

SEL EXECUTIVE SYSTEM MANUAL

James F. Blinn

01-17-72

## TABLE OF CONTENTS

1.	INTRODUCTION .....	1
2.	BASIC SYSTEM FACILITIES .....	2
2.1	Hardware .....	2
2.2	Software .....	2
2.2.1	The SEL Executive System .....	2
2.2.2	User Programs .....	3
3.	SYSTEM OPERATION .....	4
3.1	Loading the System .....	4
3.2	System Initialization and Error Comments .....	4
4.	COMMAND LANGUAGE INTERPRETER .....	5
4.1	CLEAR Command .....	5
4.2	FROM Commnad .....	5
4.2.1	Source Devices .....	5
4.2.2	Sink Devices .....	9
4.3	RUN Command .....	11
5.	DISPLAY CONTROL PROGRAMS .....	12
5.1	Root Node .....	12
5.2	Leaves .....	12
5.3	Branches .....	13
5.4	Nodes .....	16
5.5	Storage Block Format .....	16
5.6	Example Program .....	17
6.	THE LIGHT PEN .....	20
6.1	Light Pen Hits .....	20
6.2	Tracking .....	20
7.	SYSTEM SUBROUTINES .....	21
7.1	Calling Sequences .....	21

## SEL Executive System Manual

7.2 Task Scheduling Routines (#T...)	22
7.3 Buffered I/O Routines (#B...)	23
7.4 Text List Subroutine (#L)	25
7.5 Non-Buffered I/O Routines (#N...)	26
7.6 Core Allocation Routines (#C...)	26
7.7 Display Structure Operations (#S...)	27
7.8 Display State Sensing Routines (#D...)	29
7.9 Tracking Routines (#X...)	31
Appendix A: PDP-9 INSTRUCTION SET	33
A.1 Operate Instructions	33
A.2 Memory Reference Instructions	36
A.2.1 Addressing	36
A.2.1.1 Direct Addressing	36
A.2.1.2 Indirect Addressing	37
A.2.1.3 Autoindexing	37
A.2.2 Instructions	38
A.3 EAE Instructions	39
Appendix B: 339 DISPLAY CONTROL INSTRUCTIONS	43
B.1 Control State Instructions	43
B.1.1 Parameter (octal 0xxx)	43
B.1.2 Mode (octal 1xxx)	44
B.1.3 Jump (octal 2xxx)	45
B.1.4 Pop (octal 3xxx)	45
B.2 Data State Formats	45
B.2.1 VEC mode	45
B.2.2 SVEC mode	46
B.2.3 INCR mode	46

Appendix C: PDP-9 ASSEMBLY LANGUAGE .....	47
C.1 Source Language Syntax .....	47
C.1.1 Comments .....	47
C.1.2 Constants .....	47
C.1.3 Fields .....	47
C.2 Errors .....	48
C.3 Assembler Pseudo-Operations .....	48

## 1. INTRODUCTION

This manual describes the SEL Executive System for the 339 computer display terminal and provides the necessary information for writing user programs to run under it. The system was designed primarily to support user programs which communicate between the operator and the 339 via graphical interaction and between the 339 and MTS via a 201 dataset and the Data Concentrator. It provides multiprogramming, I/O buffering, core allocation and display support. The hardware configuration of the terminal consists of the following items:

- DEC PDP-9 with two 8192-word memory banks
- DEC KE09A extended arithmetic element
- DEC 339 display control
- DEC AF01B A/D convertor
- DEC AA01A D/A convertor
- AT+T 201A Data Set

## 2. BASIC SYSTEM FACILITIES

### 2.1 Hardware

The PDP-9/339 Remote Graphics Terminal is built around a DEC PDP-9 central processing unit. This is a reasonably powerful 18-bit minicomputer with 16K words of core and the usual peripherals of teletype, high speed paper tape reader and high speed paper tape punch. Included in the processor is an Extended Arithmetic Unit which provides hardware multiply and divide operations and greatly enhances the power of the PDP-9 to do such operations as rotations and scaling locally.

Attached to the PDP-9 is a DEC 339 display control. This is essentially a separate, special-purpose processor which cycles through the common memory in parallel with the PDP-9 executing its own instruction set to display lines and points. Included with the display control is a "light pen", a hand held photo sensitive device which detects the light from displayed pictures, and a push-button box for programmed function keys.

Communication with MTS is possible using a 201A dataphone dialed in to the Data Concentrator. Records can be sent or received at transmission rates up to 2000 bits per second, with automatic error recovery provided for loss of data over the line.

Finally, another mode of communication with the graphics terminal consists of a three channel analog-to-digital convertor and a 16 channel digital-to-analog convertor. These devices can be attached to whatever analog devices the user provides.

### 2.2 Software

#### 2.2.1 The SEL Executive System

The SEL Executive System serves as the interface between user written PDP-9 programs and the hardware. It provides subroutines for buffered I/O to and from the teletype, paper tape equipment, and the 201 dataset (MTS). System subroutines also manage the free core pool, do task switching, input data from the A/D convertor, push button box, and real time clock, and output data to the D/A convertor and pushbutton lights. The system command language provides various debugging operations and allows communication with MTS as a straight teletype terminal.

The system display support provides routines to draw, erase and move user generated picture elements called "leaves". It also takes care of all synchronization between the PDP-9 and the 339 necessary due to the multiprocessing nature of the system.

Input comes from the display via the light pen. Upon sensing a light pen hit, the system notifies the user program and provides information about what was hit and where on the

screen it was. Continuous X,Y coordinate input can also be obtained by using the system's tracking routines.

### 2.2.2 User Programs

The SEL System provides only the basic I/O support for the terminal. Actually building data structures and creating display leaves is the job of user written PDP-9 programs. Such user programs can then read lines from MTS (the system treats the dataphone connection to MTS as just another I/O device) and process them to provide whatever special purpose graphical operations are desired. For example, since the dataphone can transmit only about 250 characters per second, it is a good idea for the MTS program to send only the data represented by a picture and for the PDP-9 program to read this in and reformat it into the more lengthy 339 instruction codes for a display leaf. The precise division of labor between the 360 and the PDP-9 can thus be tailored to the users particular application. Generally, the 360 is used for mass storage and number crunching while the PDP-9 does highly interactive I/O operations and fast graphical operations. Applications which require small amounts of computations and very fast response times could perhaps be best done totally in the PDP-9 using the 360 only for assemblies, etc.

### 3. SYSTEM OPERATION

#### 3.1 Loading the System

- 1) Turn power on.
- 2) Set all the white toggle switches down except for switches 13 and 16 of the middle group (up).
- 3) Hit I/O RESET key, hit START key.
- 4) If the message PANEL RECOVERY appears on the screen you are done, otherwise:
- 5) Set all of the middle row of white toggle switches down.
- 6) Place the tape labeled SEL EXECUTIVE SYSTEM in the paper tape reader.
- 7) Hit the I/O RESET key, hit READ IN key.
- 8) Message SYSTEM RELOADED should appear at the end of tape read in.
- 9) If there is a tape labeled SEL SYSTEM OVERRIDES with the system tape, place it in the reader and hit on the teletype: FPC. You are now done.

#### 3.2 System Initialization and Error Comments

Various conditions described below will cause a system reinitialization. A reinitialization performs the following actions:

- Clears the display.
- Clears the task queue.
- Unlocks all system subroutines.
- Turns off pushbutton lights.
- Stops light pen tracking.
- Clears the #BD buffer.
- Disables light pen hits.
- Routes all subsequent 201 records to the display.
- Reinitializes the free storage pool.
- Stops all I/O devices and empties their buffers.

(The last action is not performed if the reinitialization is due to TASK QUEUE EMPTY.) A reinitialization is signalled to the user by ringing the teletype bell and typeing "!". A diagnostic message is then placed on the screen followed optionally by a dump of all active registers: Link, Accumulator, Multiplier Quotient, Step Counter, and Program Counter. An explanation of the possible diagnostic messages follows.

SYSTEM RELOADED  
Initial program load of the system.



## PANEL RECOVERY

Panel restart of the system from location 00022.

## INVALID INTERRUPT

The I/O interrupt processor was entered but no device flags were set. This can be caused by a hardware error or by the user program making a wild jump into the system core bank. A register dump follows. The value displayed for the PC is the contents of location 0 at the time the error condition was recognized.

## MANUAL INTERRUPT

The user hit the INTERRUPT button on the push button box. This serves as an "attention interrupt" to the local system. A register dump follows.

## TRAP

The PDP-9 executed an illegal instruction (op-code 0). This is usually caused by attempting to execute data or by jumping to an undefined symbol. A register dump follows. The value displayed for the PC is the location of the TRAP instruction.

## TASK QUEUE EMPTY AT LOC xxxxx

A call to #TF was made and there were no further tasks on the task queue. This is the normal way for a user program to terminate. xxxxx is the address of the last call to #TF.

## TASK QUEUE OVERFLOW AT LOC xxxxx

A call to #TS was made and the task queue was already full. xxxxx is the address of the last call to #TS.

## PDS OVERFLOW AT LOC xxxxx

The display attempted to push more than 64 entries on the display push down stack. This can be caused by the user having too many levels in his display hierarchy, by inserting a node as a sub-picture of itself or by inserting a bad leaf into the display structure. xxxxx is the address of the display PJMP instruction which caused the overflow. A dump of the addresses of the branches on the push down stack follows this diagnostic.

## BAD LEAF AT LOC xxxxx

The display encountered a STOP instruction. This is invariably caused by the user program inserting a bad leaf in the display structure. xxxxx is the last address PJMPed to. A dump of the addresses of the branches on the push down stack follows this diagnostic.

## BAD PARAMETER yyyyyy AT LOC xxxxx

A parameter given to one of the system subroutines did not look at all like it should have. xxxxx is the location of the offending parameter word and yyyyyy is the parameter value the routine thought was passed to it.

## 4. COMMAND LANGUAGE INTERPRETER

After a system reinitialization, the Command Language Interpreter is activated to execute commands entered at the teletype. The CLI indicates readiness for a command by typing "!" as a prompting character. In all command descriptions to follow, only the underlined characters need to be typed; the system will automatically echo the rest. If a character is typed which does not follow the specified sequence for any of the commands, it is totally ignored. At any time during the typing of a command, hitting RUBOUT will echo a question mark and delete the command.

