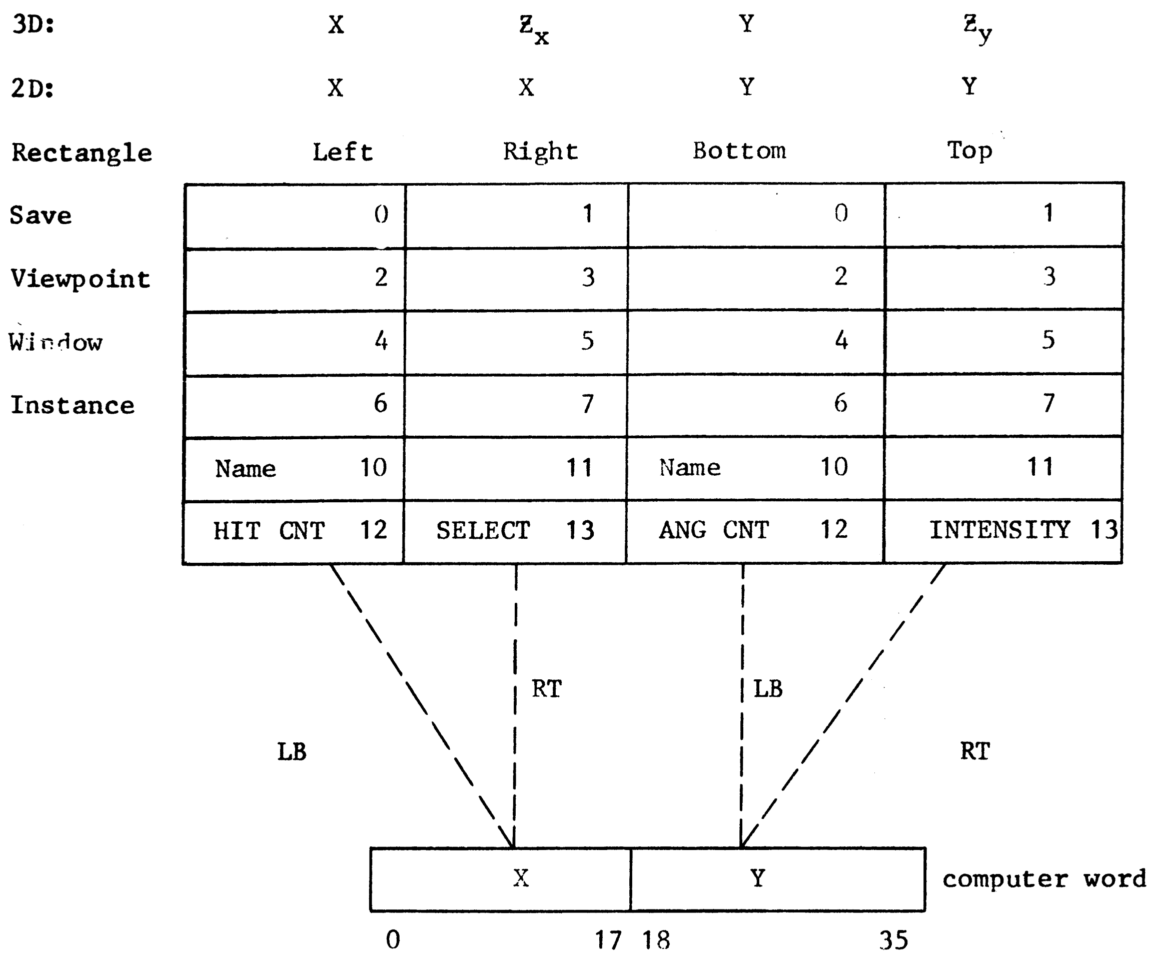
In the VIEWPORT, WINDOW, and INSTANCE registers the four components may refer to the left, right, bottom, and top of a rectangular area, $[X_L, X_R, Y_B, Y_T]$. The SAVE register specifies a single point in either two or three dimensions. In two dimensions the SAVE register contains duplicate information: [X, X, Y, Y]. In three dimensions the SAVE register contains the 3D current point: $[X, Z_x, Y, Z_y]$. We are interested here only in the two dimensional use. The SAVE, VIEWPORT, WINDOW, and INSTANCE registers can be addressed in two ways:

(1) As two component halves (addresses 0-7)
(2) As full four component vectors (addresses 14-17)

Furthermore, the same four mechanisms for specifying lines (Absolute, Relative, Size Absolute, Size Relative) are available as modes for loading the registers.

The two component halves of the SAVE, VIEWPORT, WINDOW, and INSTANCE are called SAVELB, SAVERT, VIEWLB, VIEWRT, WINDLB, WINDRT, INSTLB, INSTRT, where the LB suffix indicates 'left and bottom' and the RT suffix indicates 'right and top'. Thus a specification of LOCLA WINDLB, $[X_1, Y_1]$ (1) (i.e. a 2-component load) will put X_1 and Y_1 into the left and bottom components of the WINDOW register. (See chapter IV for mnemonic meanings.) A two-component relative load, e.g. LOCLR WINDLB, $[dX_1, dY_1]$ (1) will load the left and bottom components of the WINDOW register with information derived by adding dX_1, dY_1 to the left and bottom components of the SAVE register (not the WINDOW register).

If the full register is addressed (i.e. addresses 14-17) then a four-component load is indicated. A full four-component load of a register (even the SAVE register) specifies that the new data and the data in the SAVE register are combined just as they would be to draw a line. The two ends of this virtual (invisible) line supply the four components required. Let us suppose, for example, that the SAVE register contains [X_1, X_1, Y_1, Y_1] due to a SETPT [X_1, Y_1]. A LOCLR WIND, [dX, dY] would generate a new end-point X_1 + dX, Y_1 + dY just as would be generated to draw a line. The numbers placed in the WINDOW register then would be: [X_1, X_1 + dX, Y_1, Y_1 + dY]. Thus a full window load can be thought of as drawing a virtual (invisible) line from the lower left to the upper right of the rectangle desired. Similarly a LOCLSR WIND, [dX, dY] would place vector [X_1 - dX, X_1 + dX, Y_1 - dY, Y_1 + dY] into the WINDOW.



Addressing:

Two component:	0	SAVELB	SAVE left and bottom
	1	SAVERT	SAVE right and top
	2	VIEWLB	VIEWPORT left and bottom
	3	VIEWRT	VIEWPORT right and top
	4	WINDLB	WINDOW left and bottom
	5	WINDRT	WINDOW right and top
	6	INSTLB	INSTANCE left and bottom
	7	INSTRT	INSTANCE right and top
	12	HITANG	HIT COUNT and ANGLE COUNT
	13	SELINT	SELECT and INTENSITY
	10	NAME	NAME register
Four component:	14	SAVE	
	15	VIEW	
	16	WIND	
	17	INST	

Figure I.4
The Clipper Registers

THE DISPLAY PROCESSOR

The display processor is the interface between the main memory and the rest of the display system. Because this interface executes programs from memory, because it has an instruction repertoire, and because it is entirely reasonable for people to write programs for the display processor, it can be thought of as a computer. It is a peculiar kind of computer whose capabilities are oriented toward making pictures, rather than performing computational tasks. Many of its instructions are descriptive of drawing operations rather than computational operations. Because the display processor operates in conjunction with a general purpose computer, we have not endeavored to give it 'general' computing capability. For example it is not clear whether (even with the greatest ingenuity) it could be made to compile its own code. The function of the display processor is to interpret descriptions of pictures stored in the main memory and cause the intrepretations to be displayed or returned to memory. The instruction set of the display processor is described in detail in the following chapters. This section outlines its capabilities and introduces some of the ideas basic to its operation.

Display Programs

The display processor makes a picture by executing a program stored in main memory. The display processor PROGRAM COUNTER (PC) usually determines the location in memory from which the next instruction is to be taken, and the PC is incremented each time it is used to fetch an instruction. There are several instructions which can change the flow of control of the program; JUMP, CONDITIONAL JUMP, and PUSHJUMP are examples. The PUSHJUMP instruction gives the display processor subroutine capability. The subroutine return (the old contents of the PC) is stored in a marked stack, whose location in memory is kept by the STACK POINTER (SP). We call the stack 'marked' because what is written into memory is not only the program counter (in the right half word), but also a JUMP instruction (in the left half). The contents of any of the display processor's 12 other half word registers may be saved on the stack with the appropriate PUSH instruction. What is pushed onto the stack is a LOAD IMMEDIATE instruction of the particular register which is pushed. The marked stack is kept backwards from usual stack conventions, i.e. the SP is decremented before writing, so that what appears on the marked stack can be read just like a program. When a subroutine is called, it may push any of the display processor registers which it will destroy. The subroutine may call other subroutines, or even itself, to produce pictures of pictures. A subroutine return is accomplished by executing an instruction which causes the display processor to enter PEEL mode. In PEEL mode the display processor fetches instructions from the location specified by the SP while in PROG mode it fetches instructions from the location specified by the PC. Thus in PEEL mode the SP acts exactly like a program counter. The subroutine return causes the stack to be executed as a program, up to and including an instruction which changes the mode back to PROG (usually it is the JUMP instruction). The JUMP instruction planted by the PUSHJUMP will terminate the PEEL if it is marked with mode change information which causes the display processor to enter PROG mode. The display processor is always in either PEEL mode or PROG mode, and this information determines whether the PC or the SP will be used for the next instruction reference. Mode change information may be appended to most display processor instructions, so it is frequently possible to cause a subroutine exit (PEEL) without a separate instruction fetch.

The display processor has a group of instructions which permit the transfer of information between memory and the registers of the arithmetic units, such as the window and the viewpoint registers of the clipping divider. These instructions are multiple register transfers, and information can be transferred in units of from one to sixteen ful words. For example, it is possible with a single instruction to load the double word window register of the clipping divider, or both the window and viewpoint, or all the registers of the clipping divider. The external register transmission instructions fall into two groups. The first group, LOAD and STORE, use the READ ADDRESS REGISTER (RAR) as a pointer to data locations in memory. The address specified with a LOAD or STORE instruction may give the first address in a block to be transferred. The first part of a LOAD or STORE command may put that address into the RAR' From 1 to sixteen registers may be loaded or stored. The second group, SINK and RETRIEVE instructions, use the DATA SINK POINTER (DSP) to maintain an unmarked 'data' stack. The SINK and RETRIEVE instructions may optionally specify an initial value for the DSP, then from one to sixteen registers may be 'pushed' (SINK) or 'popped' (RETRIEVE).

Drawing Definition

The display processor has a large set of drawing definition instructions. These instructions pass drawing information to the external devices and initiate computational activity which typically results in a picture being posted on the scope. The basic instructions in this group are SET, DRAW TO, DRAW FROM, DOT, and BOX. In their simplest form these instructions address a single item of coordinate data in memory, one word if the display processor is in '2D' mode and two consecutive words if it is in '3D' mode. The data may be interpreted as either ABSOLUTE, RELATIVE, SIZE ABSOLUTE, or SIZE RELATIVE.

A special feature of the display processor is that a single drawing instruction may address a table of coordinate information. For this purpose the drawing instruction is marked with a mode change which causes the display processor to enter repeat (REPT) mode. The length of the table is specified by loading the processor's READ COUNT REGISTER (RCR) with (the 2's complement of) the length of the table. The location of the table is specified by the READ ADDRESS REGISTER (RAR), which increments through the table as the instruction proceeds.

Since the coordinate items in the table are not addressed individually by instructions which specify how each word is to be interpreted, the items in the table must be in some homogeneous format. An inhomogeneous specification is an arbitary sequence of drawing operations, SET, DRAW TO, DRAW FROM, and DOT and their associated coordinates. The drawing operations are not in any regular order or sequence, but simply represent a convenient way to draw a picture. Drawings can be specified in this inhomogeneous way by writing a program with SET, DRAW TO, DRAW FROM, and DOT instructions in which each instruction addresses a coordinate from memory. (Of course several instructions in the sequence may refer to the same address.) An homogeneous representation uses a table of coordinates in memory, and because the sequence is orderly the position of a given point in the table determines how it is to be interpreted. Because the interpretation changes in a regular, periodic way as one steps through the table, we think of the drawing as being defined by the coordinates and an associated repeat sequence. For example a reasonable format for specifying a collection of disconnected line segments is one in which the first coordinate item is interpreted as a SET ABSOLUTE, the second by a DRAW TO RELATIVE, the third by a SET ABSOLUTE, and so on. To specify a collection of chain connected line segments the first item in the table would be a SET followed by DRAW TO's. All of the simplest and most common repeat sequences are built into the hardware of the display processor, so that simple tables can be interpreted by executing a single instruction. The drawing instructions provide options for any of the following sequences:

SET, DRAW TO, DRAW TO, ---
SET, DRAW FROM, DRAW FROM, ---
DRAW TO, DRAW TO, ---
DRAW FROM, DRAW FROM, ---
SET, DRAW TO, SET, DRAW TO, ---
DRAW TO, SET, DRAW TO, SET, ---
DOT, DOT, DOT, ---
BOX, BOX, BOX, --- (obscure usefulness)

The interpretation of the data may be in any of these sequences:

ABSOLUTE, ABSOLUTE, ABSOLUTE, ABSOLUTE ---
RELATIVE, RELATIVE, RELATIVE, RELATIVE ---
ABSOLUTE, RELATIVE, RELATIVE, RELATIVE ---
RELATIVE, ABSOLUTE, ABSOLUTE, ABSOLUTE ---
ABSOLUTE, RELATIVE, ABSOLUTE, RELATIVE ---
RELATIVE, ABSOLUTE, RELATIVE, ABSOLUTE ---
SIZE RELATIVE, SIZE RELATIVE, SIZE RELATIVE ---
SIZE ABSOLUTE, SIZE ABSOLUTE, SIZE ABSOLUTE ---

In repeat mode the processor repeatedly executes the instruction originally fetched, modifying it upon each repetition as required in order to produce the sequences listed above. For this purpose the instruction to be repeated is kept in a special instruction register, called the REPEAT STATUS REGISTER (RSR). This register also preserves the repeat status should the processor be interrupted by the CPU in the midst of a table. When the count in the RCR runs out, the processor leaves repeat mode and fetches the next instruction. While the display processor is in repeat mode it is behaving very much like a 'display channel', and requires no instruction references to memory. As long as a table of coordinate information is in one of the standard repeat sequences, it may be displayed with a single instruction.

More elaborate repeat sequences can be defined in programs which use SET, DRAW TO, DRAW FROM, and DOT instructions which use the table addressing mechanisms of the processor rather than direct addressing. If the programmer has a table of data which is in a homogeneous format, but not in one of the repeat sequence formats provided in the hardware, a program loop is indicated. Let us suppose for example that a CPU program has generated a table of data which is to be interpreted using a repeat sequence:

```
SET, DRAW FROM, DRAW FROM, SET, DRAW FROM, DRAW FROM, SET, ---

ABS,    REL    ,    REL    , ABS,    REL    ,    REL    , ABS, ---
```

Such a format defines a collection of points on the page, each one of which has two lines drawn to it. This might be a convenient way to draw a collection of timepieces, each with an hour and a minute hand. Because this format is not one of the standard ones available in the hardware, it is necessary to write a program to interpret the table. First the RAR is loaded with the address of the table and the RCR is loaded with (the 2's complement of) the number of timepieces. Next the program has a 4 instruction loop: SET ABSOLUTE; DRAW FROM RELATIVE; DRAW FROM RELATIVE; INCREMENT RCR and JUMP IF RCR NEGATIVE. The drawing definition instructions use the RAR to specify an address, and the program accordingly will step through the table of data.

There are three addressing mechanisms for drawing definition instructions. (1) The first mechanism simply obtains the data and treats it as coordinate information. If repeat mode is specified, the number of complete data items (X, Y or X, Y, Z, Z) read from memory is determined by the RCR. If repeat mode is not specified, a complete data item, one word in two dimensions or two words in three dimensions, is read from memory. (2) A second mechanism interprets the word addressed as a pair of pointers. The display processor uses first the left half and then the right half to obtain two pieces of coordinate information. If repeat mode is specified, the RCR should specify the number of pointer pairs. If repeat mode is not specified, one pointer pair is used to obtain two complete pieces of coordinate information. The way in which these two items are interpreted must be specified in exactly the same way as the repeat sequence. Thus if the pointers are used to point to the end points of a line segment, both specified absolute, the instruction should specify 'SET, DRAW TO, SET, DRAW TO,...' and 'ABSOLUTE, ABSOLUTE, ...' as the repeat sequence. (3) The third mechanism is for dataless operations for which the external device uses data stored internally to perform the indicated operations. In repeat mode the operation will be done a number of times specified by the RCR but no data will be read from memory. If repeat mode is not specified, then the operation will be performed but once, again with no data references to memory.

Operating Modes

What we have described thus far are some of the ways in which a drawing may be represented in memory. The display processor reduces this potentially intricate representation to a simple sequence of coordinates and drawing commands, which are passed to the external devices such as the matrix multiplier, clipping divider, and scope. The way in which a drawing is processed by the external devices is determined by the contents of the DIRECTIVE REGISTER (DIR). The DIR contains all the modal type information which is not able to change instruction by instruction. Because this display system is an entirely digital device up to the point at which a picture is produced on the scope, and because it has arithmetic capabilities that are useful for many display-related tasks, the display system may be used to do more than produce an image on the scope. For example a drawing may be rotated, scaled, clipped, and placed in perspective and then returned to memory for output on a plotter or remote display. One may determine which lines in a picture pass through a given rectangular region, a facility which is useful for 'pointing' with a tablet or a 'mouse'.

Usually our goal is to paint a picture on the scope. If the drawing is two dimensional and need not be rotated, we specify a directive which establishes a computation path in which data from memory is sent to the clipping divider, and the scaled and clipped results are painted on the scope. Because the processor, clipping divider, and scope are independently timed, the processor may be fetching the next line from memory while the clipping divider is processing its current line, and the scope is painting the previous line. The orderly streaming of data is insured by a set of initiation and acknowledge signals appropriate to the path specified by the directive. Processing by the matrix multiplier may be included in this 'pipeline' so that two or three dimensional drawings may be rotated and scaled preliminary to being sent to the clipping divider.

The outputs of either the matrix multiplier or clipping divider may be returned to memory, possibly in addition to being passed to the next unit on the pipeline. For example the clipping divider may send its processed results to the scope, to memory, to both, or to neither. While memory-to-memory operation of each unit in the system is independent, confusion may result if more than one unit is directed to send output to memory. Information is written into memory at the location specified by the WRITE ADDRESS REGISTER (WAR), and the WAR is incremented after each word is written. An upper limit on the number of words written may be specified by the WRITE COUNT REGISTER (WCR). If the count in the WCR runs out, the processor fetches no new instructions or data but continues writing in memory until the pipeline processing units finish any tasks in process. The processor then stops.

The memory-to-memory operation of the display system makes its special computational power available to the programmer. We anticipate at least two uses for memory to memory operation.

The first is to facilitate using secondary, and generally slow, output devices such as plotters or storage tube displays. For this purpose the clipping divider's 'scope output' is sent to memory, and a viewport may be used which is convenient for the coordinate system and resolution of the secondary graphic device. A second application for memory-to-memory operation is as a filtering technique to obtain a smaller basic display file in cases where a very small part of a large and complex drawing is to be viewed. For this purpose the clipping divider's 'page output' (its output after clipping but before the perspective division and window-to-viewport mapping) is sent to memory.

In addition to specifying the form of the pipeline computation, the directive may specify several special operating modes. For example the directive may cause all lines plotted on the scope to be dashed, or may cause the clipping divider to operate in minimum effort, curve, or graph mode. The automatic stop mode is useful for performing tests which determine what is being 'pointed' at by a tablet, light pen, or comparator. The uses of the various directives are outlined in a following chapter and illustrated in the programming examples.

Communication with the CPU

Considerable care has been exercised to insure that the display system will operate gracefully in a multiple-user or time shared environment. The display may be interrupted by the CPU at the end of any instruction, and during the execution of a repeat mode sequence or multiple LOAD, STORE, SINK, or RETREIVE. The longest uninterruptable instruction, the INDIRECT DATA INTERPRETATION INSTRUCTION requires a maximum of 6 memory cycles to complete. The process can be terminated and later resumed, since the state of the instruction execution is saved in the repeat status register. The CPU may inject a 'pause' request by means of a 'conditions out' (CONO) instruction, and may then give commands to the display processor with 'data out' (DATAO) instructions. The full I/O word transferred by a 'data out' (DATAO) is interpreted as a standard display processor instruction. PUSH and STORE instructions issued in this way can be used to save the entire state of the display system in memory so that after processing another user's material it is possible to resume the interrupted process. A program to interrupt and restore a user is shown in the programming examples.

The display processor includes provision in its memory and I/O bus interfaces for the addition of mapping hardware. The mapping hardware may use the display processor's memory buffer register to make references to a page table in main memory, and may use bits in the 'conditions in' (CONI) word to indicate and cause interrupts on protection violations.

II. DISPLAY PROCESSOR REGISTER CONFIGURATION

The display processor has 13 internal registers which are of interest to the programmer. These registers are each 18 bits wide, and are arranged as a 16 X 18 fast register memory. Each of these processor registers is assigned a special function, either as an address pointer, a word count, or a control word. They are in a register memory configuration largely for convenience in addressing them individually, and should not be thought of as 'interchangeable'.

The display processor also has a memory buffer register (MBR), a read buffer register (RBR), and a write buffer register (WBR). All information passing to and from memory goes through the MBR. The MBR includes a parity net for generating and checking parity of words written to and read from memory. The RBR is used to hold information that is to be sent to the external units, and is normally loaded by the display processor in anticipation of an external device being able to accept the information (data prefetching). The WBR is used in memory-to-memory operation of the display system to buffer information waiting to be written into memory. The RBR and WBR are used concurrently, and write cycles are given a simple priority over read cycles. A diagram of the display processor's registers and data paths is shown in Figure II.1.

It is important to realize that this register configuration imposes a strong separation of control information, such as addresses, word counts, and directives, from the coordinate information. The internal registers of the display processor never hold coordinate data, and conversely the registers of the external devices never hold address information. Because processor registers can be stored in memory only by pushing them onto the marked stack (using the stack pointer SP) and loaded only by 'load immediate' instructions, there is no direct mechanism for transferring the contents of one processor register into another processor register, or into a register of one of the external devices. Because information is transferred to and from registers in the external devices using addressable load and store instructions, or onto their own 'stack' (using the data sink pointer DSP), there is considerably more flexibility in, for example, placing window information in a viewport register. There is however no direct way to get information from an external device register into a processor register.

Outlined on the following page is a summary of these internal registers of the display processor which are accessible by the programmer, together with the particular function of each register.

Processor Registers

PROGRAM COUNTER (PC) In PROG mode the PC indicates where the next instruction is to be found in memory. The PC is incremented immediately after it is used to fetch an instruction from memory. The PC may be loaded immediate to effect a JUMP instruction, and its previous value can be pushed to effect a PUSHJUMP instruction.

STACK POINTER (SP) The SP points to the last filled location in a marked stack where 'last' is toward a lower numbered memory address. In order to push an instruction onto the stack, the SP is decremented and the instruction is written at this location. In PEEL mode the stack pointer operates exactly like the program counter, incrementing after it is used to fetch an instruction from memory. This stack is the basic mechanism for accomplishing subroutining. The SP may be loaded immediate in order to go to a different stack, and its old value may be pushed onto the new stack.

DATA SINK POINTER (DSP) The DSP points to the first unused location in an unmarked 'data' stack where 'first' is toward a higher numbered memory address. To push a 36 bit data word from an external device, the word is written in the location specified by the DSP, and then the DSP is incremented. To 'retreive' a word from the data stack, the DSP is decremented before a read. [Note: While the SP loads the marked stack towards lower addresses so that the code written be in the usual reading-toward-larger-addresses way, the DSP loads the data stack towards higher addresses.] The DSP may be loaded immediate or pushed onto the marked stack, or both.

READ ADDRESS REGISTER (RAR) RAR points to the location in memory from which the next word of data is to be read in addressable load or display instructions. It is incremented each time it is used so that it can point through tables. It is also used to hold the location of an instruction to be executed by an EXECUTE instruction. RAR can be loaded immediate or pushed onto the marked stack, or both.

WRITE ADDRESS REGISTER (WAR) WAR points to the location in memory into which the next word of data is to be written in memory-to-memory operations. It is incremented each time it is used so that it can write tables of data. WAR can be loaded immediate or pushed onto the marked stack, or both.

P1, P2 These are address pointers used as temporaries in the indirect data fetch; P2 is also used as a temporary in push operations. They may be loaded immediate or pushed onto the marked stack, or both; however they need not be saved and restored as part of the usual sequence of starting or stopping a user. The contents of P1 and P2 are in general not preserved during an instruction. Load immediate and push P2 results in pushing the new (rather than the old) data onto the stack.

READ COUNTER REGISTER (RCR) The RCR is used to specify the (2's complement of the) number of complete data items fetched in repeat mode for a given data-processing instruction. It is analogous to the block counter of a channel. When RCR becomes positive the processor leaves repeat mode. If the repeat mode operation is stopped in mid-process by the CPU, the contents of RCR specifies the number of data items remaining to complete the processing originally requested. RCR may be loaded immediate or pushed onto the marked stack, or both. At the programmer's discretion it may also be used to count iterations of particular programming processes.

WRITE COUNTER REGISTER (WCR) WCR counts the number of words (not necessarily complete data items) written into memory through WAR. WCR is normally used to limit the size of tables written in memory-to-memory mode. In memory-to-memory operation WCR is incremented whenever WAR is incremented, and when WCR becomes positive it causes the processor to discontinue data and instruction fetching. Since data may be in process when WCR reaches zero, it is possible that additional words will be written into memory so that WCR will end up containing +8 or so. At most, 9 words will be written beyond runout of WCR. WCR may be loaded immediate or pushed onto the marked stack or both. With proper care it may be used, like RCR, to count iterations.

UNASSIGNED REGISTER (UR) The unassigned register is available for programmer use as desired e.g. for storing pointers to data structures.

DIRECTIVE REGISTER (DIR) The DIR contains mode information for the display system. The use of the individual bits of the DIR are discussed in a later section, but in general they control the operating modes of the various devices in the system, including to some extent the processor itself. DIR may be loaded immediate or pushed onto the marked stack, or both.

REPEAT STATUS REGISTER (RSR) The RSR is a special instruction register, certain bits of which are loaded from bits in the instruction word of the drawing definition instructions. It may be loaded immediate or pushed onto the marked stack, or both.

STATUS REGISTER (SR) The SR stores conditions, such as program flags, for the conditional instruction. The SR exists mainly so that such conditions can be saved by pushing them and restored by a subsequent PEEL.

MEMORY ADDRESS REGISTER (MAR) The MAR specifies the location in memory for all reads and writes. It may not be loaded, but it may be pushed to assist in recovery from unusual memory situations such as parity, non-existent memory, or illegal address mapping stops.

Address (Octal)	Register	
0	RAR	Read Address Register
1	WAR	Write Address Register
2	PC	Program Counter
3	SP	Stack Pointer
4	P1	Temporary
5	P2	Temporary
6	DSP	Data Sink Pointer
7	UR	Unassigned (Name) Register
10	RCR	Read Counter Register
11	WCR	Write Counter Register
12	DIR	Directive Register
13	RSR	Repeat Status Register
14	SR	Status Register
15	MAR	Memory Address Register

Table II.1: Processor Register Addresses

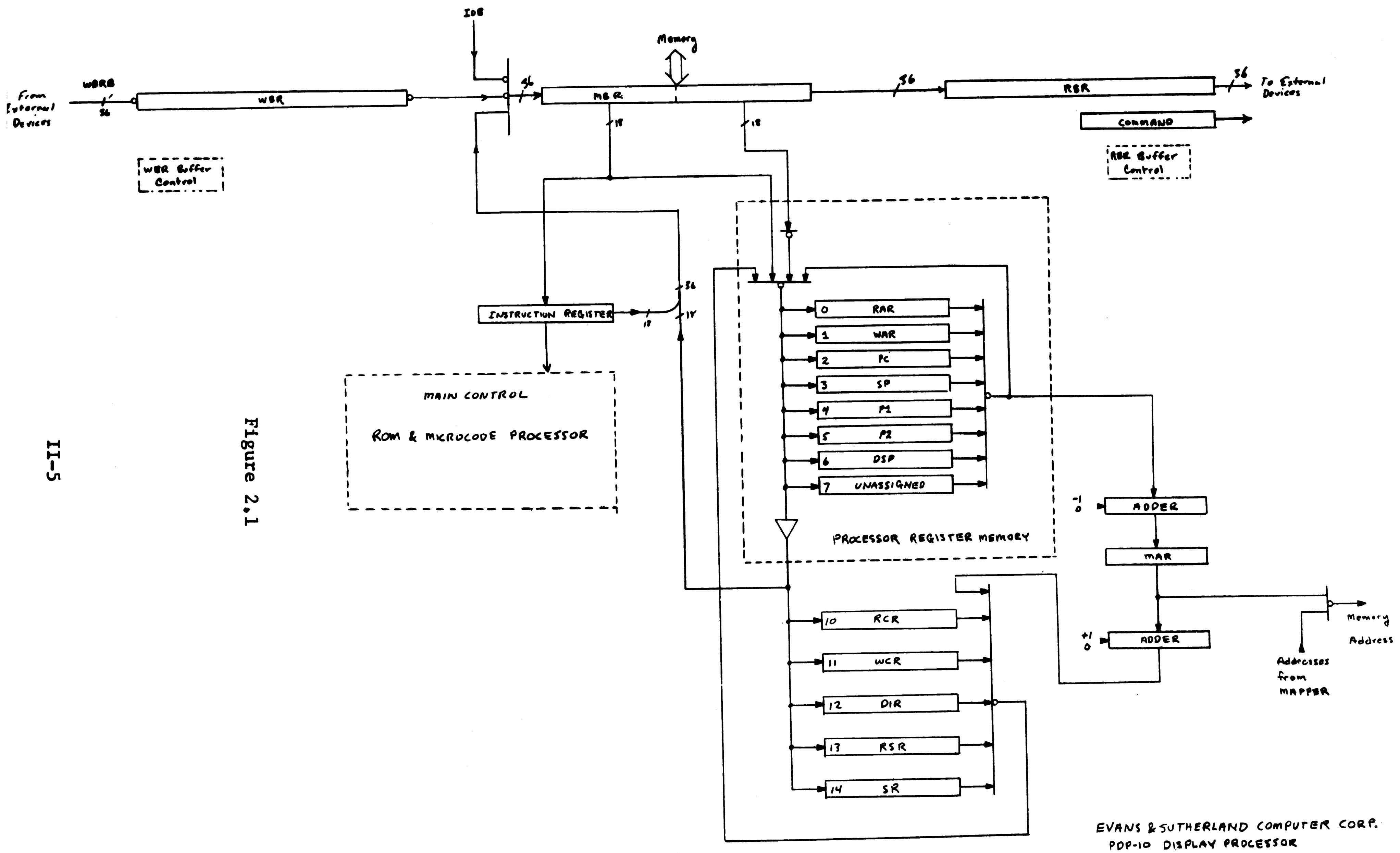


Figure 2.1

EVANS & SUTHERLAND COMPUTER CORP.
PDP-10 DISPLAY PROCESSOR

BLOCK DIAGRAM
101102-900
2-6-69
SHEET 1 OF 14

III. DISPLAY PROCESSOR INSTRUCTION REPERTOIRE

INSTRUCTION FORMAT

The instruction format of all the display processor instructions is the same. This format is:

0 8	9 12	13	14 17	18 35
Operation Code	A	I	X	Immediate Data

Display Processor Instruction Format

The instruction fields are used as follows:

(1) The operation code specifies the instruction to be performed. The first 3 bits [0-2] of this field determines which group the instruction is in, and the remaining 6 bits [3 - 8] specify variants within the group. In most cases bits 3 - 8 have a particular interpretation taken in 1, 2, or 3 bit clumps, so that many instructions may be viewed as a concatenation of the group and several independent variants.