4.1 CLEAR Command

This command requires confirmation before being executed. After typing "CD" or "CU" the user must type "O" (for OK) to confirm the command. Typing anything else echoes "NO" and the command is canceled.

CLEAR USER CORE? OK

This command stores a trap instruction (000000 octal) in the entire user core bank from location 20000 through 37777. If execution ever reaches one of these instructions, the system immediately terminates execution with the error comment "TRAP". This command is generally issued before loading an un-debugged program to hopefully catch the program if it goes wild.

CLEAR DISPLAY? OK

This command removes all text from the display screen, thus making room for more text.

4.2 FROM Command

	<u>PAPER</u> <u>TAPE</u>		<u>PAPER</u> <u>TAPE</u>
	<u>CORE</u>		<u>CORE</u>
<u>FROM</u>	<u>TELETYPE</u>	<u>TO</u>	<u>TELETYPE</u>
	<u>201</u>		<u>201</u>
			<u>DISPLAY</u>

This command copies data from the "FROM" (source) device to the "TO" (sink) device. The copying function is alphanumeric or binary depending on the combination of I/O devices specified and on the data read from the source device.

## 4.2.1 Source Devices

FROM PAPER TAPE TO ...

Paper tape is read for copying until a paper tape end-of-file character is reached. The SEL system uses blank tape (octal 000) as the end-of-file character. The tape can be alphanumeric or binary. Alphanumeric format consists of ASCII

character codes with the parity bit forced on. Lines are terminated by carriage return (octal 215) followed by line feed (octal 212). Binary tape consists of origin frames followed by a series of data frames of the form:

```

binary      | ●xxx.xxx|
origin      | ●xxx.xxx|
            | ●xxx.xxx|
binary      | ● xxx.xxx|
data        | ● xxx.xxx|
            | ● xxx.xxx|
            | ● xxx.xxx|
            | ● xxx.xxx|
            | ● xxx.xxx|
            | ● xxx.xxx|
            | ● xxx.xxx|

```

The tape is determined to be alphanumeric or binary by examining the first character read. If the high order two bits are 01 the tape is assumed to be the first character of a binary origin. Otherwise it is assumed to be an ASCII tape. If an invalid character is detected in the middle of a binary tape a comment is printed, the CLI scans for the next origin frame and continues the copy operation. This allows the user to salvage the part of the tape following the error.

#### FROM CORE TO ...

This command solicits the user for the high and low addresses of the core block to be copied. The user then fills in the blanks in the line:

```
□ BLOCK(_____,_____)
```

A transfer from core is always binary.

#### FROM TELETYPE TO ...

Copy operations from the teletype are different for each sink device.

```

... TO PAPER TAPE
... TO DISPLAY
... TO TELETYPE

```

The user is prompted for lines by the prefix character "□". Text typed in is subject to the line editing characters:

<u>Char.</u>	<u>Echo</u>	<u>Function</u>
ctl-H	backarrow	Backspace
RUBOUT	#	Delete line
RETURN	cr-lf	End of line
ctl-C	backslash	Terminate copy function

... TO CORE

This command allows the display and modification of core locations. Typing a five digit octal address causes a display of the contents of that address. The user can then type:

6-digit octal number	Store this new value in the current location and display the next sequential core location.
RETURN	Do not change the current location but display the next sequential location.
RUBOUT	Terminate copy to core.

This command is assumed whenever the CLI expects a command (prefix `␣`) and the user types in an octal address.

... TO 201

This command allows communication with MTS. The user is prompted for lines by the prefix character sent by MTS (see description of copying TO 201). Text typed in is subject to the following special character interpretations:

<u>Char.</u>	<u>Echo</u>	<u>Function</u>
ctl-H	backarrow	Backspace
RUBOUT	#	Delete line and retype prefix
RETURN	cr-lf	Send line to MTS
ctl-C	backslash	Send end-of-file to MTS
ctl-E	!	Send attention interrupt to MTS
ctl-shft-K		Invert "surpress echo" switch.
Alt-mode		Return to SEL System CLI.

This command can also be given by striking the alt-mode key when the command language interpreter is expecting a command. Thus repeatedly striking the alt-mode key alternates between talking to the local system and talking to MTS. The user can always see which system he is talking to by looking at the prompting character:

"␣" or "␣ "	SEL System
"#" or ">" or " " etc	MTS

FROM 201 TO ...

Since records arrive from the dataphone at arbitrary times, this command only sets switches to route future records to the specified sink device. The command will prompt the user with "␣" for further information of form:

␣ BINARY	All binary records received from the 201 will be sent to the specified sink device.
␣ PREFIX= <u>X</u>	All ASCII records with the specified prefix character are routed to the specified sink device.

□ DEFAULT                      All other ASCII records will be sent to the specified sink device.

A system reinitialization sets up the routing switches as though the following commands were given to route all records received from the 360 to the display:

```

↓ FROM 201 TO DISPLAY
□ DEFAULT
↓ FROM 201 TO DISPLAY
□ BINARY

```

As another example, the following sequence of commands causes all future binary records received from MTS to be punched, all ASCII records prefixed by ">" to be typed and all other ASCII records to be written on the display:

```

↓ FROM 201 TO PAPER TAPE
□ BINARY
↓ FROM 201 TO TELETYPE
□ PREFIX=>
↓ FROM 201 TO DISPLAY
□ DEFAULT

```

The following processing is performed on records read from the 201.

For copying to TELETYPE or DISPLAY:

ASCII records are copied intact (including the MTS prefix character).

For copying to TELETYPE, DISPLAY or CORE :

Binary records are interpreted in PDP-9 paper tape binary format. If non standard binary records are recieved from the 201, an attention is immediately sent to the 360 and an appropriate comment is typed.

For copying to PAPER TAPE :

ASCII records have the first character (MTS prefix character) removed to facilitate copying the tape back into a file later. Binary records are copied intact so that any arbitrary binary format may be punched.

Note that the FROM 201 command does not actually try to read from the 201. See the description of the 201 as a sink device to see how this is done.

#### 4.2.2 Sink Devices

FROM ... TO PAPER TAPE

Characters are punched in the same binary and ASCII formats described in section 3.2.1. Whenever an alphanumeric file is punched, a short length of blank tape is fed out at the beginning and at the end of the punched tape file to ensure separation of individual ASCII files. When a binary file is

punched, no such blank tape is put out. Thus, copying two consecutive binary files to the punch concatenates them into one file.

FROM ... TO CORE

Copying to core is essentially a load operation from the source device and thus is always binary. For example the command: FROM PAPER TAPE TO CORE loads a binary paper tape into core, etc.

FROM ... TO TELETYPE

Copying ASCII files to the teletype is essentially a list operation. Copying binary files produces a core image dump in the format:

```
20000 000000 000000 000000 000000 000000 000000 000000 000000
20010 000000 000000 000000 000000 000000 000000 000000 ...
```

The first number on each line is the memory address. The subsequent numbers are the contents of memory from that address on up to a maximum of eight locations per line.

FROM ... TO DISPLAY

This is similar to copying to the teletype except that the information appears on the face of the display. When the storage allocated for display text is exhausted, the push button lights are examined. If button 9 is lit and 10 is off the screen will be cleared and copying continued. If button 10 is lit and 9 is off the text will be scrolled up one line and copying continued. If 9 and 10 are both off or both on then button 11 begins flashing. The copy operation then waits for the user to hit 9 or 10 (at which time the above test is repeated) or button 11 (at which time the copy operation is aborted).

FROM ... TO 201

Issuing a copy to the 201 is what actually starts activity at the 201 interface. The copy operation begins by reading records and routing them according to the switches set by the FROM 201 command. It continues to read until it receives a record terminated by an ETX (octal 203) character. This will be an MTS prefix record which indicates that MTS is waiting for a read. When copying FROM TELETYPE TO 201, the prefix record is typed out to prompt the user for entry of a line. When copying from any other device, the prefix record is swallowed up. A line is then read from the source device and sent to MTS. When MTS sends another prefix record the next line is read and transmitted.

For example, the following sequence of commands sets things up so that the teletype acts like a normal terminal:

```
⌘FROM 201 TO TELETYPE
⌘DEFAULT
⌘FROM TELETYPE TO 201 (or alt-mode )
```

The following sequence of commands punches the contents of the file OBJ in binary. This is how to get an object tape from an assembly on the 360.

```
⌘FROM 201 TO PAPER TAPE
⌘BINARY
⌘FROM TELETYPE TO 201 (or alt-mode )
#$COPY OBJ TO *SINK*@BIN
```

The following sequence of commands copies an ASCII tape into a file in the 360.

```
#$COPY *SOURCE* TO FILE
>alt-mode
⌘FROM PAPER TAPE TO 201
```

Note that in all such operations, the user must set up the sink device first. When copying to MTS, give the MTS command first. When copying to the PDP-9, give the SEL command first.

#### 4.3 RUN Command

RUN aaaaa

Clears the screen and jumps to the address typed in as aaaaa. This is how to start up a user program.

## 5. DISPLAY CONTROL PROGRAMS

In addition to the usual jump, subroutine-call, and subroutine-return instructions, the instruction set of the 339 display control has special instructions to move the electron beam across the CRT to draw lines and points. Execution of a line drawing instruction intensifies points along the line for a brief period of time. To provide a steady image, the entire 339 program must be executed over and over so that the repeated intensifications merge into a solid picture.

The SEL system maintains the 339 display control program in the form of a tree-like data structure. Thus the operations of drawing, erasing, and translating picture elements become those of inserting, deleting, and modifying elements of the structure. This arrangement not only eases various graphical operations but also allows the user to store some relational data about the picture in the form of the tree structure that he builds.

### 5.1 Root Node

The root node is the beginning of the display program. The system re-starts the display control here at the end of each frame. Initially it consists of a 339 instruction which jumps to a stop instruction. The node in this state is said to be empty.