(2) The 4-bit A field specifies the address of a register which is involved in the execution of the instruction. Except for the multiple external register transfer instruction group, the A field refers to the address of a register of the display processor which is involved in the execution of the instruction.

(3) The I bit controls a phase of the instruction execution which is common to all display processor instructions. Instructions normally commence by loading the immediate data [bits 18 - 35 of the instruction word] into a register of the display processor. The register to be effected is usually specified by the A field of the instruction, except for the multiple external register transfer group in which the variant specifies the register to be effected. The use of the I bit is to inhibit this initial load immediate, or in the conditional load immediate instruction group to invert the sense of the condition tested.

(4) The 4-bit X field is used in the multiple external register transfer group to indicate a count of the number of registers to be loaded or stored. In the conditional load immediate group the X field specifies a condition number. In the remaining instruction groups the X field specifies a change in mode. In PROGRAM mode instructions are fetched from the location specified by the program counter (PC), and the PC is incremented immediately following each fetch. In PEEL mode instructions are fetched from the location specified by the stack pointer (SP), and the SP is incremented following each fetch. In REPEAT mode the processor

makes no instruction references, but only data references, and exits from repeat mode when the count in the read count register (RCR) runs out. In EXECUTE mode, the processor fetches an instruction from the location specified by the read address register (RAR) and the RAR is incremented following its usage. The processor will exit from EXECUTE mode unless the instruction executed is marked with a change to EXECUTE mode (multi-level chain). These mode changes are coded into the X field as follows:

IR BITS 14-17	RESULTS
0 0 0 0	Do nothing
0 0 0 1	Go to PROG mode, clear EXFF and REPT
0 0 1 0	Go to PEEL mode, clear EXFF and REPT
0 0 1 1	Clear EXFF and REPT
0 1 X X	Go to REPT mode, clear EXFF (XX indicates PROG and PEEL modes as above)
1 0 X X	Go to EXEQ mode, clear REPT
1 1 X X	Go to EXEQ and REPT

Table of How to Change Mode

(5) The right half word of every instruction is immediate data. This data may be loaded into some register of the display processor, but may never be loaded into the registers of the external devices. The immediate data may be an address, a count, or a control word. The immediate data supplied with a given instruction is generally used in the later execution phases of the instruction. In particular, when the immediate data is an address it is convenient to think of the right half of the instruction word as the 'address field' of a memory reference instruction. However to emphasize the fact that addresses must be put somewhere before they are used, we persist in calling this information immediate data, even if it is an address.

We have found it convenient to write display processor instructions using the following:

```
LABEL:    MNEM   3,@FOO(12)    ;COMMENT
```

It is expected that this will be assembled so that the opcode corresponding to the mnemonic MNEM appears in bits 0 - 8, the '3' appears in the A field, the value corresponding to 'FOO' appears in the right half, the '12' appears in the X field, and the @ sign causes bit 13 to be 1. (This format is derived from that of the DEC PDP-10.) One should also be able to specify the A and X fields symbolically. An example of a display processor instruction is:

```
     LABEL:      LIFCL      DIR,100371(PFØ)      ;COMMENT
```

which is a 'load immediate if' (conditional load immediate) of the directive register (DIR) with the immediate data 100371 if program flag 0 (PFØ) is a one and clear program flag 0. When writing display processor programs one must bear in mind that the interpretation of the fields following the instruction mnemonic will depend on the mnemonic.

In the descriptions of the display processor instructions which follow, we have attempted to define a set of mnemonics for the various operation codes, and a chart of how the mnemonics are obtained by concatenation is shown on the following page.

We have defined numerous 'extended' instruction statements in cases where a particular display processor instruction together with other information in the A and I fields has a special meaning. For example an instruction:

```
     LI      PC,ADDRESS     =     JMP      ADDRESS
```

is a load immediate of the program counter, and is better thought of as a jump instruction.

CONSTRUCTION OF MNEMONICS FOR E & S DISPLAY CODES

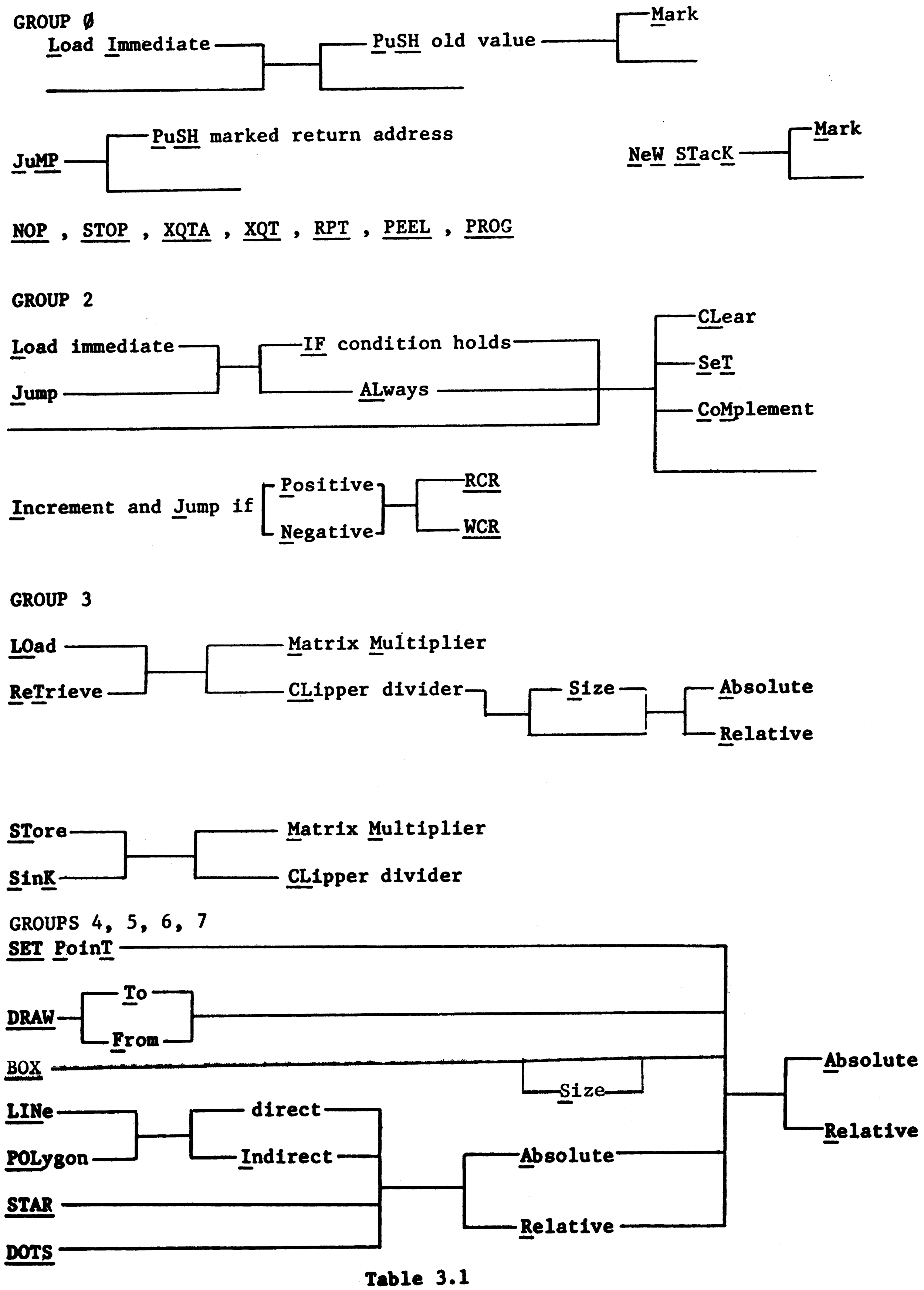
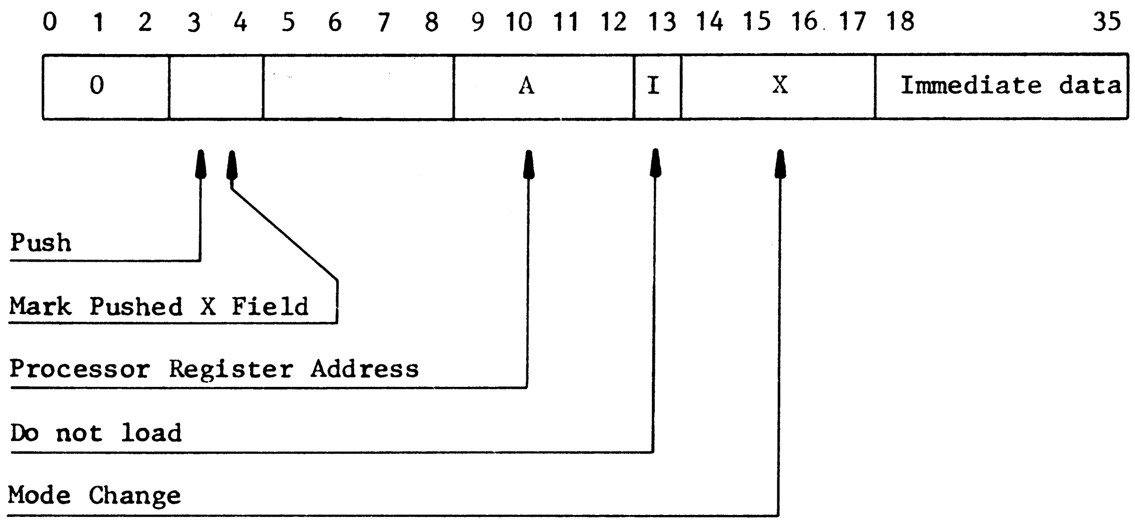


Table 3.1

THE INSTRUCTION SET

Instruction Group 0 - LOAD IMMEDIATE, PUSH OLD VALUE, AND CHANGE MODE



The sequence of operation is:

(1) The contents of the processor register specified by the A field is placed into the temporary P2.

(2) If the I bit is zero, the immediate data is placed into the processor register specified by the A field.

(3) If the push bit [3] is one, the contents of P2 are pushed onto the marked stack in the right half word, together with the A field of the instruction in the left half. If the 'mark pushed X field' bit [4] is one, the current mode of the processor is also written in the X field of the pushed word. Thus the word written onto the top of the stack is a load immediate of the register being pushed, possibly marked with the processor's mode.

(4) The mode of the next instruction fetch is determined from the X field of the instruction.

This group provides the following instructions:

LI	[000 n, WHAT] A load immediate of the register specified in the A field.
LIPSH	[040 n,WHAT] A load immediate of the register specified in the A field, followed by pushing the old contents of that register onto the marked stack.

LIPSHM	[<u>060</u> n,WHAT] Same as LIPSH, except that the pushed word is marked with the mode the processor was in when the LIPSHM was executed.
PSH	[LIPSH n,@] The register specified in the A field is pushed onto the marked stack.
PSHM	[LIPSHM n,@] Same as PSH, except that the pushed word is marked with the mode the processor was in when the PSHM was executed.
NOP	[LI ,@] A no-operation.
XQT	[LI 0,WHAT (XQTM)] An execute instruction.

Because the program counter and stack pointer may be addressed by the A field of the instruction, the following instructions result:

JMP	[LI PC,] A 'jump' instruction.
JMPPSH	[LIPSHM PC,] A 'pushjump' instruction. Notice that JMPPSH marks the X field of the pushed JMP instruction.
NWSTKM	[LIPSHM SP,] The 'new stack' instruction first loads the SP thus creating a new stack, and then saves the old SP in the new stack together with the current mode of the display processor. When executed from the main processor via a 'data out', this instruction preserves the most volatile information about the state of the display processor in such a way that the remaining registers can be pushed onto this new stack.

Any of the above instructions may be appended with mode change information, which is specified in the X field. For example:

```
     LIPSH    RCR,-100(PEELM)        ;LOAD RCR WITH -100 AND PEEL
```

causes the display processor to enter peel mode after executing the LIPSH instruction. A simple PROG, PEEL, REPT, or XQT is implemented by a mode change appended to a NOP. For example:

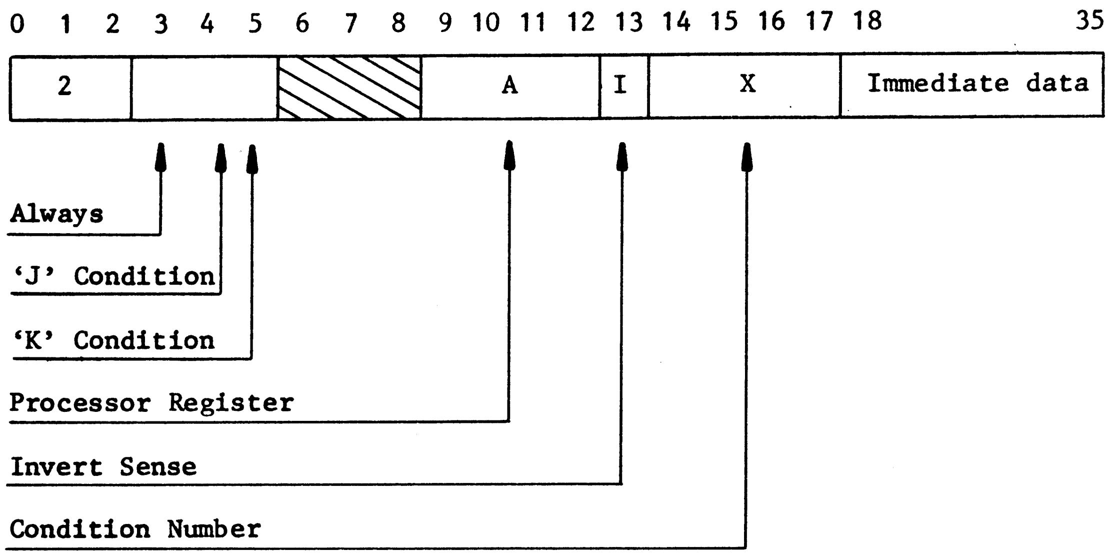
```
     XQT  [ NOP    ,(XQTM) ]  Load over and execute.
```

[NOTE: Because the processor register P2 is used as a temporary for push operations, the instruction:

```
     LIPSH    P2,DATA
```

pushes not the old value of P2 onto the stack, but instead the new contents, 'DATA'. PSH and LI operate as expected. This peculiarity may be used to push a name onto the stack with a single instruction.]

Instruction Group 2 - CONDITIONAL LOAD IMMEDIATE



The condition specified in the X field is tested, and if it is '1' while the I bit is '0', or '0' while the I bit is '1', then the immediate data are placed into the processor register specified by the A field.

After the condition is tested it is modified according to the J and K bits:

J	K	Result
0	0	no change
0	1	cleared
1	0	set
1	1	complemented

The following conditions are supplied:

Condition number	Condition
0 - 3	Program flags 0 - 3
10	RCR Negative and $\neq -1$
11	WCR ,, ,, ,,

12	'HIT' [Clipper window hit by line]
13	'AIC' [Area in common between INSTANCE and WINDOW.]
17	PROG STOP FLAG.

Unassigned conditions are available for addition of conditions by the customer.

Conditions 0-17$_{(8)}$ appear in the status register, bits 18-33, so that a user's program flags, etc. will be preserved if he is interrupted. Pushing the status register (SR) saves the conditions and loading the status register loads the conditions. However the sign bits of RCR and WCR cannot be loaded either by loading the SR or by JKing conditions 10 or 11. So that the RCR and WCR may be used to write simple iteration loops, they are incremented after being tested if the J bit is one. The HIT bit is set whenever a set-point, line or dot falls within the WINDOW. The AIC bit is cleared preliminary to a BOX operation, and set if there is any area in common between the WINDOW and INSTANCE.

We use the following mnemonics for group 2 instructions:

LIF	[$\underline{200}$ n,WHAT(condition)]	Load if condition is one.
LIFCL	[$\underline{210}$ n,WHAT(condition)]	Load if -- and clear condition.
LIFST	[$\underline{220}$ n,WHAT(condition)]	Load if -- and set condition.
LIFCM	[$\underline{230}$ n,WHAT(condition)]	Load if -- and complement condition.
LAL	[$\underline{240}$ n,WHAT(condition)]	Load always.
LALCL	[$\underline{250}$ n,WHAT(condition)]	Load always -- and clear condition.
LALST	[$\underline{260}$ n,WHAT(condition)]	Load always -- and set condition.
LALCM	[$\underline{270}$ n,WHAT(condition)]	Load always -- and complement condition.

If the program counter is the register addressed, the instruction is a conditional jump, i.e.:

JIF	[LIF PC,]	Jump if condition is one.
JIFCL	[LIFCL PC,]	Jump if -- and clear conditon.

etc. through

JALCM	[LALCM PC,]	Jump always -- and complement condition.

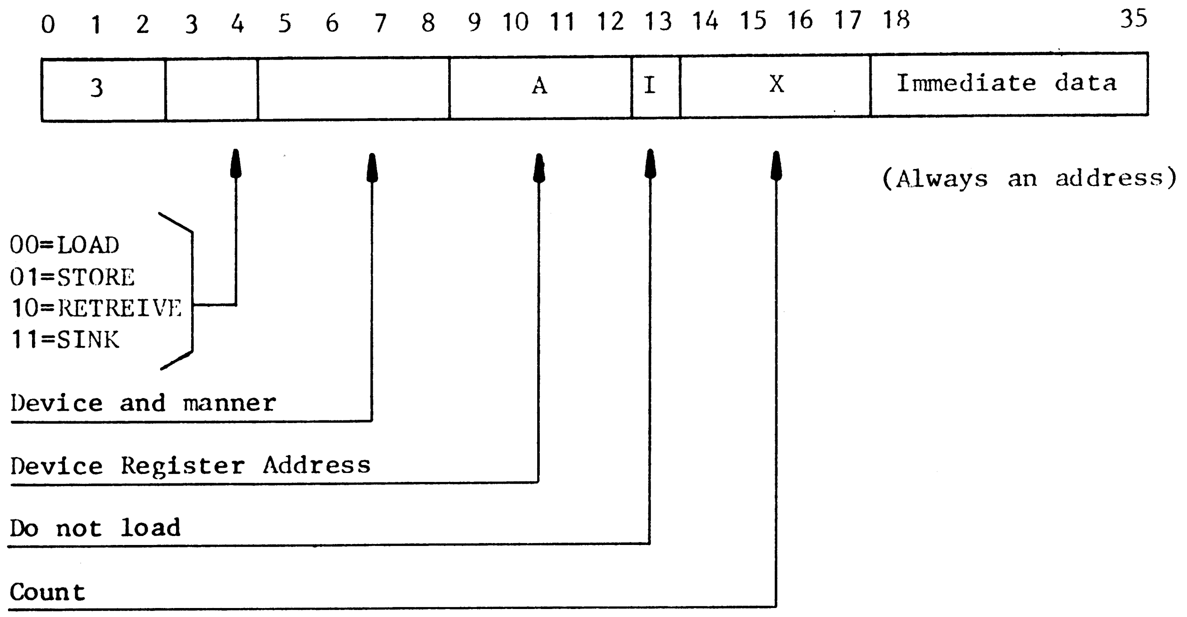
The sense of any conditional may be inverted by the indirect bit, so that:

```
          JALST    @(PFØ)        never jumps, but sets PFØ.

          JIFST    @A(PFØ)       jumps to A if PFØ = 0, and sets PFØ.
```

Incrementing and testing RCR or WCR is a simple iteration loop termination:

```
IJNRCR     [ JIFST      (RCRN) ]  Increment RCR and jump if (result) negative.

IJNWCR     [ JIFST      (WRRN) ]  Increment WCR and jump if (result) negative.
```

Instruction Group 3 - EXTERNAL REGISTER TRANSMISSION



These instructions are provided to load and store the parameter registers of the external devices. Group 3 instructions begin with a load immediate to RAR (LOAD and STORE), or to DSP (SINK and RETREIVE), unless the I bit is one. The immediate data will always be an address. The I bit is usually one for SINK and RETREIVE instructions so that the DSP is not loaded preliminary to the data transfers. The X field indicates how many data items are to be transferred to or from the specified device, and the A field specifies the first device register to be effected. After each transfer the X field is decremented and tested for zero, and A is incremented (LOAD, STORE, or SINK) or decremented (RETREIVE). Thus the X field specifies how many transfers are to be accomplished. An initial value of X = 0 specifies that 16 data items are to be transferred. The LOAD and STORE variants use the RAR to specify the address in memory, and the RAR is incremented after it is used. The SINK variant uses DSP to specify the memory address to be written, and the DSP is incremented after it is used, pointing to the next (vacant) location towards higher memory addresses. The RETREIVE variant is a special case, using DSP to specify the memory address to be read, however the DSP is decremented before it is used. Thus SINK and RETREIVE are consistent with the direction of the DSP stack. SINK and RETREIVE work correctly for data items even though they may occupy more than one word in memory.

Bits 5-8 in group 3 instructions specify both the external device and other information as required regarding the manner in which the data is interpreted. The assignments currently made are:

5 6 7 8	Device and Manner	Abbreviation
0 0 0 0	Clipper - absolute	CLA
0 0 0 1	Clipper - relative	CLR

0 0 1 0	Clipper - center size absolute	CLSA
0 0 1 1	Clipper - center size relative	CLSR
0 1 X X	Matrix Multiplier - absolute	MM

The abbreviations indicated above are appended to the 4 basic instructions: LOAD, STORE, RETR, and SINK, in order to specify completely the operation to be performed. Of course one may STORE or SINK only absolute. In two dimensions a typical instruction in this group is:

```
LOCLR      WINDLB, VALUE (2)
```

which adds the contents of memory location 'VALUE' and 'VALUE + 1' to the SAVELB and SAVERT respectively, and leaves the results in the window register WINLB and WINRT respectively. The instruction:

```
SKCLA      SAVELB,@ (14)
```

saves all the registers of the clipping divider in the data sink. Please note that if for example the viewport and window (VIEWLB, VIEWRT, WINDLB, WINDRT) are pushed into the data sink with an instruction:

```
SKCLA      VIEWLB,@ (4)
```

they must be retreived in the opposite order:

```
RTCLA      WINDRT,@ (4).
```

For convenience this may be written:

```
RTCLA      VIEWLB+3,@ (4)
```

The clipping divider select register, which is used to determine which scope(s) the picture is to appear on, provides protection against a user painting a picture on another user's scope(s). When an instruction which loads the select register originates from the main processor, both the select register and a select mask register are loaded with the select information provided. The select mask is otherwise entirely inaccessible to the processor user. An alarm is provided in the 'conditions in' (CONI) word in case a user attempts to select a scope which is masked off to him and the processor is stopped.

The high eight bits of the SELECT register contain select information for scopes 0-7. The next eight bits contain the select mask (PERMIT) for scopes 0-7. Scope 0 should stand up as the real scope.

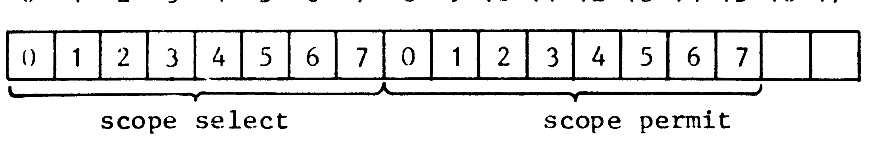
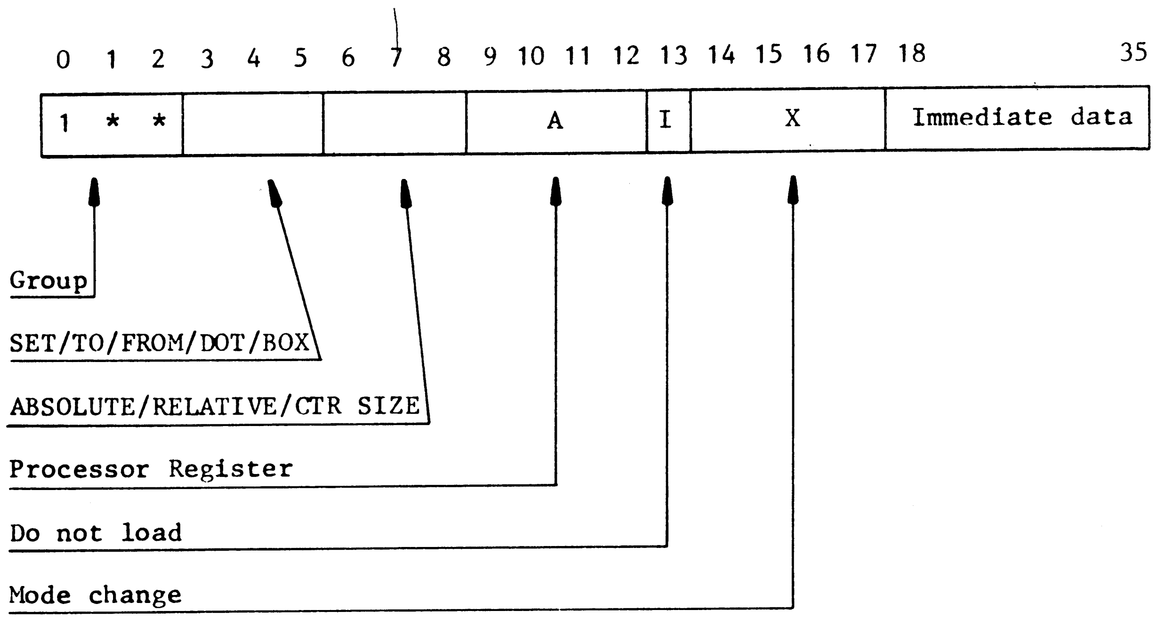


Figure 3.2

↑ THE OPERATION OF THIS FEATURE IS QUESTIONABLE. (BIT POSITIONS MAYBE DIFFERENT)

Instruction Groups 4, 5, 6, 7 - DRAWING DEFINITION INSTRUCTIONS



Instructions in these groups start by placing the immediate data into the processor register specified by the A field, unless the I bit is one. Second, the X field is used to determine any mode change, and the contents of the instruction register, bits 1-8, are copied into the corresponding positions in the repeat status register (RSR).

If the instruction is not marked to cause the processor to enter repeat mode, the contents of RAR is used to specify an address in memory from which to obtain either:

(1) A single word of coordinate information in 2D mode, or two consecutive words of coordinate information in 3D mode.

(2) A pointer word whose left half is placed in P1 and whose right half is placed in P2. First P1 and then P2 are used to obtain two complete coordinate items from memory. Each coordinate item consists of one word in 2D mode or two words in 3D mode.

(3) No information from memory.

The addressing mechanism used is determined by the instruction group. Group 4 corresponds to item (1) above, and is called DRAW DIRECT. Group 5 corresponds to item (2) and is called DRAW INDIRECT. Group 6 corresponds to item (3), is called DRAW INTERNAL, and is used in cases where the data is not required for the indicated operation. Group 7 is reserved for non-coordinate information such as characters and video data. After all of the data have been obtained and have been passed to the external devices, the processor takes the next instruction in sequence.

If the instruction has been marked to cause the processor to enter repeat mode, the instruction held in the repeat status register is executed repeatedly, exactly as described in the previous paragraph. Because the RAR is incremented each time it is used to obtain a data item from memory, successive executions will point through a table of data. This table may consist of 2D or 3D coordinate information, pointer words, or may be non-existent in the DRAW INTERNAL case. In order to count out the length of the table, the read count register (RCR) for each repetition of the instruction is incremented (if the processor is in repeat mode). The RCR should contain initially (the two's complement of) the number of DATA items to be processed. In DRAW DIRECT that is the number of points (each point is 2 words in 3D); in DRAW INDIRECT it is the number of pointer pairs (one word each); and in DRAW INTERNAL it is the number of DRAW INTERNAL operations. Each time an execution is completed, the sign of the RCR is tested to determine whether the processor should exit from repeat mode, and the pause request bit is tested to determine whether CPU has requested to interrupt the instruction.