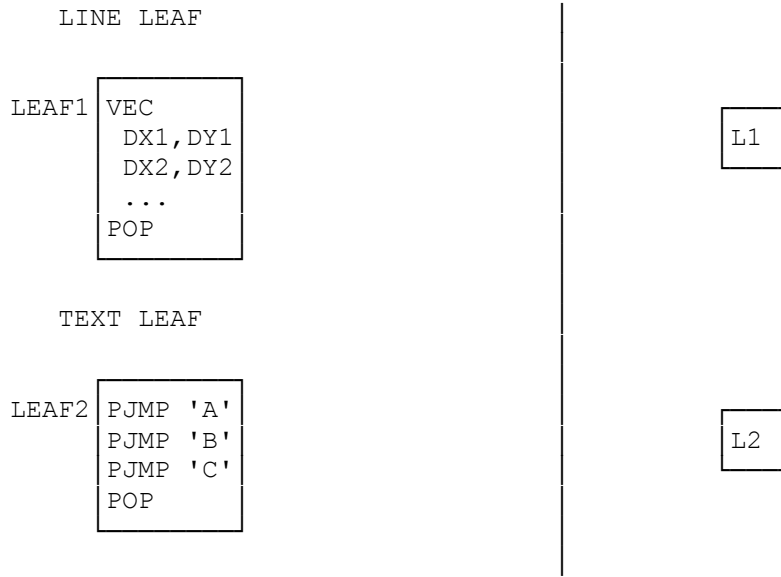


### 5.2 Leaves

A leaf is the structure element which actually paints a part of the picture on the screen. Leaves that make line drawings must be created by the user, either by defining a region of constants in the user program or by dynamically computing the leaf contents while the program is running. Leaves that display a line of text are created by calling a system subroutine (#BD). A leaf can, in general, consist of any sequence of 339 display instructions terminated by a display POP instruction. It is recommended, however, that the following conventions be followed.

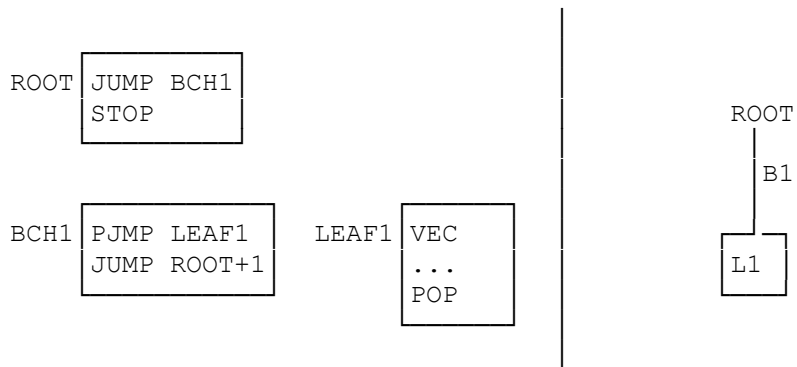
- 1) The first word in the leaf should be a display PAR instruction.
- 2) All drawing in the leaf should be done in VEC, SVEC, or INCR mode.
- 3) The sum of all X and Y displacements within the leaf should be zero (i. e. The leaf closes back on itself).



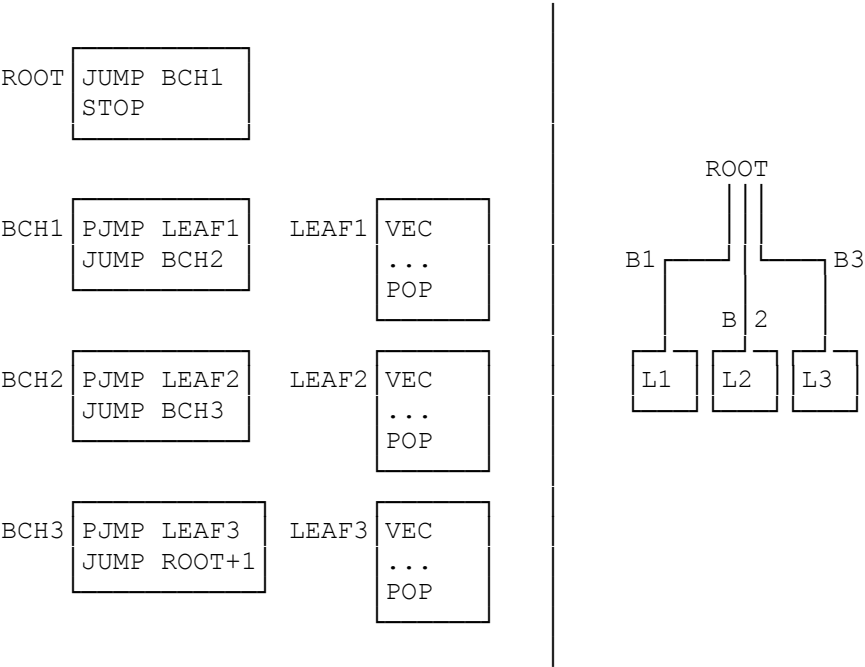


### 5.3 Branches

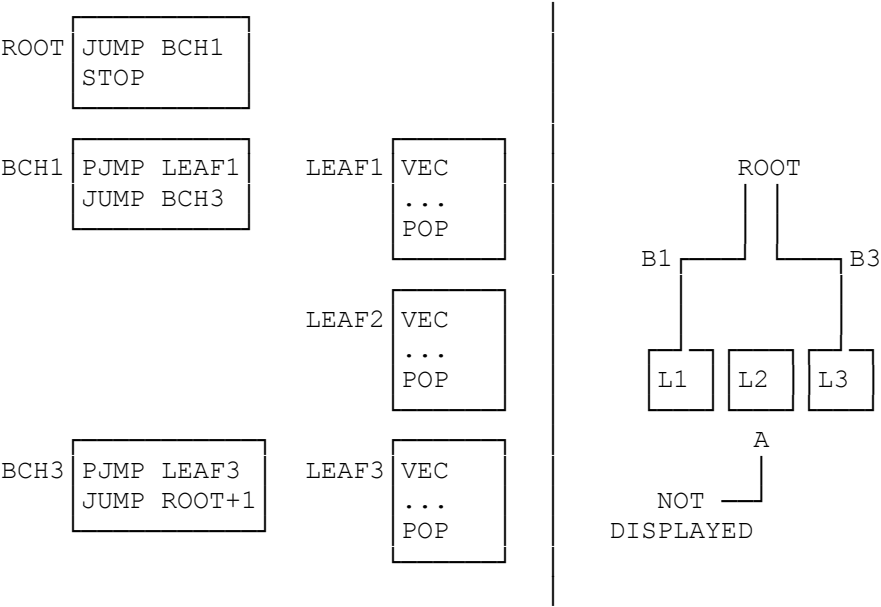
Simply creating the leaf-subroutine will not get it displayed, it must first be linked into the 339's execution loop. Such a link is called a branch. To display a leaf on the screen, the user calls a routine called #SBC to create a branch from the root node to the leaf. This routine modifies the structure to look like the following.



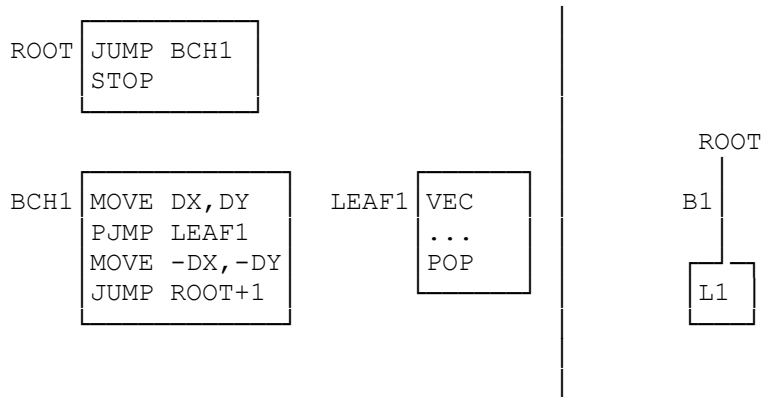
Several different leaves may be similarly linked into the structure to get several pictures on the screen.



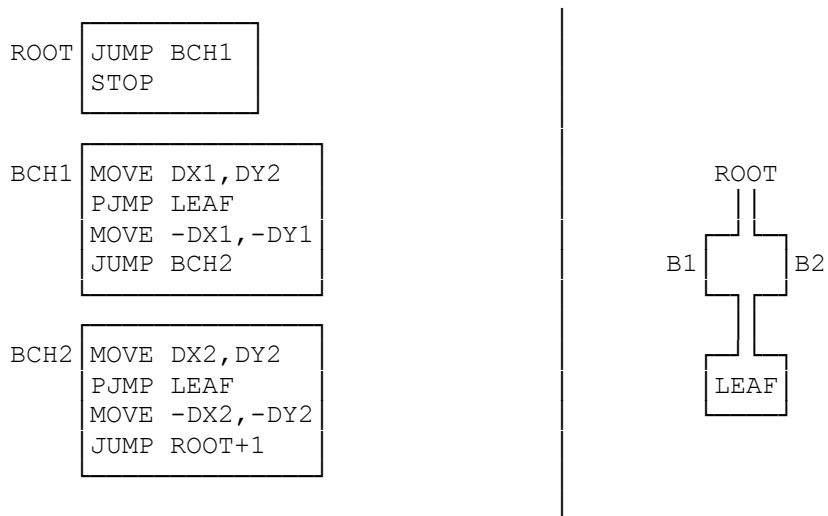
To remove a leaf from the screen, call the routine #SBD to destroy the appropriate branch.



Once a leaf has been placed on the screen, it can be repositioned by calling #SBAC to apply a translation to the branch. A translated branch looks like:

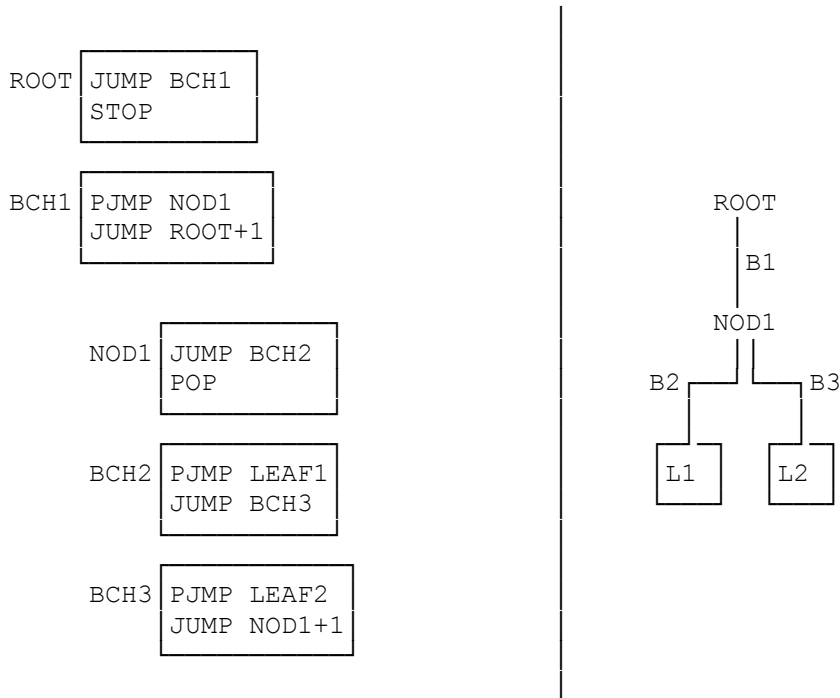


Using branches with translations, you can have one leaf appearing at two places on the screen.



### 5.4 Nodes

With several leaves attached to the root node it might be desirable to move this entire picture. For this purpose, the user sets up a more complex structure by calling #SNC to create his own nodes. He may then attach leaves to them and then attach the node to the root node.



Then altering the coordinates of B1 moves the whole picture while altering the coordinates of B2 or B3 alter only a part of it.

### 5.5 Storage Block Format

This structure serves not only to provide a display control program for the display of the picture but also provides relational information about the picture. In order for the program to get at this relational data, the block format of the structure elements is described below.

#### NODE

60200a	display JUMP instruction
0aaaaa	address of first branch in node
763000	display POP instruction

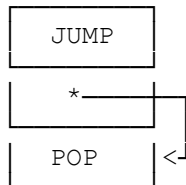
#### BRANCH with (0,0) displacement

60201a	display PJMP instruction
0aaaaa	address of node or leaf branched to
60200a	display JUMP instruction
0aaaaa	address of next branch in node

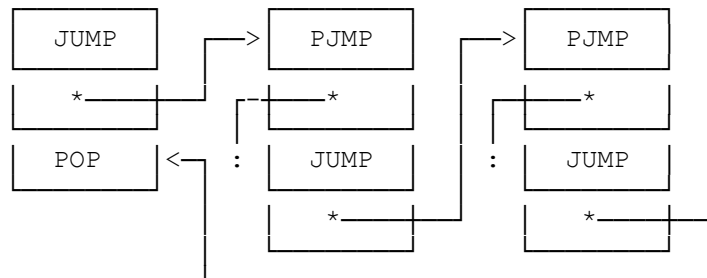
BRANCH with (x,y) displacement

761121	display VEC mode instruction
00yyyy	y displacement in VEC data mode format
00xxxx	x displacement in VEC data mode format
60201a	display PJMP instruction
0aaaaa	address of node or leaf branched to
761121	display VEC mode instruction
00yyyy	-y displacement in VEC data format
00xxxx	-x displacement in VEC data format
60200a	display JUMP instruction
0aaaaa	address of next branch in node

Upon creation, a node has no branches in it so the second word of the node block points to the third word.



After inserting two branches into the node the structure looks like:

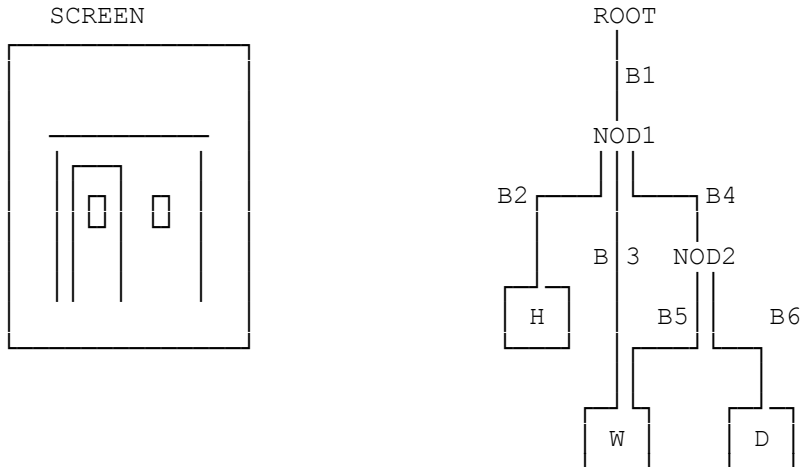


The user can use the routines NXTBCH, NXTNOD, and GETXY to examine the structure. These routines can be included in the source stream by the following statements.

COPY	W339:LIB(600) -- NXTBCH
COPY	W339:LIB(700) -- NXTNOD
COPY	W339:LIB(800) -- GETXY

### 5.6 Example Program

To illustrate the operation of the display control when executing a typical program and to describe the function of the 339's push down stack mechanism we will consider the following picture and display structure.



Leaf H draws the outer walls of the house, leaf W draws a square window and leaf D draws a door frame. NOD2 represents the entire door consisting of frame and window, while NOD1 represents the entire house consisting of walls, window and door.

The display control starts execution at ROOT. It jumps to branch B1 which calls NOD1 as a subroutine. The 339 subroutine call instruction is called a push-jump, that is, it saves the return address in a core buffer called a push-down-stack. Thus while executing the NOD1 subroutine, the address of B1 is on the push down stack. NOD1 jumps to branch B2 which calls leaf H to draw the walls. While the 339 executes this leaf the push down stack looks like:

```

B1
B2

```

when leaf H returns the 339 removes the address of B2 from the stack and jumps to branch B3. This calls leaf W to draw the window in the wall of the house and the push down stack looks like:

```

B1
B3

```

upon return from W the display jumps to branch B4 which calls NOD2. This jumps to branch B5 which calls W again. While the 339 draws the window in the door the push down stack looks like:

```

B1
B4
B5

```

W returns and the 339 jumps to B6 and calls leaf D. While the door frame is being drawn the PDS looks like:

B1  
B4  
B6

The frame finally terminates by NOD2 returning to NOD1, NOD1 returning to ROOT and the display stopping.

In summary, note that while the 339 executes a leaf the push down stack contains the addresses of the branches gone through to get from ROOT to that leaf. This information is quite useful when processing light pen hits on complex display structures.

## 6. THE LIGHT PEN

The light pen serves as the input device from the display. With it the user can input X,Y coordinate data (via "tracking") or he can indicate a part of the displayed picture to the program (via "light pen hits").

### 6.1 Light Pen Hits

A light pen hit occurs whenever the user points at an intensified portion of the screen while depressing the shutter button of the light pen. A light pen hit is possible on a particular leaf only if the following three conditions are met:

- 1) The light pen hardware is enabled by a parameter instruction placed (by the programmer) in the first word of the leaf (see Appendix B).
- 2) Light pen hits have been globally enabled by a call to #DE.
- 3) A service task has been assigned (via #DP) to the class number of the leaf. The class number is stored (by the programmer) in the high order six bits of the first word of the leaf.

If these conditions are met then the system, upon detecting a light pen hit, saves the entire status of the 339 hardware (339 program counter, X coordinate, Y coordinate, push down stack). It then disables light pen hits (via #DD), schedules the service task (via #TS) and restarts the display. The service task can then process the light pen hit without the screen going blank. No further light pen hits will be honored until the task calls #DE. While running, the task can find the state of the display control at the time of the light pen hit by calling on the subroutines #DX, #DY, #DA, and #DO.

### 6.2 Tracking

The position of the light pen can be continuously monitored by the process of tracking. Tracking will usually be initiated by the service task of a light pen hit. The task calls #XI which places the tracking cross on the screen at the coordinates of the light pen hit. The system then continually adjusts the position of the tracking cross to keep it centered on the light pen. The user program can read out the current position of the cross by the #XX and #XY routines. Tracking is terminated and the cross is removed from the screen if the program calls #XT or if the operator closes the shutter of the light pen, thus preventing it from seeing the tracking cross. The program can detect this condition via the #XS routine.



## 7. SYSTEM SUBROUTINES

7.1 Calling Conventions

All system subroutines are in the system core bank (locations 0000 through 17777). Since the only way to reference across core banks on the PDP-9 is with indirect addressing, calls on system routines must be of form:

```
JMS*  =#BK
```

Parameters are passed in the words following the JMS instruction. A parameter word is interpreted in the following way.

If the three high order bits of the contents of a parameter word are zero the contents of the parameter word is the data.

```
JMS*  =#BT
DC     301  Top bits are clear, data here
```

If the three high order bits of the contents of a parameter word are non-zero, the parameter word is interpreted as an instruction which, when executed, will load the data into the accumulator.

```
JMS*  =#BT
LAC    DATA Instruction executed
                        to get data in AC
.
.
DATA   DC     301
```

This type of parameter will be represented in the following subroutine descriptions as:

```
JMS*  =#BT
LAC    ----  character to type
```

The first character of the name of a system subroutine is always "#". System subroutine entry points must be defined to the assembler by including the pseudo-operation:

```
COPY  W339:EQU
```

as the first statement of an assembler source program.

7.2 Task Scheduling Routines

## #TS (Task Schedule)

Calling sequence: JMS\*   =#TS  
                   LAC     ----       address of task

Adds the parameter value to the bottom of the task queue and immediately returns.

## #TF (Task Finish)

Calling sequence: JMS\*   =#TF       does not return

Fetches the next address from the top of the task queue and jumps to it. If the task queue is empty, it returns to the system and activates the command language interpreter.

## #TW (Task Wait)

Calling sequence: JMS\*   =#TW  
                   ----       returns here (eventually)

Calls #TS to schedule the return location and then calls #TF, thus doing a wait on the task queue. Note that the contents of the AC, MQ and Link are unpredictable upon return from this routine. #TW should be called in all loops that are extremely long or that check switches set by other tasks, to give other tasks a crack at the CPU.

## #TL (Task Lock)

Calling sequence:  
                   SUBROU   DC     0  
                               JMS\*   =#TL  
                               DC     0

This routine is useful to provide mutual exclusion between two (or more) tasks which call the same subroutine. This mutual exclusion is necessary if 1) the two tasks are scheduled simultaneously and 2) the subroutine calls #TW (explicitly or implicitly). Immediately upon entry to SUBROU, #TL examines the location at SUBROU+2. If it is zero, the return pointer from SUBROU is stored here and #TL returns to SUBROU+3. If SUBROU+2 is nonzero another task has already called SUBROU but has not returned yet. #TL then reschedules the call to SUBROU (via #TS) and terminates (via #TF). SUBROU can access parameters passed to it via the return pointer in SUBROU+2. Note that locked subroutines cannot pass parameters in registers since it might call #TS immediately upon entry. The initialization procedure of the users program should unlock all such routines by zeroing out the SUBROU+2 locations.