As described in Chapter I, the DATA processes in repeat mode are interpreted in one of several simple sequences. The particular sequence to be used is specified by bits 3-8 of instructions of these groups. Each time a coordinate item is read from memory, it is tagged with the next interpretation in the specified sequence. This information comes from the RSR bits 21-26 which are initialized from the corresponding bits (3-8) in the instruction word. A table of all possible sequences was listed in Chapter I. Largely to facilitate the formation of mnemonics, we have ascribed to each sequence a name which is characteristic of what the sequence might be used to draw; for example SET, DRAW TO, DRAW TO, --- is reminiscent of a POLYGON (POL), SET, DRAW FROM, DRAW FROM, --- is useful for STAR (STAR)-like figures, and SET, DRAW TO, SET, DRAW TO, --- is used for disconnected LINES (LIN).

In an entirely similar way one may specify a sequence of loading methods: ABSOLUTE, RELATIVE, SIZE ABSOLUTE, SIZE RELATIVE, in bits 6-8 of the instruction word, according to the coding illustrated in figure 3.1. A table of all possible sequences may be found in Chapter IV. We have assigned to each sequence a two-letter abbreviation.

For the addressing mode we use the two-letter abbreviations:

DD	for	DRAW DIRECT
DI	for	DRAW INDIRECT
DN	for	DRAW INTERNAL

To specify a line drawn by indirection through a pointer word, in which the left half points to an absolute coordinate and the right half points to a displacement relative to the first coordinate, we concatenate DRAW INDIRECT (DI), LINES (LS), ABS-REL (AX) and obtain:

```
DILSAX      RAR,PNTR
```

These mnemonics include 192 distinct instructions and it is fairly easy to get used to them. In order to simplify life a little, we have defined a set of extended instruction mnemonics, all of which place the immediate data in RAR so that they appear to be memory reference instructions. The complete table of such mnemonics can be found in Chapter IV. A few examples are listed below:

SETPTA	[DDLSAB 0,]	Set absolute
SETPTR	[DDLSRE 0,]	Set relative
DRAWTA	[DDTOAB 0.]	Draw to absolute
DRAWTR	[DDTORE 0,]	Draw to relative
DRAWFA	[DDFRAB 0,]	Draw from absolute
DRAWFR	[DDFRRE 0,]	Draw from relative
DOTSA	[DDDTAB 0,]	Dot absolute
DOTSR	[DDDTRE 0,]	Dot relative
BOXA	[DDBXAB 0.]	Box absolute
BOXR	[DDBXRE 0,]	Box relative
BOXSL	[DDBXSL 0,]	Box size relative

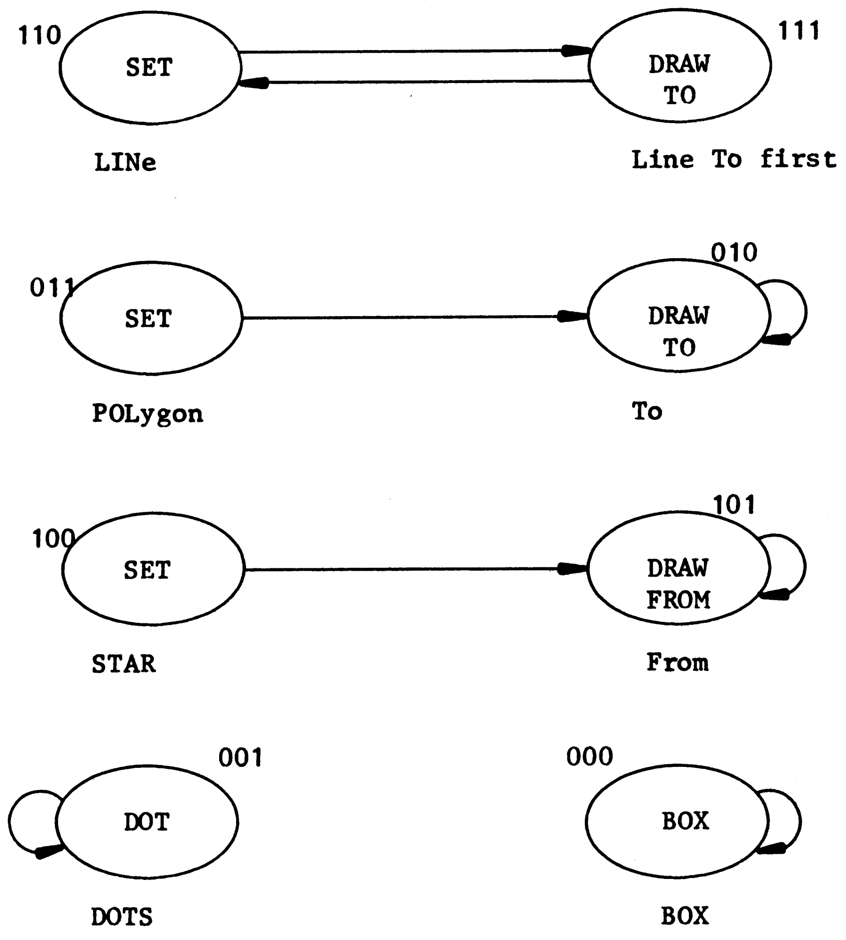


Figure III.2: Specification of Set/Draw and Dot and Box

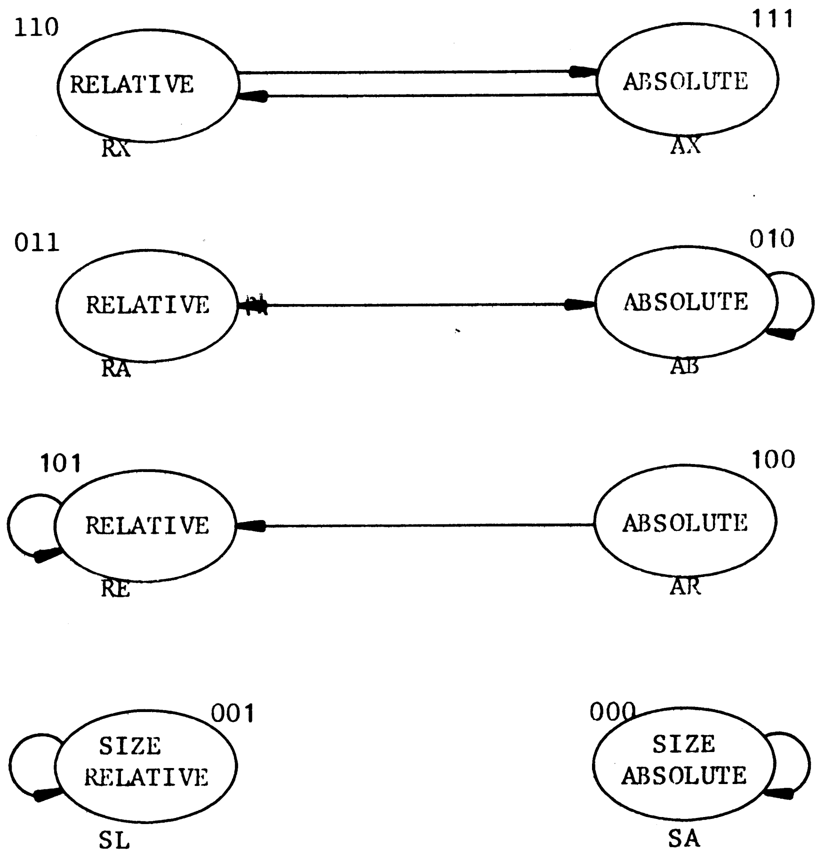


Figure III.3: Specification of Absolute/Relative sequences

IV. CONTROL STATUS

INTRODUCTION

This chapter lists the detailed mnemonics, control bits, internal register addresses, and I/O control bits for the E & S Line Drawing System Model 1. This information is presented in several tables for ease of reference. Individual instruction classes and sample instructions have been discussed in Chapter III; actual programs using this information are presented in Chapter V.

MNEMONICS

Table 4.1 lists the format for the basic LDS-1 instructions. The mnemonics for the instructions listed can be specialized by appending various sub-mnemonics. Particular mnemonics are constructed as diagrammed in Table 4.2. As an example, the first instruction of Table 4.1, LI****, can be specialized to LIPSHM, LIPSH, or just LI. These three instructions are developed by starting at Load Immediate at the upper left of Table 4.2 and following the diagram to the right. At each move, the capital letters for the branch chosen are appended to the previously existing letters. A branch with no letters denotes an ending (or a starting) point. Thus, PSH and PSHM are also legitimate instruction mnemonics.

Table 4.1 also shows the correct form of the string of arguments for each instruction. Normally, the setting of the indirect bit (bit 13) will be denoted by the "@" symbol placed in front of the Data or Memory Address field of the instruction. This is the convention followed in Chapter III and which will be followed in Chapter V.

Table 4.3 lists the mnemonics for the operand arguments that denote the Display Processor Registers, the Next Mode, the Conditions, and the Clipper Divider Registers. With each mnemonic is listed the *octal* number of the register, mode or condition. In the case of the registers, this number denotes the "address" of the register in the E & S equipment. In the case of the next mode, it denotes the octal equivalent of the binary bits that set the Next Mode register.

Table 4.4 lists the actual *octal* numbers that correspond to each operator mnemonic. In each case, right and left half words are separated by a comma. The convention has been chosen that a space *after* a designation denotes left justification within the half word. *No* space, or a space *before* a designation denotes right justification within the half word.

CONTROL BITS

Table 4.5 lists the bits in the Channel Control directive register. The function of each bit is briefly discussed in the notes that follow the table.

Table 4.6 lists the instruction register bits. Table 4.7 lists the bits in the command register. Tables 4.8 and 4.9 list the contents of the repeat status and status register bits.

I/O CONTROL

The LDS-1 system uses the DATAO instruction to force processor instructions and the CONO and CONI instructions of the PDP-10 for control. However, the DATAI instruction is not used.

Table 4.10 lists and briefly discusses the CONO instructions, while table 4.11 does the same for the CONI bits.

The DATAO instruction causes the processor to execute the instruction in the I/O bus if the processor is in the PAUSED state. The instruction is executed in the priviledged mode. The only affect this has is that the PERMIT bits of the clipper may be loaded in the priviledged mode. At the end of the instruction, the processor will go back to the paused state.

NB: If a count is specified, or repeat mode is entered, only one iteration will be performed before going back to the PAUSED state.

THE INSTRUCTIONS:

MNEMONIC	ARGUMENTS	INDIRECT BIT SET
GROUP Ø		
LI****	DP-REGISTER,DATA(NEXT MODE)	**
PSH*	DP-REGISTER,(NEXT MODE)	**
JMP***	ADDRESS(NEXT MODE)	NO JUMP
NWSTK*	ADDRESS(NEXT MODE)	NO SP-CHANGE
XQTA	ADDRESS	**
NOP		**
XQT		**
RPT	THE ADDRESS PART IS IGNORED	**
PROG		**
PEEL		**
STOP		**
GROUP 2		
LIF**	DP-REGISTER,DATA(CONDITION)	REVERSE DECISION
LAL**	DP-REGISTER,DATA(CONDITION)	**
JIF**	ADDRESS(CONDITION)	REVERSE DECISION
JAL**	ADDRESS(CONDITION)	NO JUMP
CL	(CONDITION)	**
ST	(CONDITION)	**
CM	(CONDITION)	**
IJ**CR	ADDRESS	REVERSE DECISION
GROUP 3		
LO****	CD-REGISTER,ADDRESS(N)	**
ST**	CD-REGISTER,ADDRESS(N)	**
RT****	CD-REGISTER,(N)	**
SK**	CD-REGISTER,(N)	REVERSE DECISION
GROUPS 4, 5, 6, 7		
SETPT*	ADDRESS-OF-DATA(NEXT MODE)	**
DRAW**	ADDRESS-OF-DATA(NEXT MODE)	**
LIN**	ADDRESS-OF-DATA(RPTM)	**
LINI**	ADDRESS-OF-POINTER(RPTM)	**
POL**	ADDRESS-OF-DATA(RPTM)	**
POLI**	ADDRESS-OF-POINTER(RPTM)	**
DOTS**	ADDRESS-OF-DATA(RPTM)	**
STAR**	ADDRESS-OF-DATA(RPTM)	**
BOX**	ADDRESS-OF-DATA	**

**If the indirect bit is set, the information in the address field of this instruction is not loaded into the processor. Memory address registers used will be incremented after use in the normal manner.

Table 4.1

CONSTRUCTION OF MNEMONICS FOR E & S DISPLAY CODES

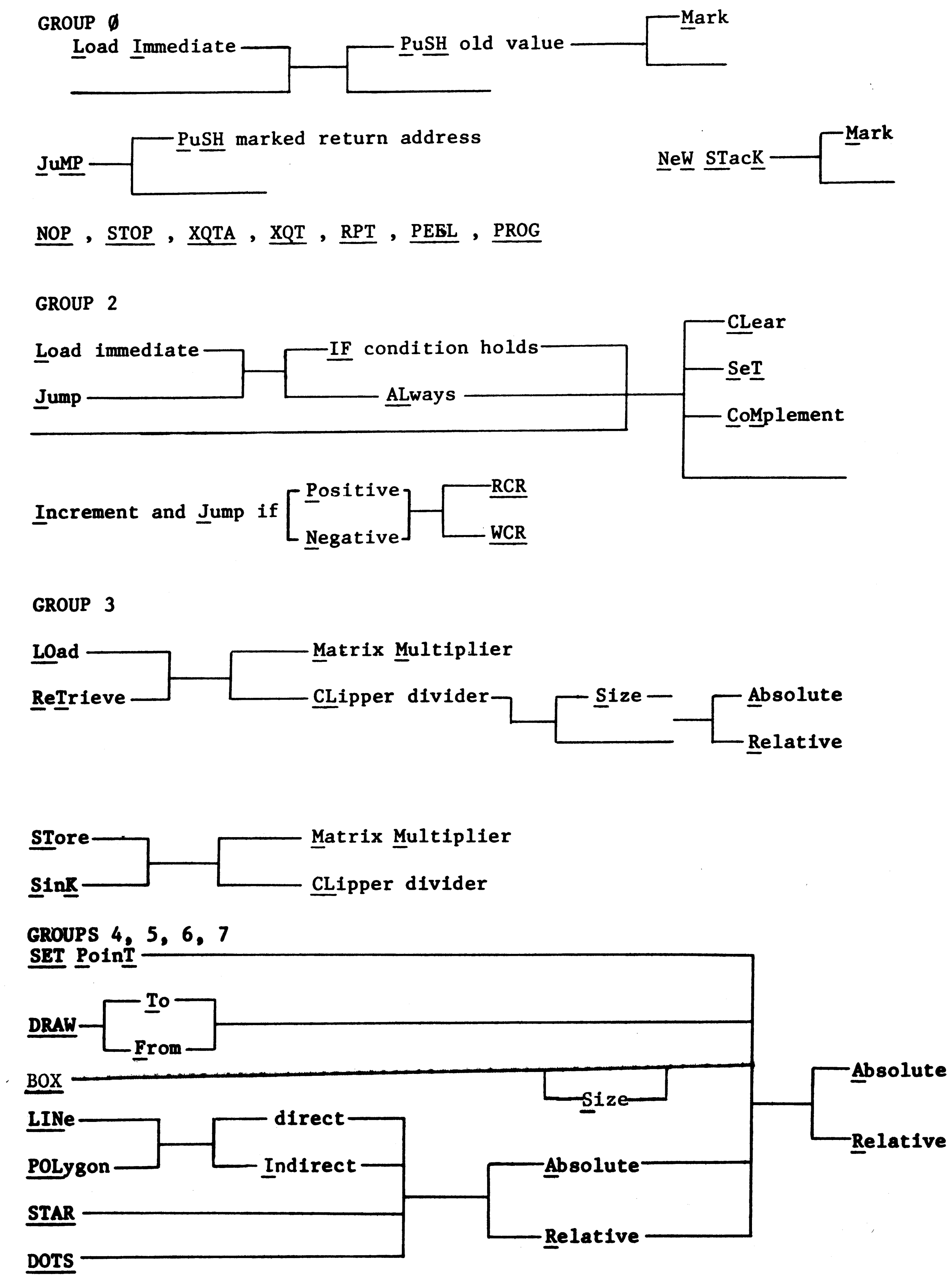


Table 4.2

DEFINITION OF ARGUMENTS FOR

THE E & S DISPLAY SYSTEM INSTRUCTIONS

THE DISPLAY PROCESSOR (DP-REGISTER):