## #TU (Task Unlock)

Calling sequence: JMS\*   =#TU  
                   DC     SUBROU

Unlocks the subroutine SUBROU so that other tasks may enter and effects a return from SUBROU through the return pointer in

SUBROU+2. The subroutine must have the entry sequence described for #TL above. #TU preserves the AC, MQ, and Link contents so that the user subroutine can return values in those registers.

### 7.3 Buffered I/O Routines

#BK (Buffered Keyboard)

Calling sequence: JMS\* =#BK

---- ASCII character in AC.

Reads a character from the teletype. Note that the teletype is a full duplex device, i. e. Hitting a keyboard key does not automatically type it on the paper. The user program has to explicitly do this.

#BT (Buffered Teleprinter)

Calling sequence: JMS\* =#BT

LAC ---- character to type  
---- returns here

Types an ASCII character on the teletype.

#BR (Buffered Reader)

Calling sequence: JMS\* =#BR

---- reader character in AC

Reads the 8-bit pattern from paper tape into the accumulator. If the reader is out of tape, and end-of-tape character (viz. -1) is returned. The system automatically starts the reader upon a call to #BR if the reader buffer is empty and automatically stops the reader when the reader buffer is full or when a blank tape character (000 octal) is read after a non-blank character. A user program still must explicitly ignore the blank tape at the beginning and end of a punched tape.

#BP (Buffered Punch)

Calling sequence: JMS\* =#BP

LAC ---- character to be punched  
---- returns here

Punches the binary bit combination in the low order 8-bits of the parameter value.

#BFI (Buffered Fone Input)

Calling sequence: JMS\* =#BFI

---- fone character in AC

One whole record is received from MTS by the 201 dataset and this record is fed to the user program one character at a time by #BFI. The first character of a record indicates if the record is binary or ASCII character codes and will be:

001 (SOH) Rest of record is ASCII characters

002 (STX) Rest of record is binary bytes written by MTS with the @BIN modifier.

The last character of a record is followed by an end-of-record character flagged by or-ing 760000 with it, thus making it appear negative. The possible end-of-record characters are:

760227 (ETB) Normal end-of-record

760203 (ETX) This character terminates a prefix record and indicates that MTS is waiting for a read.

#BFO (Buffered Fone Output)

Calling sequence: JMS\* =#BFO

LAC ---- character to output  
 ---- returns here

The system fills up the output buffer one character at a time. The first character of a record must be a control character and is not transmitted to MTS. The valid control characters are:

001 (SOH) Rest of record is to be interpreted as ASCII character codes and will be translated to EBCDIC character codes by the Data Concentrator. These records are vulnerable to command language operations both in the DC and in the MTS DSR system.

002 (STX) Rest of record is to be interpreted as 8-bit binary bytes with no modification performed between the PDP-9 and MTS. This record should be read by MTS with the BIN modifier set.

If the first character given to #BFO is other than the above, an SOH (ASCII record) is inserted in front of the first character.

When an end of record character is given to #BFO, the entire record is transmitted. An end-of-record character is distinguished by having its high order bit set, thus appearing negative. The valid end-of-record characters are:

400000+xxx227 (ETB) Normal end-of-record, record sent to MTS

400000+xxx203 (ETX) Record is sent to MTS followed by an end-of-file indication.

If the end-of-record character given to #BFO is other than the above, an ETB is assumed.

#BFA (Buffered Fone Attention)

Calling sequence: JMS\* =#BFA  
 ---- returns here

Sends an attention interrupt to the 360 and clears the dataphone input buffers. The next character read via #BFI will be the first character of the record sent as a response to the attention.

#BD (Buffered Display)

Calling sequence: JMS\* =#BD  
 LAC ---- ASCII character to display  
 ---- returns here if no storage  
 ---- address of leaf in AC  
 (if created)

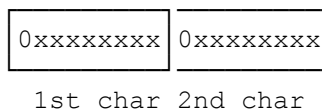
Adds an ASCII character to the display leaf buffer (all non-printing ASCII characters are ignored). If an end-of-line character is given to #BD, it creates a leaf which displays the characters currently in the buffer and then clears the buffer. An end-of-line character is distinguished by having its high order bit set, thus appearing negative. The text leaves are created from the free storage pool and should be returned to the free storage pool (via #CF) when they are no longer needed. The first word of the created leaf will be octal 000000 so that the user can store a leaf class and display parameter instruction of his own choosing there. Note that a failure return is only possible if the routine attempts to create a leaf, that is, if the parameter is negative.

#### 7.4 Text List Subroutine

#L (List)

Calling sequence: JMS\* =#L  
 LAC ---- address of text list  
 LAC ---- address of output routine

This subroutine gets a character string from a text list and call the output subroutine once for each character. A text list consists of eight bit ASCII character codes packed two to a word in the format:



The character string is terminated by the 400000 or 400 bit being set in the character position after the last character. This appears in octal as:

DC	301400	list contains "A"
DC	301302	
DC	303304	list contains "ABCD"
DC	400000	

the calling sequence of the output subroutine should be:

```
JMS  outsub  (e. g.  #BT,#BF,#BD)
DC    ----   character stored here
```

Note that this is a special case of the calling sequence of the system buffered character output routines. Thus the call:

```
JMS*  =#L
DC    LIST    address of text list
DC    #BT     teletype output routine
```

types a list on the teletype. The #L subroutine is written so that #BD can also be used as the output routine. However, since #L cannot give it an end-of-line character, #BD will never fail when called by #L. To create a text leaf from a text list, use the following:

```
JMS*  =#L
DC    LIST
DC    #BD
JMS*  =#BD
DC    -1
-----   no storage for leaf
-----   success; leaf address in AC
```

### 7.5 Non-Buffered I/O routines

#NC (Non-buffered Clock)

```
Calling sequence: JMS*  =#NC
-----           return here with time in AC.
```

Returns the current value of the clock location. This location is incremented by the system every 1/60 second. This routine is used chiefly for measuring time intervals.

#NPR (Non-buffered Pushbutton Read)

```
Calling sequence: JMS*  =#NPR
-----           pushbutton state in AC
```

Reads the current state of pushbutton lights 0-11 into bits 6-17 (low order 12 bits) of the accumulator. A set bit means "on" and a clear bit means "off". The state of a pushbutton light is inverted by pushing that button.

#NPS (Non-buffered Pushbutton Set)

```
Calling sequence: JMS*  =#NPS
LAC    ----   state to set pushbuttons to
-----           returns here
```

Sets pushbutton lights 0-11 according to bits 6-17 of the parameter. A set bit means "on" and a clear bit means "off". The state of a pushbutton light is inverted by pushing that button.

7.6 Core Allocation Routines

## #CG (Core Get)

Calling sequence: JMS\* =#CG  
                   LAC ---- number of words to get  
                   ---- failure; no more core  
                   ---- OK; address of block in AC

Gets a block of core of specified length from the free storage pool.

## #CF (Core Free)

Calling sequence: JMS\* =#CF  
                   LAC ---- address of block to free  
                   ---- returns here

Returns a block of core to the free storage pool. The parameter must be the address of a block previously returned by #CG. That is, you can not get a block of core and then only free part of it.

## #CA (Core Add)

Calling sequence: JMS\* =#CA  
                   LAC ---- address of block to add  
                   ---- returns here

Adds part of the user core bank to the free storage pool. The added block begins at the address given as a parameter and extends through location 37777.

7.7 Display Structure Operations

## #SNC (Structure Node Create)

Calling sequence: JMS\* =#SNC  
                   ---- failure; no storage  
                   ---- OK; address of node in AC

Creates the three word block for a node out of storage obtained from the free storage pool. See section 5.5 for the block format.

## #SND (Structure Node Destroy)

Calling sequence: JMS\* =#SND  
                   LAC ---- address of node  
                   ---- failure; node not empty  
                   ---- success; node destroyed

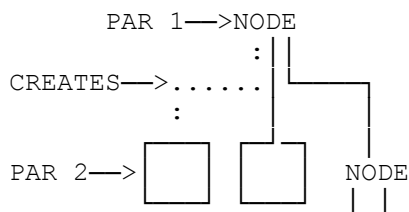
Returns the storage occupied by a node to the free storage. The node must not be inserted in the active display structure at this time or the destruction of the node will leave a big hole in the display program. The node must also not have any branches inserted in it or the #SND routine will give a failure return.

## #SBC (Structure Branch Create)

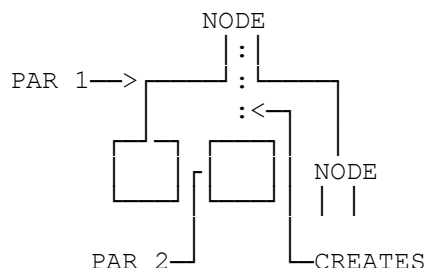
Calling sequence: JMS\* =#SBC

LAC	----	address of node or branch
LAC	----	address of node or leaf
LAC	----	y displacement (scale 2)
LAC	----	x displacement (scale 2)
----		failure; no storage
----		success; addr. of branch in AC

Creates a branch from the first parameter to the second parameter with the given displacement. The displacement coordinates are in scale 2 (octal 1000 goes all the way across the screen). If a value of zero is given as the first parameter, the branch is made from the "root" node. If a node address is given as the first parameter, the branch is inserted as the first branch in that node.



If the first parameter is a branch address, the branch is inserted after the parameter branch.



## #SBD (Structure Branch Destroy)

Calling sequence: JMS\* =#SBD

LAC	----	addrss of branch
-----	------	------------------

Deletes a branch from the display structure thus disconnecting the nodes or leaves at each end. The storage occupied by the branch is returned to the free storage pool.

## #SBAS (Structure Branch Alter Subpicture)

Calling sequence: JMS\* =#SBAS

LAC	----	address of branch
LAC	----	address of new node or leaf
----		failure; no storage
----		OK; new branch addr. in AC

Switches the node or leaf that the branch push jumps to to



a new node or leaf. This routine actually creates a whole new branch block and swaps it for the old one. The old branch block is destroyed and all pointers to it are then invalid. The best thing to do when calling this routine is:

```
JMS*  =#SBAS
LAC   BRANCH    branch to alter
LAC   NEWLEF    new leaf to branch to
JMP   UGH       no storage
DAC   BRANCH    update pointer to branch
```

#SBAC (Structure Branch Alter Coordinates)

```
Calling sequence: JMS*  =#SBAC
                  LAC   ----    address of branch
                  LAC   ----    new y displacement
                  LAC   ----    new x displacement
                  ----    failure; no storage
                  ----    OK;new branch addr.  In AC
```

Alters the displacement coordinates of a branch. This routine actually creates a new branch and swaps it for the old one. Therefore, the same considerations apply as for #SBAS.

### 7.8 Display State Sensing Routines

#DX (Display X-coordinate)

```
Calling sequence: JMS*  =#DX
                  ----    x coordinate in AC
```

Returns the x coordinate of the last light pen hit in scale 2 coordinates (octal 1000 points across the screen) relative to the center of the screen.

#DY (Display Y-coordinate)

```
Calling sequence: JMS*  =#DY
                  ----    y coordinate in AC
```

Returns the Y coordinate of the last light pen hit in scale 2 coordinates (octal 1000 points across the screen) relative to the center of the screen.

#DA (Display Address)

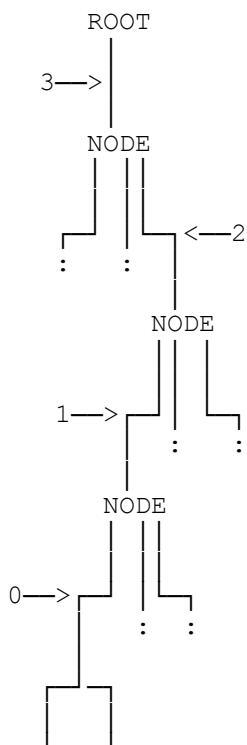
```
Calling sequence: JMS*  =#DA
                  ----    address in AC
```

Returns the address of the display instruction which was being executed at the time of the last light pen hit. This should be somewhere within a the data state portion of a leaf. If in VEC mode, the address will be the address of the Y vector just after the one which was hit.

## #DO (Display Owner)

Calling sequence: JMS\*   =#DO  
                   LAC    ----    level of ownership  
                   ----    fail; not that many levels  
                   ----    OK; address of branch in AC

Returns the address of a branch appearing on the push down stack at the time of the last light pen hit. If the structure was:



and the above leaf was pointed to with the light pen, the levels of ownership would be as shown. Calling #DO with a parameter of 4 or greater would (in this example) result in a failure return.

## #DE (Display Enable)

Calling sequence: JMS\*   =#DE  
                   ----    returns here

Enables light pen hits on the leaf classes which have had service tasks attached to them by #DP.

## #DD (Display Disable)

Calling sequence: JMS\*   =#DD  
                   ----    returns here

Disables light pen hits on all leaf classes (but does not detach them from their service tasks).

## #DP (Display Pen)

```

Calling sequence:  JMS*  =#DP
                  LAC    ----    light pen super-class
                  LAC    ----    address of 8-word table

```

This subroutine associates the addresses of service tasks with leaf class numbers. The leaf classes are first divided into eight "super classes" given by the first octal digit of the class number. Each call to #DP attaches a table to that super class. (If the second parameter to #DP is zero, a null table is attached). The eight entries in the table represent the second octal digit in the class number. A light pen hit on a particular leaf class causes the first digit of the class number to be examined to determine which table to look at. The second digit determines which entry in the table to look at. If that entry is zero, the light pen hit is ignored. If the entry is non-zero, it is taken as the address of the light pen service task and is scheduled (with #TS). Thus:

```

                JMS*  =#DP
                DC     3
                DC     TABLE
                .
                .
                .
TABLE          DC     SVC0     service task for class 30
                DC     SVC1     service task for class 31
                .
                .
                .
                DC     SVC7     service task for class 37

```

Later on, the program decides to ignore light pen hits on class 32 it can either call #DP with a whole new table, or it can zero out the appropriate entry in the currently attached table.

```

DZM    TABLE+2

```

## 7.9 Tracking Routines

#XI (X-ing Initiate)

```

Calling sequence:  JMS*  =#XI
                  ----          tracking started

```

Places the tracking cross on the screen at the coordinates obtained by calling #DX and #DY.

#XT (X-ing Terminate)

```

Calling sequence:  JMS*  =#XT
                  ----          tracking terminated

```

Removes the tracking cross from the screen.

#XS (X-ing Stopped)

Calling sequence: JMS\* =#XS

----

cross is on the screen

cross is not on the screen

Tests to see if tracking has been stopped. Three things can stop tracking: a call to the #XT routine, the user closing the shutter of the light pen so that it can no longer sense light from the screen, or the user moving the light pen outside of the active display region.

#XX (X-ing X-coordinate)

Calling sequence: JMS\* =#XX

----

X coordinate in AC

Returns the current X coordinate of the tracking cross in scale 2 (1000 octal points across the screen) relative to the center of the screen.

#XY (X-ing Y-coordinate)

Calling sequence: JMS\* =#XY

----

Y coordinate in AC

Returns the current Y coordinate of the tracking cross in scale 2 (1000 octal points across the screen) relative to the center of the screen.

## Appendix A PDP-9 INSTRUCTION SET

This appendix will briefly describe the instruction set of the PDP-9. It is intended for those who have had some assembly language experience but not necessarily with the PDP-9.

The hardware of the PDP-9 CPU consists of the following registers:

Accumulator (AC) for arithmetic operations (18 bits).

Link (L) to catch overflows out of the AC (1 bit).

Program Counter (PC) which points to the next instruction to be executed (15 bits).

Multiplier Quotient (MQ) used in multiply and divide instructions (18 bits).

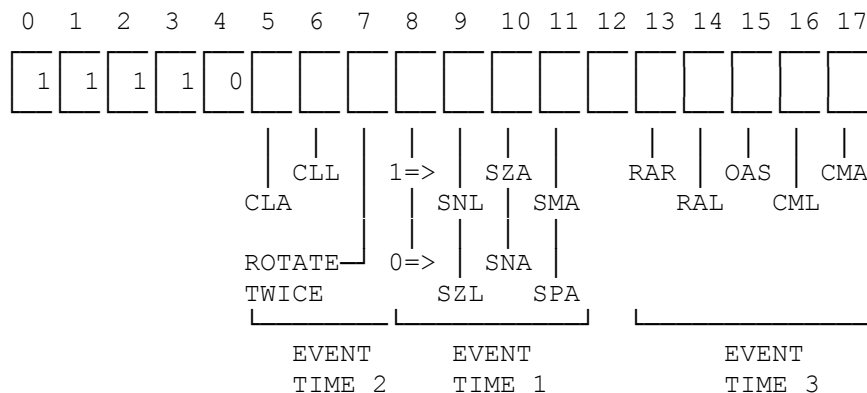
Step Counter (SC) used in shift and normalize instructions (6 bits).

The high order four bits of the instruction word contain the operation code. We will divide the PDP-9 instructions into three groups: operate instructions, memory reference instructions, and EAE (Extended Arithmetic Element) instructions.

### A.1 Operate Instructions

Operate instructions (op code 1111) are used to sense and/or alter the contents of the AC and Link without referencing memory.

The first type of operate instruction has a zero in bit 4. Each of the remaining bits of the instruction word corresponds to an individual operation; if the bit is set that operation is performed, if the bit is clear the operation is not performed.



The bits are examined during one of three "event times" to

determine if their corresponding action should be taken. Those operations which fall into the same event times but conflict in their actions should not be combined into one word.

When bit 8 is clear the skip conditions represented by the SNL, SZA and SMA bits are or-ed together. If bit 8 is set, the conditions represented by SZL, SNA and SPA are and-ed together.

<u>Instruction</u>	<u>Mnemonic</u>	<u>Octal Code</u>
No Operation NO bits are set so no operation is performed.	NOP	740000
Complement Accumulator Complement (invert) each bit of the accumulator.	CMA	740001
Complement Link Complement (invert) the Link bit.	CML	740002
Or Accumulator Switches Inclusively or the contents of the accumulator switch register (the right hand group of white toggle switches) with the AC and place the result in the AC.	OAS	740004
Rotate Left Rotate the contents of the AC and Link one bit to the left with the high order bit of the AC moving into the Link and the Link moving into the low order bit of the AC.	RAL	740010
Rotate Right Rotate the contents of the AC and Link one bit to the right with the Link bit moving into the high order bit of the AC and the low order bit of the AC moving into the Link.	RAR	740020
Skip on Minus AC If the sign bit (high order bit) of the AC is 1, increment the PC thus skipping the next instruction. If the sign bit is 0 go on to the next instruction.	SMA	740100
Skip on Zero AC If the contents of the AC is zero increment the PC by one to skip the next instruction. If the AC is non-zero, go on to the next instruction.	SZA	740200
Skip on Non-Zero Link If the Link bit is 1, increment the PC to skip the next instruction. If the Link is 0, go on to the next instruction.	SNL	740400
Skip Increment the PC by one to skip the next instruction.	SKP	741000
Skip on Positive AC If the sign bit of the AC is zero, increment the PC to skip the next instruction. If the sign bit is non-zero, go on to the next instruction.	SPA	741100

Skip on Non-Zero AC                      SNA                      741200  
 If the contents of the AC is not zero, increment the PC to skip the next instruction. If the AC is zero, go on to the next instruction.

Skip on Zero Link                      SZL                      741400  
 If the contents of the Link is zero, increment the PC to skip the next instruction. If the Link is non-zero, go on to the next instruction.

Rotate Two Left                      RTL                      742010  
 Rotate the AC and Link left just as with RAL but rotate two bit positions instead of one.

Rotate Two Right                      RTR                      742020  
 Rotate the AC and Link right just as with RAR but rotate two bit positions instead of one.

Clear Link                      CLL                      744000  
 Clear the Link bit to 0.

Clear Accumulator                      CLA                      750000  
 Clear the accumulator to 000000.

Set Link                      STL                      744002  
 A combination of CLL+CML. First clear the link to 0 and then complement it to 1.

Clear Link and Rotate Left                      RCL                      744010  
 A combination of CLL+RAL. First clear the link to 0 and then rotate left one bit position.