RAR	=	Ø
WAR	=	1
PC	=	2
SP	=	3
P1	=	4
P2	=	5
DSP	=	6
UR	=	7
RCR	=	1Ø
WCR	=	11
DIR	=	12
RSR	=	13
SR	=	14

THE MODE (NEXT-MODE):

XQTM	=	1Ø
RPTM	=	4
PEELM	=	2
PROGM	=	1

CONDITIONS WHICH MAY BE CHECKED (CONDITION):

PFØ	=	Ø	Program Flag #Ø
PF1	=	1	Program Flag #1
PF2	=	2	Program Flag #2
PF3	=	3	Program Flag #3
RCRN	=	1Ø	RCR Less Than -1
WCRN	=	11	WCR Less Than -1
HITF	=	12	Hit Flag
AICF	=	13	Area In Common Flag
STOPF	=	17	Stop Flag

THE CLIPPER DIVIDER REGISTERS (CD-REGISTER):

SAVELB	=	Ø	Two Components	
SAVERT	=	1	Two Components	
VIEWLB	=	2	Two Components	
VIEWRT	=	3	Two Components	
WINDLB	=	4	Two Components	
WINDRT	=	5	Two Components	
INSTLB	=	6	Two Components	
INSTRT	=	7	Two Components	
NAMELB	=	1Ø	Two Components	
NAMERT	=	11	Two Components	
HITANG	=	12	Two Components,	Hit Count + Angle Count
SELINT	=	13	Two Components,	Select + Intensity
SAVE	=	14	Four Components	
VIEW	=	15	Four Components	
WIND	=	16	Four Components	
INST	=	17	Four Components	

Table 4.3

THE COMPLETE LIST OF INSTRUCTIONS

GROUP Ø - LOAD IMMEDIATE INSTRUCTIONS

LI	=	[ØØØ ,Ø]	Load Immediate
LIPSH	=	[Ø4Ø ,Ø]	Load Imm. Push-Old-Value
LIPSHM	=	[Ø6Ø ,Ø]	Load Imm. Push-Old-Value Marked
PSH	=	[LIPSH ,@Ø]	Push-Old-Value, Without Load
PSHM	=	[LIPSHM ,@Ø]	Push-Old-Value Marked, Without Loading
NOP	=	[LI ,@Ø]	
JMP	=	[LI PC,Ø]	Note: JMP For Display, JUMP for 1Ø
JMPPSH	=	[LIPSHM PC,Ø]	A Marked [JMP Ø] Is Saved In Stack
NWSTK	=	[LIPSH SP,Ø]	Unmarked [LI SP,Ø] Is Saved In Stack
NWSTKM	=	[LIPSHM SP,Ø]	A Marked [LI SP,Ø] Is Saved In Stack
XQTA	=	[LI RAR,(XQTM)]	Execute the Instruction in E
XQT	=	[NOP ,XQTM)]	Enter Execute Mode
RPT	=	[NOP ,(RPTM)]	Enter Repeat Mode
PEEL	=	[NOP ,(PEELM)]	Enter Peel Mode
PROG	=	[NOP ,(PROGM)]	Enter Program Mode

GROUP 2 - CONDITIONAL LOAD IMMEDIATE

LIF	=	[2ØØ ,Ø]	Load Immediate if Condition Holds
LIFCL	=	[21Ø ,Ø]	LIF And Clear
LIFST	=	[22Ø ,Ø]	LIF And Set
LIFCM	=	[23Ø ,Ø]	LIF And Complement
LAL	=	[24Ø .Ø]	Load Always, LAL Is Slower Than LI
LALCL	=	[25Ø ,Ø]	LAL And Clear
LALST	=	[26Ø ,Ø]	LAL And Set
LALCM	=	[27Ø ,Ø]	LAL And Complement
JIF	=	[LIF PC,]	Jump If Condition Holds
JIFCL	=	[LIFCL PC,]	JIF And Clear
JIFST	=	[LIFST PC,]	JIF And Set
JIFCM	=	[LIFCM PC,]	Jif And Complement
JAL	=	[LAL PC,]	Jump Always, JAL Is Slower Than JMP
JALCL	=	[LALCL PC,]	JAL And Clear
JALST	=	[LALST PC,]	JAL And Set
JALCM	=	[LALCM PC,]	JAL And Complement
JIFDED	=	[JIF ,(STOPF)]	JMP If Stopped
IJNRCR	=	[JIFST ,(RCRN)]	Increment RCR, JMP If Negative
IJNWCR	=	[JIFST ,(WCRN)]	Increment WCR, JMP If Negative
IJPRCR	=	[IJNRCR ,@0]	Increment RCR, JMP If Positive
IJPWCR	=	[IJNWCR ,@0]	Increment WCR, JMP If Positive
CL	=	[LALCL ,@Ø]	Clear Condition
ST	=	[LALST ,@Ø]	Set Condition
CM	=	[LALCM ,@Ø]	Complement Condition
STOP	=	[ST (STOPF)]	Stop The Processor

Table 4.4

THE COMPLETE LIST OF INSTRUCTIONS (cont'd)

GROUP 3 - EXTERNAL REGISTERS TRANSMISSIONS

LOCLA	=	[3ØØ ,Ø]	Load Clipper Absolute
LOCLR	=	[3Ø1 ,Ø]	Load Clipper Relative
LOCLSA	=	[3Ø2 ,Ø]	Load Clipper Size Absolute
LOCLSR	=	[3Ø4 ,Ø]	Load Clipper Size Relative
LOMM	=	[3Ø4 ,Ø]	Load Matrix Multiplier
STCL	=	[32Ø ,Ø]	Store Clipper Absolute
STMM	=	[324 ,Ø]	Store Matrix Multiplier
RTCLA	=	[34Ø ,Ø]	Retrieve Clipper Absolute
RTCLR	=	[341 ,Ø]	Retrieve Clipper Relative
RTCLSA	=	[342 ,@Ø]	Retrieve Clipper Size Absolute
RTCLSR	=	[343 ,@Ø]	Retrieve Clipper Size Relative
RTMM	=	[344 ,@Ø]	Retrieve Matrix Multiplier
SKCL	=	[36Ø ,@Ø]	Sink Clipper Absolute
SKMM	=	[364 ,@Ø]	Sink Matrix Multiplier

NOTE: 'Load SAVELB' , 'Store SAVELB' , 'Sink SAVELB' ,
But 'Retrieve NAMERT'

GROUP 4-7 - DISPLAY INSTRUCTIONS

DD	=	4ØØ ,Ø	Do Direct
DI	=	5ØØ ,Ø	Do Indirect
DN	=	6ØØ ,Ø	Do Internal

THE WHAT-TO-DO MACHINE:

LS	=	Ø6Ø ,Ø	Lines	=	(SET-DRAWTO)**
LT	=	Ø7Ø ,Ø	?????	=	(DRAWTO-SET)**
PO	=	Ø3Ø ,Ø	POLYG	=	SET-(DRAWTO)**
TO	=	Ø2Ø ,Ø	TO	=	(DRAWTO)**
SS	=	Ø4Ø ,Ø	STAR	=	SET-(DRAWFROM)**
FR	=	Ø5Ø ,Ø	FROM	=	(DRAWFROM)**
DT	=	Ø1Ø ,Ø	DOTS	=	(DOT)**
BX	=	ØØØ ,Ø	BOXES	=	(BOX)**

THE ABS/REL-MODES MACHINE:

RX	=	ØØ6 ,Ø	(REL-ABS)**
AX	=	ØØ7 ,Ø	(ABS-REL)**
RA	=	ØØ3 ,Ø	REL-(ABS)**
AB	=	ØØ2 ,Ø	(ABS)**
AR	=	ØØ4 ,Ø	ABS-(REL)**
RE	=	ØØ5 ,Ø	(REL)**
SL	=	ØØ1 ,Ø	(SIZE REL)**
SA	=	ØØØ ,Ø	(SIZE ABS)**

Table 4.4 (cont'd)

THE COMPLETE LIST OF INSTRUCTIONS (cont'd)

THE ABS/REL-MODES MACHINE (cont'd):

SETPTA	=	[DD+LS+AB]	Set-Point-Absolute
SETPTR	=	[DD+LS+RE]	Set-Point-Relative
DRAWTA	=	[DD+TO+AB]	Draw-To-Absolute
DRAWTR	=	[DD+TO+RE]	Draw-To-Relative
DRAWFA	=	[DD+FR+AB]	Draw-From-Absolute
DRAWFR	=	[DD+FR+RE]	Draw-From-Relative
LINAA	=	[DD+LS+AB]	Line-(Absolute-Absolute)
LINAR	=	[DD+LS+AX]	Line-(Absolute-Relative)
LINRA	=	[DD+LS+RX]	Line-(Relative-Absolute)
LINRR	=	[DD+LS+RE]	Line-(Relative-Relative)
LINIAA	=	[DI+LS+AB]	Line-Indirect-(Absolute-Absolute)
LINIAR	=	[DI+LS+AX]	Line-Indirect-(Absolute-Relative)
LINIRA	=	[DI+LS+RX]	Line-Indirect-(Relative-Absolute)
LINIRR	=	[DI+LS+RE]	Line-Indirect-(Relative-Relative)
POLAA	=	[DD+PO+AB]	Polygon-Absolute's
POLAR	=	[DD+PO+AR]	Polygon-Absolute-Relative's
POLRR	=	[DD+PO+RE]	Polygon-Relative's
POLRA	=	[DD+PO+RA]	Polygon-Relative-Absolute's (???)
POLIAA	=	[DI+PO+AB]	Polygon-Indirect-(Absolute's)
POLIAR	=	[DI+PO+AX]	Polygon-Indirect-(Absolute-Relative's)
POLIRR	=	[DI+PO+RE]	Polygon-Indirect-(Relative's)
POLIRA	=	[DI+PO+RX]	Polygon-Indirect-(Relative-Absolute's)
STARAA	=	[DD+SS+AB]	Star-Absolute's
STARAR	=	[DD+SS+AR]	Star-Absolute-Relative's
STARRR	=	[DD+SS+RE]	Star-Relative's
STARRA	=	[DD+SS+RA]	Star-Relative-Absolute's (???)
DOTSAA	=	[DD+SS+AB]	Dots-Absolute's
DOTSAR	=	[DD+DT+AR]	Dots-Absolute-Relative's
DOTSRR	=	[DD+DT+RE]	Dots-Relative's
DOTSRA	=	[DD+DT+RA]	Dots-Relative-Absolute's (???)
BOXA	=	[DD+BX+AB]	Box-Absolute
BOXR	=	[DD+BX+RE]	Box-Relative
BOXSA	=	[DD+BX+SA]	Box-Size-Absolute
BOXSR	=	[DD+BX+SL]	Box-Size-Relative

Table 4.4 (cont'd)

DIRECTIVE REGISTER BITS

18	Matrix Multiplier active (To do multiplication, etc.)
19	STOS (Clipper scaled output to scope)
20	STOM (Clipper scaled output to memory)
21	ZTOS (Clipper Z intensity to scope)
22	PTOM (Clipper clipped output to memory)
23	ATOM (Clipper name to memory)
24	3D (If not set, 2D is implied)
25	CURVE mode (Clipper and Matrix Multiplier)
26	Surface mode (Matrix Multiplier only)
27	MEF (Minimum Effort Mode)
28	SELFX (Clipper)
29	SELFY (Clipper)
30	DOTTED LINE
31	Matrix output to memory
32	Matrix output to Clipper
33	DO TWICE
34	Stop on WCR +
35	Stop on HIT

NOTES ON DIRECTIVE BITS

STOS indicates that the scaled output is to be sent to the scope. This bit should be set anytime that the scope is to be used.

STOM indicates that the scaled output is to be sent to memory.

ZTOS indicates that the depth queing is to be used and, after clipping the Z coordinate, is to be passed to the scope. This bit should be set when 3D information is being clipped for presentation to the scope.

PTOM indicates that the clipped page coordinates are to be sent back to memory. This bit does not prohibit the use of the scope. This is considered to be intermediate output from the Clipper.

ATOM indicates that the NAME REGISTER is to be shipped to the memory after the clipping has been done and a hit has been encountered.

3D directs the clipper and the processor to make two memory references for data in the form X,Y for the first word and Z_x,Z_y for the second word. The clipper then uses the three-dimensional algorithm for clipping.

CURVE puts the clipper into a mode such that lines within the negative Z extension of the cone of viewing will be displayed. This mode is useful for displaying complex curves.

MEF is the bit to direct the clipper to terminate its operation as soon as it finds a point of the line within the viewing area (window or cone of viewing).

Table 4.5

SELFX and SELFY are used for plotting mode. If SELFX is set, data is taken from memory as a normal operation, but only the Y data is considered to be valid. The X data is retrieved from the INSTANCE register, left and right (X) sections. SELFY has the same effect as SELFX except that the incoming data is valid in X only and the Y portions of the INSTANCE register are used for the Y data. The SELF bits imply a TO RELATIVE operation. See the description of DO TWICE for further explanation of plotting mode.

DOTTED LINE causes the display to go into a mode where all lines are dashed. This is performed by hardware in the scope itself and is not a function of the processor or the clipper.

DO TWICE causes the processor to enter a mode which raises the clipper's input flag twice for each memory data access. The second flag causes the SWAP directive bit to be raised for the clipper. This means that the X and Y data to the clipper has been swapped. DO TWICE operations allow packing of data for the plotting mode since only half the data is normally used. If SELFY and DO TWICE are set, the data is to be prepared as follows:

X_1, X_2

X_3, X_4

etc.

If SELFX and DO TWICE are set, the data must appear as:

Y_2, Y_1

Y_4, Y_3

etc.