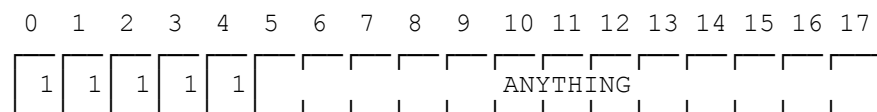
Clear Link and Rotate Right                      RCR                      744020  
 A combination of CLL+RAR. First clear the link to 0 and then rotate right one bit position.

Clear and Complement AC                      CLC                      750001  
 A combination of CLA+CMA. First clear the AC to 000000. Then complement it to 777777.

Load Accumulator Switches                      LAS                      750004  
 A combination of CLA+OAS. Clear the AC to 000000 and then or in the contents of the AC switch register.

Get Link                      GLK                      750010  
 A combination of CLA+RAL. Clear the AC and then rotate the Link into the low order bit.

If the operate instruction has bit 4 set:

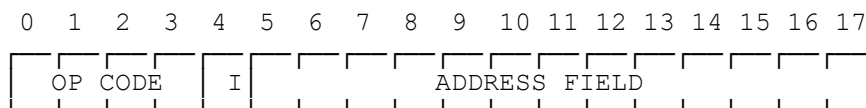


it is a "Load AC with" instruction.

Load Accumulator With           LAW n           760000+n  
This instruction simply copies itself into the AC when executed. The programmer can place any constant he wishes into the low order 13 bits of the instruction word. Note that the op code (760000) is also loaded into the AC. This instruction is useful for generating negative constants in the AC (e.g. loop counts) and for generating 339 instruction words in the AC (since the 339 ignores the high order 6 bits of an instruction word.)

## A.2 Memory Reference Instructions

A memory reference instruction word consists of three parts: a four bit operation code, a one bit indirect address flag and a 13 bit address field.



The operation code indicates which one of the PDP-9's thirteen memory reference instructions is to be performed. The indirect address bit and the address field together specify the "effective address" for the instruction. The operand of the instruction is then found in this effective address.

### A.2.1 Addressing

The PDP-9 can directly address up to 8192 locations in memory and indirectly up to 16,384 locations. The available locations have octal addresses 00000 through 37777 with the following allocations:

Memory Bank 0  
     00000 through 15777 - SEL System  
     16000 through 17777 - Free Core Pool  
 Memory Bank 1  
     20000 through 37777 - User Programs

Since user programs reside in memory bank 1, the following sections will assume that all instruction words are stored in locations 20000 through 37777.

#### A.2.1.1 Direct Addressing

For direct addressing, the indirect address flag (bit 4) is 0. In this case, the address field of the instruction specifies the displacement from the beginning of the user core bank to use as the effective address. For example, the instruction:

LAC   20100

appears in binary as:



```

010 000 000 001 000 000
└──┬──────────┘
  OP  ADDRESS

```

The address field specifies that the effective address is 00100 locations beyond location 20000 (the beginning of the user core bank). The thirteen bit address field allows direct addressing of any location within the 8192 word user core bank.

#### A.2.1.2 Indirect Addressing

For indirect addressing, the indirect address flag (bit 4) is 1. In this case, the contents of the address field specify a location in the user core bank in which the effective address is stored. For example, the instruction

```
LAC* 20100
```

appears in binary as

```

010 010 000 001 000 000
└──┬──────────┘
  OP  ADDRESS

```

This instruction directs the PDP-9 to load the AC, not with the contents of location 20100, but with the contents of the location whose address is stored in location 20100. If location 20100 contained 020077 the instruction

```
LAC 20100
```

would load 020077 into the AC. The instruction

```
LAC* 20100
```

would load the contents of location 20077 into the AC.

Indirect addressing defers execution of the instruction for one machine cycle while the effective address is read from memory. An instruction can have only one level of indirect addressing. Indirect addressing is the only way a user program can access the system core bank, e.g. to call a system subroutine or to use blocks of storage from the free storage pool.

#### A.2.1.3 Autoindexing

Eight locations, 00010 through 00017, of the system core bank serve as "auto-index" registers. Of these, 00010 through 00013 are reserved for system operations and 00014 through 00017 are free for user programs. Indirect addressing of locations 20014 through 20017 causes the contents of the corresponding locations 00014 through 00017 to be incremented by one and then taken as the effective address of the instruction. Thus,

addressing of sequential memory locations can be achieved by storing the initial address minus one into an auto-index register and then indirect addressing the auto-index register until the required operation is completed. Note that auto-indexing occurs only upon an indirect reference through 20014...20017. When directly addressed, these locations function in the same manner as the rest of memory.

### A.2.2 Instructions

In the following instruction descriptions the symbol "y" shall stand for the effective address of the instruction.

<u>Instruction</u>	<u>Mnemonic</u>	<u>Binary/Octal Code</u>
Load Accumulator Read the contents of location "y" into the AC. The contents of "y" are unchanged.	LAC y	0100/20
Deposit Accumulator Store the contents of the AC into location "y". The contents of the AC are unchanged.	DAC y	0001/04
Deposit Zero in Memory Set the contents of location "y" to 000000. The contents of the AC are unchanged.	DZM y	0011/14
Add Add the contents of location "y" to the AC following the rules of ones complement arithmetic. An arithmetic overflow sets the Link bit to 1, otherwise the Link is cleared. The contents of "y" are unchanged.	ADD y	0110/30
Twos Complement Add Add the contents of location "y" to the AC following the rules of twos complement arithmetic. A carry out of the high order bit of the AC complements the Link bit. The contents of "y" are unchanged.	TAD y	0111/34
Exclusive Or Exclusive-or the contents of location "y" with the AC and leave the result in the AC. The contents of "y" are unchanged.	XOR y	0101/24
Skip if AC differs Compare the contents of location "y" with the contents of the AC. If they differ, increment the PC to skip the next instruction. If they are the same, go on to the next instruction. The contents of "y" are unchanged.	SAD y	1011/54
Increment and Skip If Zero Increment the contents of location "y" by one. If this results in a zero in location "y", increment the PC to skip the next instruction. If the contents of "y", after being incremented, are not zero, go on to the next instruction. The contents of the AC are unchanged.	ISZ y	1001/44

Execute XCT y 1000/40  
Execute the instruction stored at location "y" as though it were stored at the location of the XCT. The contents of the PC are unchanged unless "y" contains a JMS, JMP, SAD etc.

The Extended Arithmetic Element instructions perform the more complex arithmetic operations such as long shifts, multiply, divide, and normalize. All arithmetic operations performed by the EAE assume ones complement number representation. The first group of EAE instructions deals with simple register operations with the EAE registers. The word format is as follows:

<u>Instruction</u>	<u>Mnemonic</u>	<u>Octal Code</u>
Or SC with AC	OSC	640001
Inclusively or the contents of the SC with the AC and leave the result in the AC. The contents of the SC are unchanged.		
Or MQ With AC	OMQ	640002
Inclusively or the contents of the MQ with the AC and leave the result in the AC. The contents of the MQ are unchanged.		
Complement MQ	CMQ	640004
Invert each bit of the MQ.		

Load AC from SC                      LACS                      641001  
Copy the SC into the low order 6 bits of the AC. Clear the high order 12 bits of the AC and leave the SC unchanged.

Load AC from MQ                      LACQ                      641002  
Copy the MQ into the AC. The contents of the MQ are unchanged.

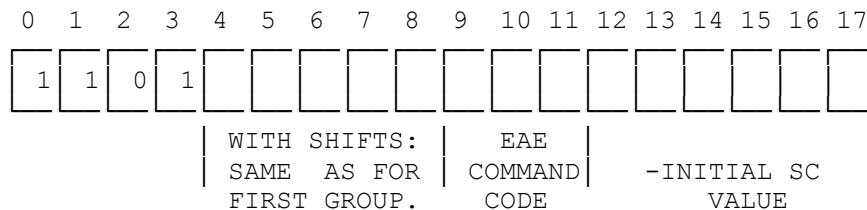
Clear MQ                              CLQ                      650000  
Clear the MQ to 000000.

Absolute Value of AC                  ABS                      644000  
Complement the AC if the sign bit is a 1 (AC is negative).

Get Sign and Magnitude              GSM                      664000  
Copy the sign bit of the AC into the Link and then complement the AC if the sign bit is 1 (AC is negative).

Load MQ                              LMQ                      652000  
Copy the AC into the MQ. The contents of the AC are unchanged.

The second group of EAE instructions perform repeated operations using the step counter (SC). At the beginning of execution the step counter is initialized to the two's complement of the low order six bits of the instruction word. The instruction then proceeds incrementing the SC until it reaches 0. The instruction format is:



Instruction	Mnemonic	Octal Code
Long Right Shift	LRS n	640500+n
Shift the AC and MQ as one 36 bit register n bits to the right. Bits shifted out of the bottom of the AC move into the top of the MQ. Bits shifted out of the bottom of the MQ are lost. The contents of the Link remains unchanged and enters the high order end of the AC at each step to replace the vacated bits.		
Long Right Shift Signed	LRSS n	660500+n
Copy the sign bit of the AC into the link at event time 1. Then proceed as with LRS, this time propagating the original sign bit onto the vacated bits of the AC.		
Long Left Shift	LLS n	640600+n
Shift the AC and MQ as one 36 bit register n bits to the left. Bits shifted out the top of the AC are lost. Bits shifted out the top of the MQ enter the bottom of the AC. The Link bit is		

unchanged and enters the low end of the MQ at each step to replace the vacated bits.

Long Left Shift Signed            LLSS n            660600+n  
Copy the sign bit of the AC into the Link at event time 1. Then proceed as with LLS, this time propagating the original sign bit into the vacated bits of the MQ (one's complement arithmetic).

Accumulator Left Shift            ALS n            640700+n  
Shift the contents of the AC n bits to the left. Bits shifted out of the top of the AC are lost. The link bit enters the low end of the AC at each step to replace the vacated bits. The Link and the MQ are unchanged.

Accumulator Left Shift Signed    ALSS n            660700+n  
Copy the sign bit of the AC into the Link at event time 1. Then proceed as with ALS, this time propagating the original sign bit into the vacated low order bits of the AC (ones complement arithmetic).