The reason for this is that data is *always* taken normally the first time and reversed the *second* time.

The two low-order bits of the directive register are used as "mask" bits and, if set to one, will cause the processor to go into a stop state if the WCR goes positive or if the HIT bit gets set by the clipper. The two bits are mutually independent.

All other bits have no effect on a system without a matrix multiplier.

Table 4.5 (cont'd)

INSTRUCTION REGISTER BITS

0-2	Instruction group		
		0	Load immediate (May push, mark, change mode)
		1	Unused
		2	Conditional load immediate
		3	Multiple register load, store, sink, retrieve
		4	Load immediate and direct display data
		5	Load immediate and indirect display data
		6	Internal data
		7	Unused
IN 0	3	Push	
	4	Mark	
IN 2	3	Always	
	4	"J" after test	
	5	"K" after test	
IN 3	3-4	Load (00), store (01), retrieve (10), sink (11)	
	5-8	Device and manner	
IN 4-6	3-8	Finite state machine	
	9-12	Which processor register	
	13	"Indirect" (Inhibit load immediate, reverse testing)	
	14-17	Next mode or condition number	
		14	XCT
		15	RPT
		16	PEEL
		17	PROG

Table 4.6

COMMAND REGISTER BITS

0	Fetch/load matrix multiplier
1	Fetch/load clipper
2-3	Line,setpoint,box,dot
4-5	Retrieve (00), load (01), sink(10), store (11)
6	Size
7	Relative
8	From
9-12	Register address
13	Priviledged
14-17	Missing
18-33	Copies of the directive register

Table 4.7

REPEAT STATUS REGISTER BITS

18-20	Instruction type (copy of IR 0-2)
21-23	FSM for setpoint/endpoint (copy of IR 3-5)
24-26	FSM for absolute/relative (copy of IR 6-8)
27-30	Which device register counter (copy of IR 9-12)
31	(missing) (copy of IR 13)
32-35	Short count for multiple load/store (copy of IR 14-17)

Table 4.8

STATUS REGISTER BITS

18-21	Program flags (test 0 - test 3)
22-25	(missing) (test 4 - test 7)
26	Read count register sign (test 10)
27	Write count register sign (test 11) Read and write counter signs cannot be loaded "J" condition increments the counter
28	Hit bit (test 12)
29	AIC bit (area in common) (test 13)
30-32	(missing) (test 14 - test 16)
33	Program stop flag (test 17)
34-35	(missing) (not testable)

Table 4.9

CONO BITS

18 - System clear. This has the same effect as the console I/O reset switch and the clear switches on the Clipping Divider and Channel Control. The clipper and processor are cleared and the processor is sent to the STOP state while the clipper is sent to the INPUT WAIT state. Two successive clears are necessary if the clipper has been operating in the 3D mode. The clipper will not finish the line it is working on, nor will the processor complete its instruction. No information is lost in the processor.

19 - Allow Memory Alarm Interrupt. This allows non-existent memory and parity errors to cause the host computer to interrupt on the selected channel.

20 - Disallow Memory Alarm Interrupt.

21 - ~~Unused.~~ MAP/NOT MAP (PROTECTION / RELOCATION)

22 - Allow Map/Protect Interrupt. This bit is used in connection with memory protection.

23 - Disallow Map/Protect Interrupt.

24 - Set I/O Stop.

25 - Allow Stop Interrupt.

26 - Disallow Stop Interrupt.

27 - Clear I/O Stop.

28 - Clear Program Stop.

29 - Clear Hit.

30 - Step.

31 - Unused.

32 - Allow Priority Interrupt Assignment.

33, 34, 35 - Priority Interrupt Assignment.

Table 4.10

CONI BITS

0 -	Program Flag 0
1 -	Program Flag 1
2 -	Program Flag 2
3 -	Program Flag 3
4-17 -	Unused
18 -	Parity Alarm
19 -	NXM Alarm (non-existent memory)
20 -	Alarm Interrupt On
21 -	Map/Protect Violation
22 -	Map/Protect Interrupt On
23 -	Unused
24 -	Stopped and Ready
25 -	Stop Interrupt On
26 -	Memory To Memory Stop
27 -	I/O Stop
28 -	Program Stop
29 -	Hit Stop
30 -	Scope Select Violation Stop
31 -	Unused
32 -	LDS-1 Caused Interrupt (i.e. Interrupt has actually occurred)
33, 34, 35 -	Priority Interrupt Assignment

Table 4.11

V. PROGRAMMING EXAMPLES

INTRODUCTION

This chapter is devoted entirely to examples of code for the LDS-1 system. A prior knowledge of the PDP-10 assembly language is assumed. The code shown in these examples is not guaranteed to be the optimal code for accomplishing a task, but is presented to illustrate the use of certain features of LDS-1.

START UP

The first example is a subroutine which initializes the system and starts the processor at a specified location. This subroutine is employed in nearly all the examples, so it behooves the reader to become familiar with the functions performed even if he doesn't care to understand it in detail. This subroutine is included in the MACRO-10 file which defines the operation codes for LDS-1.

```
        .TITLE STARTU
        /START UP SUBROUTINE TO INITIALIZE PROCESSOR
DP=L3Ø
DPPI=4
SINK:   BLOCK 177
STACK:  BLOCK 1
OPDEF CONODP <CONO DP,DPPI>
INDISP:         Ø
        CONODP  515300
        MOVE    Ø,<JSR DPINTR>
        MOVEM   Ø,4Ø+2*DPPI
        CONSO   DP,4ØØØ
        JRST    .-1
        DATAO   DP,<LI DIR,2ØØØØØ(PRGM)>
        CONSO   DP,4ØØØ
        JRST    .-1
        DATAO   DP,<LI SR,O>
        CONSO   DF,4ØØØ
        JRST    .-1
        DATAO   DP,<LOCLA SELINT,<XWD 4ØØ4ØØ,77ØØØØØ>(1)>
        HRRZ    Ø,(16)
        ADD     Ø,<JMP Ø>
        MOVEM   Ø,DSTQQQ+4
        CONSO   DP,4ØØØ
        JRST    .-1
        DATAO   D/,<JMP DSTQQQ>
        CONODP  4ØØ
        JRA     16,1(16)
DSTQQQ:         LI      SP,STACK
        LI      DSP,SINK
        LOCLSA  VIEW,<XWD 3777,3777>(1)
        LOCLSA  WIND,<XWD 1ØØØ,1ØØØ>(1)
        STOP
```

The LDS-1 system is set up as unit 130 on the PDP-10 and is assigned priority interrupt level 4 by the assignment DPPI = 4. A combination stack and sink are supplied. Remember that the stack is loaded toward lower addresses and the sink toward higher addresses. A new .OPDEF is given for the CONO instruction. CONODP is a CONO directed toward the LDS-1 with the priority interrupt level always supplied.

The first instruction to the system is via the CONO. The system is master cleared, all interrupts turned off, and pause request issued. A JSR DPINTR is planted in the interrupt location. (DPINTR is a minimal interrupt routine which is supplied with INDISP, but not explained here). The CONSO loop causes the PDP-10 to wait until the LDS-1 system has returned to the PAUSE state. Then all conditions which might cause the system to stop are cleared. These conditions must be cleared via the DATAO instruction because the system will return to the STOP (or PAUSE) state if all stop conditions have not been cleared. Therefore, a sequence of instructions from memory would not necessarily clear all stop conditions. One must wait for the system to return to the PAUSE state before issuing another DATAO instruction. (NB, The DATAO instruction has no effect on the system if the LDS-1 is not in a PAUSE state.)

The stop conditions and means of clearing each are listed below:

STOP on WCR +	Clear directive bit mask
STOP on HIT	Clear directive bit mask
PROGRAM STOP	Clear status register bit
Scope Selection Violation	Load proper permit and select bits (this can be done only by a DATAO)

After clearing all stop conditions, a JMP POOH is placed at the end of the setup code located at DSTQQQ and the system is given the command to load the PC with DSTQQQ. Then the resume pulse is issued via a CONO. This causes the pause and stop flip flop to be cleared. (NB, The Stop flip flop may be cleared for the execution of one instruction, but if a stop condition still exists, it will be set again. Therefore, issuing RESUME does not clear the condition, only the Stop flip flop.) The system then starts executing the code located at DSTQQQ. The first two instructions load the stack and sink pointers. The viewport is loaded to give full screen deflection and the window is loaded with the arbitrary numbers -1000, +1000 for both X and Y. Note the use of size absolute in conjunction with a 72-bit load. The plated JMP instruction is then executed which starts the user program.

Included with the INDISP subroutine is the following macro definition:

```
DEFINE DSTART (POOH)
 <JSA 16,INDISP
  JUMP POOH>
```

This macro will be used throughout the examples.

2D PICTURE

This will display a star whose center is conrolled via the switches and load the hit and angle counters in the console lights. The counts are cleared each time through the loop before any drawing instructions are issued. The setpoint data is controlled by the console switches. The RCR is used to control the number of THRURR type instructions executed in repeat mode. The RAR is loaded with the address of CNTR2D via the SETPTA instruction and will be incremented by SETPTA. Therefore, it is not necessary to load RAR in the THRURR instruction since the data is located immediately after CNTR2D.

```
                         TEST 2D
OPDEF   THRURR<DD+TO+SL>
TEST2D:         DSTART(SHOW2D)
        RSW     CNTR2D
        DATAO   PI,STACK
        JRST    TEST2D+2
SHOW2D:         LI       RCR,-4
        LOCLA   HITANG,<Ø>(1)
        SETPTA  CNTR2D
        THRURR  @(RPTM)
        STCL    HITANG,STACK(1)
        JMP     SHOW2D
CONTR2D:        Ø
        XWD     6ØØ,6ØØ
        XWD     6ØØ,Ø
        XWD     6ØØ,-6ØØ
>       XWD     Ø,-6ØØ
```

```
                              TEST 3S (SQUARES IN 3D SPACE)
TEST3S:         DSTART(SHOW3S)
        RSW     Ø
        AND     Ø,<77>
        MOVN    Ø,Ø
        HRRM    Ø,CNT3S
        RSW     INPT3S
        BITSM(INPT3S,<Ø>,<16>,X3S)
        BITSM(INPT3S,<17>,<35>,Y3S)
        COMBIN(X3S,Y3S,XY3S)
        JRST    TEST3S+2
;
X3S:    Ø
Y3S:    Ø
INPT3S:         Ø
;
SHOW3S:         LI      DIR,244ØØØ
        SETPTA  PNT3S
CNT3S:  LI      RCR,-2
        LIPSHM  RCR,-4
        DRAWTR  TAB3S(RPTM)
        SETPTR  @(PEELM)
        IJNRCR  CNT3S+1
        JMP     SHOW3S+1
;
PNT3S:  XWD     4ØØ,4ØØ
        XWD     4ØØ,4ØØ
TAB3S   XWD     -1ØØØ,Ø
        XWD     Ø,Ø
        XWD     Ø,-1ØØØ
        XWD     Ø,Ø
        XWD     1ØØØ,Ø
        XWD     Ø,Ø
        XWD     Ø,1ØØØ
        XWD     Ø,Ø
XY3S:   XWD     Ø,Ø
        XWD     4Ø,4Ø
;
        /BITSM AND BITS SUBROUTINES
DEFINE BITSM(A1,A2,A3,A4)
<JSA 16,BITS
        LI      A1
        LI      A2
        LI      A3
        MOVEM   Ø,A4>
;
        ENTRY BITS      ; BITS(X,M,N) GETS BITS M-N OF X
        Ø
BITS:   Ø
        MOVEM   1,BITS-1
        MOVE    Ø,@Ø(16)        ; X TO ACØ
        MOVE    1,@1(16)        ; LSH ACO M BITS
        LSH     Ø,Ø(1)
        MOVE    1,@2(16)        ;AC1 = N
        SUB     1,@1(16)        ;AC1 = N-M
        SUBI    1,↑D35          ;AC1 = N-M-35
        ASH     Ø,Ø(1)          ;RIGHT LSH (35-M+N) BITS
        MOVE    1,BITS-1
        JRA     16,3(16)
>
```

3D PICTURE

This will display a set of parallel squares equally spaced along the Z axis. The X and Y starting point for the first square and the number of squares are controlled by the switches. Note that the directive must be loaded for 3D operation, and ZTOS must be selected in addition to OUTPUT TO SCOPE. The RCR is used for two purposes in this example. First, it is loaded with the negative of the number of squares requested. The RCR is then Pushed Marked and loaded with -4 to control repeat mode. Note that in 3D it is necessary to access two words for each data point, but the RCR refers to data points, not PDP-10 words. After the square is drawn with the DRAWTR instruction, a SETPTR is issued (without reloading the RAR) which moves Z out 40 units and goes into PEEL mode. PEEL mode causes the stacked value of the RCR to be unstacked and, since the PUSH instruction was marked, the mode is changed back to PROGRAM. The IJNRCR causes the RCR to be incremented and the PC to be loaded with CNT3S+1 until the RCR goes to -1. (The RCR is tested before it is incremented.)

CHESSB (SHOWS CHESS BOARD)

```
;
CHESS: DSTART(SCHESS)
       HALT
SCHESS:        LOCLA   WINDLB,DCHESS(2)        ;LOAD WINDOW
       ST      (PFØ)            ;SET PROGRAM FLAG Ø
       SETPTA  <XWD Ø,Ø>
       LI      RCR,-1Ø          ;SET COLUMNS COUNT
CHESS1:        LIPSHM  RCR,-1Ø  ;SET ROWS COUNT
CHESS2:        JMPPSH  SQUARE
       SETPTR  <XWD Ø,1ØØ>      ;ADVANCE TO NEXT LINE
       IJNRCR  CHESS2           ;MORE LINES?
       SETPTR  <XWD 1ØØ,-1ØØØ>  ;ADVANCE TO NEXT COLUMN
       CM      (PFØ)            ;CHANGE CROSS-HATCHING PARITY
       PEEL                     ; RETRIEVE COLUMN COUNT
       IJNRCR  CHESS1           ;MORE COLUMNS
       JMP     SCHESS+1         ;START ALL OVER AGAIN
;
SQUARE:        LIPSH   RCR,-5
       POLRR   TABCHS(RPTM)     ;DRAW A SQUARE
       SETPTR  @
       JIFCM   .+2(PFØ)         ;CHECK CROSS-HATCHING PARITY
       PEEL
       LI      RCR,-13
       LINRR   @(RPTM)          ;CROSS-HATCHING
       PEEL
;
DCHESS:        XWD     Ø,Ø      ;WINDOW SIZE
       XWD     1ØØØ,1ØØØ
;
TABCHS:        XWD     1Ø,1Ø    ;DATA FOR SQUARES
       XWD     6Ø,Ø
       XWD     Ø,6Ø
       XWD     -6Ø,Ø
       XWD     Ø,-6Ø
       XWD     -1Ø,-1Ø
;
       XWD     1Ø,5Ø            ;DATA FOR CROSS-HATCHING
       XWD     2Ø,2Ø
       XWD     -2Ø,-4Ø
       XWD     4Ø,4Ø
       XWD     -4Ø,-6Ø
       XWD     6Ø,6Ø
       XWD     -4Ø,-6Ø
       XWD     4Ø,4Ø
       XWD     -2Ø,-4Ø
       XWD     2Ø,2Ø
       XWD     -7Ø,-3Ø
END OF FILE REACHED BY:
P 6Ø
>
```

CHESS BOARD

This displays a chess board on the screen. Cross hatching of the squares is controlled by program flag 0. Note the use of a processor subroutine (SQUARE). When exit from the subroutine is effected by the PEEL instruction, the RCR is also peeled since it was pushed, unmarked, when the subroutine was entered. Cross hatching is carried out in the SQUARE subroutine by testing PF0.