Normalize                            NORM            640444  
Shift the AC and MQ left until either the high order two bits of the AC do not match or the shift count runs out after 36 (decimal) steps. The SC will then contain the number of bits shifted minus 44 (octal). The actual number of bits shifted can then be obtained by executing an LACS instruction and then adding 777744 (octal) to the AC. Note that the SC value must be read out before executing another instruction which alters the SC.

Normalize signed                    NORMS            660444  
Copy the sign bit of the AC into the Link at event time 1. Then proceed as with NORM.

Multiply                            MULS            657122  
Multiply the ones complement number in the AC by the contents of the memory location immediately after the MULS instruction. This memory location is taken as containing the absolute value of the number to multiply by. The sign of this value must be entered into the Link previous to executing the MULS. The resultant 36 bit ones complement product is left in the AC (high order half) and MQ (low order half). The location containing the multiplier is unchanged. The next instruction executed will be two locations after the MULS instruction. A typical sequence to multiply numbers from locations X and Y would then be:

LAC	X	pick up X
GSM		magnitude in AC, sign in L
DAC	*+3	store magnitude
LAC	Y	pick up Y
MULS		multiply
DC	0	X  stored here
...		next instruction

## Divide

DIVS

644323

Divide the 36 bit ones complement value contained in the AC (high order half) and MQ (low order half) by the contents of the memory location immediately after the DIVS instruction. This location is taken as containing the absolute value of the divisor. The sign of this value must be placed in the Link previous to execution of the DIVS instruction. If the divide is successful, the link will be cleared, the remainder will be in the AC and the quotient in the MQ. If a divide overflow occurred, (e.g. division by zero) the link will be set to 1 and the AC and MQ will be meaningless. A typical sequence to divide the 36 bit number from locations YHIGH, YLOW by the number in location X would be:

LAC	X	pick up X
GSM		magnitude in AC, sign in L
DAC	*+5	store magnitude
LAC	YLOW	pick up low half
LMQ		put it in MQ
LAC	YHIGH	pick up high half
DIVS		divide
DC	0	X  stored here
SZL		overflow?
JMP	OFLO	yes.
...		next instruction

## APPENDIX B

### 339 Display Control Instructions

The 339 display control is a separate processor which executes its own instruction set. Since it will ignore the high order six bits of each PDP-9 memory word the programmer may use this area for flags. The high order six bits of the first word of each display leaf, for example, contain, by system convention, the leaf class number.

At any time, the 339 can be in "control" state or in "data" state. (All leaves start out in control state.) the instruction word interpretation is different for the two states.

#### B.1 Control State Instructions

##### B.1.1 Parameter (octal 0xxx)

This instruction is a "load immediate" of three global control registers in the 339. The current contents of these registers affect the production of the picture when the 339 is in data state.

##### Intensity (3 bits)

The contents of the intensity register control the brightness of the displayed picture. Its contents can be from 0 (off) to 7 (highest intensity). In order for the light pen to be able to see light from a leaf, it must be drawn in intensity 4 or greater. Upon entry to a leaf, the intensity register contains 7. Op codes to set the intensity register are:

INT0	0010
INT1	0011
INT2	0012
.	
.	
.	
INT7	0017

##### Light Pen Enable (1 bit)

This bit controls whether the light pen is disabled (bit clear) or enabled (bit set). In order to get a light pen hit on a displayed picture, the light pen enable bit must be set while the picture is being drawn by the 339. Upon entry to a leaf the bit is clear. Therefore the user must place a LPON instruction in the first word of those leaves where he wishes to get light pen hits. Op codes to change the light pen enable bit are:

LPON	0060
LPOF	0040

##### Scale (2 bits)

The contents of the scale register purportedly control the relative size of the displayed picture. While instructions in data state specify how many raster points are to be intensified in a line segment, the scale register specifies how far apart these points are to be. If the scale register contains 0 the points are about .01 inch apart (scale 1), if it contains 1 points are .02 inch apart (scale 2), if it contains 2 points are .04 inch apart (scale 4), and for 3 they are .08 inch apart (scale 8). Since the size of an intensified point is about .015 inches, lines drawn in scale 1 and scale 2 look reasonably continuous. In scale 4 they look quite grainy and scale 8 is good only to display dotted line segments. Op codes to set the scale register are:

SCL1	0400
SCL2	0500
SCL4	0600
SCL8	0700

Any combination of the three control registers may be set by one instruction word by inclusively or-ing the appropriate op codes together.

#### B.1.2 Mode (octal 1xxx)

This instruction puts the 339 into data state and selects one of several "modes" within data state. The modes differ in their interpretation of the words following the mode instruction. They essentially represent different packings of coordinate data within the 12 bit 339 word. Section B.2 describes these data formats. While other modes exist, VEC, SVEC, and INCR modes are recommended for use because they move the electron beam incrementally. Op codes to go to various data modes are:

INCR	1111
VEC	1121
SVEC	1141

#### B.1.3 Jump (octal 2xxx)

This is a two word instruction. The jump address is constructed from the low order 3 bits of the JUMP or PJMP instruction word and the low order 12 bits of the next sequential core location. The PDP-9 assembler will automatically perform this splitting up of the address with the following instruction sequence.

JUMP	ADDR
DC	ADDR

The JUMP instruction is simply a transfer of control within the 339. The PJMP instruction is a subroutine jump. The contents of the intensity, light pen enable, and scale registers along with the Display Address Counter are saved on a push down stack and then the jump is taken. Display leaves are entered in this manner. Op codes for these instructions are:



JUMP	200X
PJMP	201X

## B.1.4 Pop (octal 3xxx)

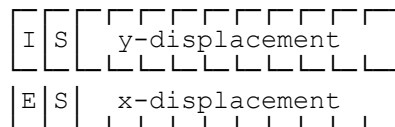
This instruction is a return from a push-jumped to subroutine. The scale, intensity, light pen enable registers and the Display Address Counter are restored from the push down stack. Since all display leaves are branched to by a PJMP instruction they must have a POP as their last instruction. Op code is:

POP 3000

## B.2 Data State Formats

The 339 is placed in "data" state by the instructions VEC, SVEC, and INCR. The particular bit interpretation for each data mode provides a means for "escaping" back to control state.

### B.2.1 VEC mode

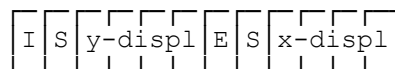


Data comes in pairs, a Y displacement and an X displacement in sign-magnitude form. These give the displacements (in raster points) relative to the current beam position along which to draw a line. If the I (intensify) bit is set, the line is intensified while being drawn. If the E (escape) bit is set, the display leaves data state after drawing the current vector and the next instruction is interpreted in control state. Many program bugs are caused by the user forgetting to set the escape bit at the end of a list of VEC mode words.

Subroutines to convert back and forth between twos complement binary and VEC mode signed-magnitude formats can be obtained by including the following line into the source stream:

COPY W339:LIB(200)

### B.2.2 SVEC mode



This is similar to VEC mode but both displacements are packed into one word and can be only four bits long.

## B.2.3 INCR mode



Each increment is of form:



Two increments are contained in each word, each telling a direction and a distance to move the beam. The direction is given by the three bit quantity DIR which represents:

```

    3 2 1
    4 * 0      (*=current beam position)
    5 6 7
  
```

The D (distance) bits stand for:

```

    00    move one raster point and escape
    10    move two raster points
    11    move three raster points
  
```

## APPENDIX C PDP-9 ASSEMBLY LANGUAGE

Assembly of user programs for the SEL system is done on the 360. The object module of the assembler is found in the file:

W339:ASR

Logical I/O units referenced are:

SCARDS	source program input
SPUNCH	PDP-9 absolute binary output
SPRINT	assembly listing
SERCOM	error comments

### C.1 Source Language Syntax

#### C.1.1 Comments

Source lines which begin with an asterisk (\*) are treated as comments. They are reproduced in the assembly listing but are otherwise ignored. Also the fourth field of each source line is a comment field and is treated similarly.

#### C.1.2 Constants

All constant numeric quantities are interpreted by the assembler as octal and should not be greater than 6 digits in length.

#### C.1.3 Fields

A source line is composed of up to four fields separated by one or more blanks. These are: Location, Operation, Operand, and Comment.

##### Location Field

A symbol found in the location field is assigned the value of the current location counter (unless the operation field contains the pseudo-op EQU, OPD, or OPDM). If the first character of the source line is blank, the location field is empty.

##### Operation Field

The operation field contains one of the following:

- 1) A pseudo-operation. Special action is taken according to which pseudo-op is present.
- 2) An instruction mnemonic which requires an operand. Indirect addressing is indicated here by appending an asterisk (\*) to the right of the mnemonic.

- 3) An operandless instruction or set of them separated by plus signs (+) signifying that they are to be exclusively or-ed together. Here the operand field is not present.

### Operand Field

The operand field, if present, contains symbols or octal constants which can be combined by the operations of addition (+) or subtraction (-). An asterisk stands for the address of the current location. Literals are denoted by an equal sign (=) appended to the left of the operand field. The low order 13 bits of this evaluated expression are or-ed with the operation field.

### C.2 Errors

Errors in the source stream are flagged by a one character error flag to the left of the line in the assembly listing. The symbol #ERROR in the cross reference listing will be referenced by all flagged lines. The error flags stand for:

U	Undefined symbol
S	Syntax error
M	Multiply defined symbol
C	invalid Character in line
P	direct reference across core banks
O	undefined or invalid Operation
L	missing or invalid Label

### C.3 Assembler Pseudo-operations

The following pseudo-operation codes may be written in the operation field of a source line to perform special functions. They do not, in general, produce any machine instructions, they just control the assembly.

DC	A word containing the value of the expression in the operand field is produced.
TITLE	The character string which follows this pseudo-op is printed as a heading on each page of the assembly listing.
EJECT	The assembly listing is advanced to the top of the next page.
SPACE	A blank line is inserted in the assembly listing
COPY	Copy the contents of the file named in the operation field into the source stream.
END	End of source stream.

For the following pseudo-ops, all symbols which appear in the operation field must be predefined.

- ORG    The location counter within the assembler is set to the value of the expression in the operand field.
- DS    The value of the expression in the operand field is added to the location counter within the assembler.
- EQU   The symbol in the location field is assigned the value of the expression in the operand field.
- OPD   The operandless instruction mnemonic in the location field is assigned the value of the expression in the operand field.
- OPDM The operandless instruction mnemonic in the location field is assigned the value of the expression in the operand field.