The RCR is used to control the column count, the row count, and the count for two repeat mode instructions. This multiple use of the RCR is made possible by correct use of the stack.

```
;
DOTSLF: DSTART(SHOWSF)
        HALT .
;
SHOWSF: LI       DIR,200000
        LOCLA    VIEWLB,VIEW1(6)
        SETPTA [XWD 100,4000]
        LI DIR,200200    ;SELF X
        LI RCR,-140
        DOTSRR [XWD 0,10]
        IJNRCR .-1
        LI RCR,-300
        DOTSRR [XWD 0,-10]
        IJNRCR .-1
        LI RCR,-140
        DOTSRR [XWD 0,10]
        IJNRCR .-1
        LI       DIR,200000
        LOCLA VIEWLB,VIEW2(6)
        SETPTA [XWD 4000,10000]
        LI DIR,200104    ;SELF Y + SWAP
        LI RCR,-20


        DOTSRR [XWD 20,60]
        IJNRCR .-1
        LI RCR,-40
        DOTSRR [XWD -20,-60]
        IJNRCR  .-1
        LI      RCR,-20
        DOTSRR  [XWD 20,60]
        IJNRCR  .-1
        JMP     SHOWSF
;
;
VIEW1:  XWD     -3777,0
        XWD     0,3777
        XWD     0,0
        XWD     10000,10000
        XWD     10,0
        XWD     10,0
VIEW2:  XWD     0,-3777
        XWD     3777,0
        XWD     0,0
        XWD     10000,10000
        XWD     0,-40
        XWD     0,-40
;
;
```

SELF MODE

This demonstrates the plotting (SELF) mode. Two graphs are plotted; one in the upper left of the screen, and one in the lower right.

The upper left display uses SELFX; the lower right display uses SELFY and SWAP. SELFX is not given until after the viewport, window, and instance have been loaded and the starting point has been set. Then the DOTSRR instruction is used within three loops controlled by the RCR. After drawing the graph, SELFX is cleared. When SELFX is being used, the X data is always 10, since 10 is loaded into the X portion of the instance register.

The lower right graph uses SELFY and SWAP. The instance is loaded with -40 in the Y portion for the SELFY data. This time both an X and Y are given in the DOTSRR instruction. Normally, the Y would be ignored, but since SWAP is set, the normal X data (20 or -20) is used for X, then the normal Y data (60 or -60) is used for X.

```
BOX2D:  DSTART(SUB2D)
BXLOOP: RSW     14
        MOVEI   10,WS
        JSA     16,FIX
        HRRZ    15,WS
        LSH     15,1
        HRRM    15,WS+3
        XOR     15,[XWD 0,-1]
        HRRM    15,WS+1
        HLRZ    15,WS
        LSH     15,1
        HRLM    15,WS+4
        XOR     15,[XWD 0,-1]
        HRLM    15,WS+2
        LSH     14,-6
        MOVEI   10,WC
        JSA     16,FIX
        LSH     14,-6
        MOVEI   10,LSZ
        JSA     16,FIX
        LSH     14,-6
        MOVEI   10,LC
        JSA     16,FIX
        MOVEI   15,100
        SOJGE   15,,
        JRST    BXLOOP



;
;
;
FIX:    0
        TRNN    14,1
        JRST    XM
        HRLZI   15,1
        ADDM    15,0(10)
XM:     TRNN    14,2
        JRST    YP
        HRLZI   15,-1
        ADDM    15,0(10)
YP:     TRNN    14,4
        JRST    YM
        HRRZI   15,1
        ADD     15,0(10)
        AND     15,[XWD 0,-1]
        HRRM    15,0(10)
YM:     TRNN    14,10
        JRA     16,0(16)
        HRRZI   15,-1
        ADD     15,0(10)
        AND     15,[XWD 0,-1]
        HRRM    15,0(10)
        JRA     16,0(16)
```

```
;
OPDEF LISR[DD+TO+SL]
;
SUB2D:  LOCLA   VIEWLB,VIEW11(4)
        SETPTA  WC
        LI      RCR,-5
        POLRR   @(RPTM)
        SETPTA  LC
        LISR    @Ø
        SETPTA  WC
        LOCLSR  @WIND,(1)
        LOCLA   VIEWLB,VIEW21(2)
        SETPTA  LC
        LOCLSR  @INST,(1)
        JIF     @NOBOX(AICF)
        SKCL    VIEWLB,(4)
        BOXSA   MASTER
        SETPTA  @
        LI      RCR,-2Ø
        LINRR   @(RPTM)
        RTCLA   WINDRT,(4)
NOBOX:  SETPTA  LC
        LISR    @Ø
        JMP     SUB2D

MASTER: XWD     2ØØØ,2ØØØ
        XWD     -2ØØØ,Ø
        XWD     Ø,Ø
        XWD     2ØØØ,Ø
        XWD     Ø,1ØØØ
        XWD     Ø,-2ØØØ
        XWD     Ø,14ØØ
        XWD     1ØØØ,4ØØ
        XWD     Ø,Ø
        XWD     Ø,1ØØØ
        XWD     Ø,-4ØØØ
        XWD     Ø,1ØØØ
        XWD     Ø,Ø
        XWD     -1ØØØ,4ØØ
        XWD     Ø,Ø
        XWD     1ØØ,Ø
        XWD     -1ØØ,Ø
        XWD     7Ø,-7Ø
        BLOCK   2Ø
VIEW11: XWD     -3777,Ø
        XWD     Ø,3777
        XWD     4ØØØØØ,4ØØØØØ
        XWD     377777,377777
VIEW21: XWD     Ø,-3777
        XWD     3777,Ø
WC:     XWD     Ø,Ø
WS:     XWD     4ØØØØ,4ØØØØ
        XWD     Ø,-1ØØØØØ
        XWD     -1ØØØØØ,Ø
        XWD     Ø,1ØØØØØ
        XWD     1ØØØØØ,Ø
LC:     XWD     Ø,Ø
LSZ:    XWD     4ØØØ,4ØØØ
```

BOXING

This produces two pictures, the first is in the upper left which displays a window and a line representing the diagonal of the instance. The other is in the lower right and displays that part of the instance (a transistor) within the window plus the diagonal line. The PDP-10 portion of the program changes the size and position of the window and instance.

After the code for the upper left picture has been executed, the window and viewport are changed and the instance loaded with a 72-bit load (this is necessary to test Area In Common). AIC is tested with a group 2 instruction and if the instance has no area in common with the window, control is passed to NOBOX which displays the diagonal of the instance.

If there is AIC, the window and viewport are "sinked." The BOXSA command is given with the RAR set to point to MASTER. Since the box command is given as size absolute and MASTER contains 2000, 2000, the master definition space is -2000 to 2000 in X and Y. The box command causes the viewport and window to be changed to reflect the mapping directly from master space to scope coordinates. The commands are then issued to draw the transistor. The old viewport and window are then retrieved from the data stack.

```
TESTSW: DSTART(USER1)
        JSA      16,WAIT
        JSA      16,SAVV
        LI       SAVVU1
        DSTART(USER2)
LOOPSW: DATAO    PI,[2]
        JSA      16,WAIT
        JSA      16,SAVV
        LI       SAVVU2
        JSA      16,RESTOR
        LI       SAVVU1
        DATAO    PI,[1]
        JSA      16,WAIT
        JSA      16,SAVV
        LI       SAVVU1
        JSA      16,RESTOR
        LI       SAVVU2
        JRST     LOOPSW
WAIT:   Ø
        RSW
        AND      Ø,[7777]
WAIT1:  CONO     APR,1ØØØ
        CONSO    APR,1ØØØ
        JRST     .-1
        SOSLE
        JRST     WAIT1
        JRA      16,(16)

USER1:  LOCLA    VIEWLB,VWU1(4)
        LI       RCR,-5
        POLAR    TABSW(RPTM)
        JMP      USER1+1
USER2:  LOCLA    VIEWLB,VWU2(4)
        LI       RCR,-5
        POLAR    TABSW(RPTM)
        JMP      USER2+1
TABSW:  XWD      1ØØ,1ØØ
        XWD      -2ØØ,Ø
        XWD      Ø,-2ØØ
        XWD      2ØØ,Ø
        XWD      Ø,2ØØ
VWU1:   XWD      -3777,Ø
        XWD      3777,3777
        XWD      -2ØØ,-2ØØ
        XWD      2ØØ,2ØØ
VWU2:   XWD      -3777,-3777
        XWD      3777,Ø
        XWD      -1ØØ,-1ØØ
        XWD      1ØØ,1ØØ
        BLOCK    14
SAVVU1: BLOCK    14
        BLOCK    14
SAVVU2: BLOCK    14
```

```
;
; A SAVED-USER-TABLE ('USERSV')
;

;       -14   LI   DSP,(PROGM)   ; AFTER SINKING THE CD
;       -13   LI   RSR,
;       -12   LI   DIR,
;       -11   LI   WCR,
;       -10   LI   RCR,
;        -7   LI   UR,



;        -6   LI   RAR,
;        -5   LI   WAR,
;        -4   LI   DSP,(PROGM)
;        -3   LI   PC,(PROGM)
;        -2   LI   SR,(PROGM)
;        -1   LI   SP,(PROGM)
; USERSV:     BLOCK 14 ; FOR THE CD
;
;
DEFINE IOMBR(A1)
<DATAO DP,A1
        CONSO    DP,4000
        JRST     .-1>

SAVV:   0
        MOVE     0,@0(16)
        HRRM     0,SAVV1
        HRRM     0,SAVV4
        CONO     DP,4000
        CONSO    DP,4000
        JRST     .-1
        IOMBR(SAVV1)
        IOMBR(SAVV2)
        IOMBR(SAVV3)
        CONO     DP,40000 ; RESUME
        CONSO    DP,40000
        JRST     .-1
        JRA      16,(16)
SAVV1:  NWSTKM   0          ; 'USERSV' PLANTED HERE
SAVV2:  LIPSHM   SR,0
SAVV3:  JMPPSH   SAVV4
SAVV4:  LIPSHM   DSP,0      ; 'USERSV' PLANTED HERE
        PSH      WAR,
        PSH      RAR,
        PSH      UR,
        PSH      RCR,
        PSH      WCR,
        PSH      DIR,
        PSH      RSR,
        SKCL     SAVELB,(14)
        PSHM     DSP,
        STOP
```

```
RESTOR: Ø
        MOVE     Ø,@Ø(16)
        SUBI     Ø,14
        HRRM     Ø,RESTR1
        CONO     DP,4ØØØ
        CONSO    DP,4ØØØ
        JRST     .-1
        IOMBR(RESTR2)
        IOMBR([JMP RESTR1])
        CONO     DP,4ØØØØ
        CONSO    DP,4ØØØØ
        JRST     .-1
        CONO     DP,4ØØØ
        CONSO    DP,4ØØØ
        JRST     .-1
        IOMBR([PEEL])
        IOMBR([PEEL])
        IOMBR([PEEL])
        CONO     DP,4ØØØØ
        JRA      16,(16)
RESTR1: LI       SP,Ø(PEELM)        ; 'USERSV'-14 PLANTED HERE
        RTCLA    SELINT,(14)
        PEEL
        STOP
RESTR2: LI       SR,Ø
;
```

TIMESHARING

This program swaps two users on the same scope. The switches control the time that each user has the scope. The console lights show which user has the scope by displaying one or two.

Subroutine WAIT is the timeout subroutine.

SAVV is the subroutine to save a user in an area specified in the calling sequence. First a pause request is issued, then, via a DATAO, a new stack is formed and the status register and PC are both pushed marked. Then the processor is given control to execute the code at SAVV4 which saves all other registers plus the registers of the clipper. Note that the data stack pointer is pushed marked and then pushed marked again after saving the clipper registers.

RESTOR is the routine to restore a user from a saved area. First the status register is cleared to clear program stop which was set at the end of the SAVV routine. Then the stack pointer is set to the top of the stack for the saved user and PEEL mode entered which causes the data stack pointer to be unstacked (the DSP was the last thing pushed in SAVV and it was pushed marked). Then all registers of the clipper are retrieved via the restored DSP. PEEL mode is entered again which restores all registers but the PC, SR and SP. These three registers get restored via DATAO instruction and the resume is issued.

The examples were taken from the acceptance tests. These examples were chosen for inclusion with comments here because they demonstrated many of the features of the LDS-1 system. Certainly not all features are included here. For further examples (without the benefit of complete comments), see the acceptance tests.

Appendix I: CLIPPING DIVIDER PROCESSING TIME.

The clipping divider is a partly synchronous and partly asynchronous device. While it is in an input or output waiting state, or while it is engaged in passing data, it behaves as an asynchronous device and its internal clock is not running. Thus initiating a computation and passing the data relevant to the computation takes very little time. When a computation starts, the clock is set running and the computation steps coincide with clock times. The period of the clock is 0.5 usec. On a 'best effort' basis we are attempting to operate the clock considerably faster than this, probably in the 0.25 usec region.

Clipping divider processing of dots and lines starts with 5 clock times of 'setup'. Center size specification and processing dots requires one additional clock time during the 'setup' phase. Processing then proceeds to the 'clipping' step, in which any portion of the picture outside the window is eliminated. Dots require only a single clock time in this phase of the computation. The time required for a line depends on its position relative to the window. If both ends of a line are within the window, the 'clipping' phase is completed in 1 clock time. Otherwise the processing time is from 1 to 20 clock periods, tending toward smaller times if the line is entirely outside the window. The maximum clipping time is $[\log_2 L]$ clock periods where L is the larger of the X and Y line lengths. If the dot or any segment of the line was within the window, the processing then proceeds to the perspective division and the window-to-viewport mapping. The processing time here, following 1 'setup' step, is from 1 to 20 clock times depending upon the size of the window and the viewport. The maximum processing time will be $[\log_2 W]$ clock periods where 'W' is the larger of the dimensions of the viewport or the larger of the dimensions of the window, whichever is smaller. The worst case for processing lines is 46 clock times (for 20 bit numbers) or 42 clock times (for 18 bit numbers) and the best time for rejecting a line is 6 clock times. Sending to memory requires several clock times, sending to the scope requires from 1 to 2 clock times, and sending depth cueing information to the scope requires several clock times.

After the usual 'setup' operations, boxing requires 2 clock times for determining that the window and instance have no area in common, or from 12 to 31 steps to compute a new window and viewport. Loading and unloading parameters requires from 4 to 6 clock times.